The Unexpected side of Logic Programming

MSc Thesis *(Afstudeerscriptie)*

written by

Lucrezia Maddalena Mosconi

under the supervision of **Dr. Ronald de Haan** and **Dr. Giovanni Sileno**, and submitted to the Examinations Board in partial fullfilment of the requirements for the degree of

MSc in Logic

at the *Universiteit van Amsterdam.*

Date of the public defence:	**Members of the Thesis Committee:**
August 25, 2025	Prof. Maria Aloni (Chair)
	Dr. Gregor Benkhe
	Dr. Malvin Gattinger
	Dr. Ronald de Haan (Supervisor)
	Dr. Giovanni Sileno (Supervisor)

Institute for Logic, Language and Computation

Abstract

Through this thesis, we investigate computational approaches to surprise. We define a novel computational framework, CompLogDL, building on an existing blueprint by Sileno et al. [1]: both rest on the Simplicity Theoretic definition of surprise, according to which an event is unexpected if it is more complex to generate than it is to describe. Our contribution through this thesis is two-fold.

At a conceptual level, we provide a formal argument against the received view of surprise, arguing that surprise ought not to be defined in terms of low probabilities. We first show that such a definition is neither necessary nor sufficient, and then that probabilistic interpretations encounter severe difficulties when predicting surprise. We then observe that this difficulties can easily overcome by a Simplicity-based definition of surprise, according to which an event is unexpected if it is more complex to generate than it is to describe. On this basis, we define a novel computational framework, CompLogDL, informed by a previous proposal by Sileno et al. [1]. At a formal level, we introduce the syntax and semantics of CompLogDL: its inference mechanism is made of two separate components, one computing the description complexity of an instance and the other computing its generation complexity. We specify each inference task and analyze their complexity in detail.

Contents

Chapter 1

Introduction

> "*Scientists have calculated that the chances of something so patently absurd actually existing are millions to one. But magicians have calculated that million-to-one chances crop up nine times out of ten.*"
>
> — Terry Pratchett, Mort

Surprise is a universal human experience, whether induced by an unforeseen plot twist in a murder mystery, or by meeting an old friend on the street. Due to its pervasiveness, surprise has been a topic of research in psychology and related disciplines for a while: the first characterisations of surprise as a mental and behavioural phenomenon date back to Aristotle [2].

Vast research has been carried out to characterise this cognitive phenomena and its ramifications, however the question of when and how an agent remains surprised is still open. A brief review of the literature (see [3] for an introduction) regarding this reveals that the commonly accepted view considers surprise as related to expectation [4–6]: a surprising event is one that is unexpected, and expectation is characterised in terms of probabilities. That is, an event is unexpected if it is a low probability event, and viceversa.

Though this mainstream view has been considered the default, alternative theories have been developed which reject the idea that low probability is the identifier of surprise. One of such alternatives is Simplicity Theory, first introduced by Dessalles [7] to explain narrative interest in humans: this theory hypothesizes that an event is unexpected if it is simpler to describe than it is to generate. The origins of Simplicity Theory are rooted in Algorithmic Information Theory, and in particular Kolmogorov Complexity [8], as this framework allows one to consider the complexity of a given event not in terms of the probability distribution that generated it, but in terms of its shortest description.

The framework of Simplicity Theory has found applications in a wide range of areas from Artificial Intelligence to Cognitive Psychology: it has been applied in the context of subgraph mining [9], smart-home design [10] to analogical reasoning [11]. At the same time, experimental results have confirmed its predictions regarding a variety of cognitive phenomena, such as narrative interest [7], near-miss effects [12], coincidences [13], and moral responsibility [14].

A notable application of Simplicity Theory which has been proposed is in the context of declarative programming [1]: a blueprint for a novel computational framework was proposed, called CompLog. Its purpose is to compute the unexpectedness of a given situation. This work is inspired by probabilitistic forms of logic programming such as ProbLog [15], whose

task is to compute the probability of a given event, given a model of the world specified as a set of facts and probabilistic dependencies between them. The version of CompLog presented in [1] served to motivate this novel inference task presenting an initial implementation.

Through this thesis, we expand on this existing work in the following two directions:

- **Conceptual**: We provide a formal argument against the received view of surprise, showing that low probability is not a necessary nor sufficient condition for an event to be surprising. We then argue that, in order to predict surprise, we should drop the assumption that an agent's model of the world ought to be represented via a probability space: we do so by considering the closely related phenomena of awareness growth.
- **Formal**: We supplement the blueprint given in [1] by providing a precise syntax and semantics for an updated version of CompLog, which we call CompLogDL. This novel framweork revisits some of the earlier design choices, while maintaining the original intuition: its inference inference mechanism is composed of two separate components; one will compute the descriptive complexity of an instance, while the other will compute its generation complexity. Unexpectedness is then defined as the difference between these two computations. After providing a formal definition of each computation, we also analyse their complexity.

The rest of this work is structured as follows. In Chapter 2, we provide much of the conceptual justification for moving to assessing the surprise of an event following simplicity theory, by showing that probability theory, and especially bayesian epistemic models, encounter severe difficulties. In Section 2.3, we give arguments for why low probability is not neccessary, nor sufficient to identify surprise, while in Section 2.4 we give an argument which relates surprise to the phenomena of awareness growth. In Chapter 3, we provide a brief introduction to description logics and computational complexity theory, introducing those concepts which we will refer to in the following two chapters. In Chapter 4, we introduce the CompLogDL architecture: Section 4.1 specifies how the description complexity is computed, starting from an annotated knowledge base, while Section 4.2 specifies how the generation complexity is computed by specifying a set of causal dependencies; Section 4.3 will define a CompLogDL program and the notion of conditional unexpectedness, while Section 4.4 briefly compares CompLogDL to CompLog. Finally, in Chapter 5 we present some complexity results regarding descriptive and generation complexity computations, respectively. We conclude with Chapter 6, where we consider limitations of our current approach and indicate possible directions for future work.

Chapter 2

Predicting Surprise

Probability theory underlies much of modern science— from statistics to cognitive modeling and decision theory ([16–18]), as it provides a rigorous framework for quantifying uncertainty, updating beliefs, and making predictions. Through this chapter, we examine the gap between probabilistic (and in particular Bayesian) epistemic models and the subjective feeling of unexpectedness, to then present an alternative account that resolves some of probability theory's shortcomings.

After a brief introduction to Bayesian Epistemology in Section 2.1, in Section 2.2 we give a cognitive-behavioural definition of surprise as a schema-discrepancy detector. We then show in Section 2.3 why low probability alone does not suffice to characterize unexpectedness. Section 2.4 critiques the closed-world assumption in Bayesian reasoning, highlighting the dynamics of awareness growth and the recalibration of surprise. Finally, Section 2.5 introduces Simplicity Theory, which defines unexpectedness in terms of complexity drops, and demonstrates how this new metric avoids the pitfalls of bayesian epistemic models.

2.1 Bayesian Epistemology

We briefly introduce those notions of probability theory and to which we will be referring through this chapter. These are the standard definitions of concepts as presented in [19], but a variety of introductions to the field are available (see for example [20–22]).

Definition 2.1. (**Probability Space**) A probability space is a tuple $(P, \Omega, \mathbf{F})$, where Ω is a non-empty set of events, $\mathbf{F}$ is a σ-algebra on Ω and $P : \mathbf{F} \to \mathbb{R}$ is a a measurable function (called a probability measure) which obeys the following three axioms:
1. (Non-negativity): $P(A) \geq 0$
2. (Normalisation): $P(\Omega) = 1$
3. (Finite Additivity): $P(A \cup B) = P(A) + P(B)\ \forall A, B \in \mathbf{F}$ such that $A \cap B = \emptyset$.

One can then define a *conditional* probability measure for any given event E:

Definition 2.2. (**Conditional Probability**) Let $(P, \Omega, \mathbf{F})$ be a probability space, and let $E \in \mathbf{F}$. The conditional probability $p(\cdot \mid E)$ of any event $H \in \mathbf{F}$ given E is defined as

$$P(H|E) = \frac{P(H \cap E)}{P(E)}$$

From the above definition, we immediately obtain the following:

Theorem 2.3. (**Bayes Theorem**) The conditional probability of H given E can be computed by

$$P(H|E) = \frac{P(E|H)P(H)}{P(E)}$$

This result stands at the basis of so called Bayesian methods, which are prominent within statistics as well as formal epistemology ([23–25]). Within the latter, a Bayesian approach is characterised by a commitment to the claim that an agent's model of the world (or epistemic state) may be represented by a function $p : \Omega \to [0, 1]$ which assings a degree of belief to every proposition the agent is aware of (represented by the set Ω). What we will present here is a classical version of Bayesian Epistemology, which we will call *orthodox*, following De Canson [26]. An orthodox Bayesian[1] is committed to the following three principles:

1. **Probabilism**: The function $P : \Omega \to [0, 1]$ should be a probability function, meaning that the tuple $(P(\cdot), \Omega, \mathbf{F})$ ought to be a probability space, where $\mathbf{F}$ is a σ-Algebra on Ω.
2. **Conditionalisation**: Upon learning that a proposition $E \in \Omega$ is true, the agent's epistemic state ought to be updated to $(P(\cdot \mid E), \Omega, \mathbf{F})$.
3. **Extremality**: A proposition is assigned an *extremal* credence (either 0 or 1) if and only if said proposition is logically necessary.

The classical arguments for these norms are given in terms of what are commonly called Dutch book arguments[2], a version of which can be briefly stated as follows:.

If an agent's degrees of belief do not conform to the probability axioms, a wager could always be constructed such that the agent's expected gain (computed according to the agent's utility function) is always strictly less than zero. In other words, there is a sequence of bets which the agent is sure to lose[3].

It is beyond the scope of this section to provide a full critique of Bayesian Epistemology, and while empirical studies (see for example [30]) question whether Dutch Book arguments have any standing at all, we will grant that such a model is a plausible formalisation of a normative framework, that is: an *ideal* rational agent ought to be modelled in accordance with orthodox Bayesianism.

Now that the orthodox Bayesian view has been introduced, we will begin showing why this framework is not well suited to predict the unexpectedness of a given event.

[1] This categorisation is not maximally specific: amongst orthodox Bayesians, approaches differ widely, from the inclusion of additional principles beyond those presented here, to their interpretation and justification for such principles.

[2] An alternative justification is given in terms of *accuracy arguments*, such as the ones presented by Leitgeb and Pettigrew [27]

[3] The argument in its original form is given in detail in Ramsey [28] or De Finetti [29]

2.2 Defining surprise

When thinking of possible outcomes of a given event, we seem to have a decent intuitive understanding of which possibilities would be most unexpected. For example, we would all seem to agree on the claim that the sight of a pink elephant roaming the street is an unexpected event, but what makes it so?

In common usage, surprise is defined as the emotion elicited by unexpected events of various kinds [31], from a plot twist in a novel to an unforeseen outcome at the elections. Within the academic community, psychologists and philosophers alike ([32, 33]) long researched the feeling of surprise that unexpected situations prompt in agents.

Theories regarding this universal experience range widely (see [34] for a survey): some hold that surprise is a necessary condition for learning and development, and suggest that surprise results from a cognitive appraisal of an eliciting event [35]. The theory that is perhaps most aligned with our common sense understanding is the cognitive-evolutionary model of surprise, which defines surprise as a non-propositional signal, resulting by an unconsciously operating schema discrepancy detector: surprise is elicited by events that deviate from a *schema* [6].

This definition is better understood in the context of what is commonly known as schema theory [36]: human thought and action is hypothesized to be largely dependent on a complex set of belief structures which we call *schemas*. These can be thought of as the cognitive representation of what are otherwise humans' informal theories about the world: their function is to serve as an interpretation of past and future events, as well as predictions. As a schema's function is to guide action, it must be at least congruent with the input we receive from the world around us. The suprise mechanism in humans plays a central role within this theoretical framework. As we continuously compare the current schemas with the input we receive from the world newly acquired information, the surprise mechanism works as a schema discrepancy detector [3]: as long as no variation is detected between our schema and the input, we need not revise our schema.

While it should be noted that the cognitive framework of schema theory has not gone unchallenged ([37–39]), schema theory is a well established and active area of research within cognitive psychology [40], providing us with a good starting point to define the notion of surprise as a schema discrepancy detection mechanism. Now that we have clarified what we take to be the definition of surprise, we can state the main claim of this chapter:

Proposition 2.4. Orthodox Bayesianism is not a good framework to predict when an event is surprising.

The following two sections will present two different arguments as to why Bayesian epistemology is not suited for this task.

- In Section 2.3, we will argue that the condition of being assigned a low probability is not sufficient, nor necessary, for an event to be unexpected.

- In Section 2.4, we discuss the problem of Awareness Growth and show that, since Orthodox Bayesianism is committed to *probabilism*, it is unfit to predict surprise.

2.3 Improbable $\neq$ surprising

Through this section, we will argue the following:

Thesis 1: Low probability is not necessary, nor sufficient to predict when an event is surprising.

We have mentioned earlier that surprise can be thought of as a schema discrepancy: we are surprised when the input we receive by the world varies from our representation of the world, or schema. At first glance, the framework of Bayesian Epistemology as presented above seems like a natural fit to characterise this incongruence, as it represents a way of assigning a certain probability mass to a series of possible events: events that we believe are unlikely to happen according to our representation of the world are assigned a low probability, while events that are in line with our model are assigned a higher probability.

Suppose that we are given a probability space $(P, \Omega, \mathbf{F})$ representing an agent's model of the world. We could define an event $S \subseteq \Omega$ as unexpected using the following principle:

S is unexpected if and only if it is a low probability event.

We argue that there exist low-probability events which would not be unexpected, and there exist unexpected events which would not have a low probability, meaning that the fact that an event is assigned a low probability is not necessary, nor sufficient, for such an event to be considered surprising. To show this, we will present three different scenarios in which the probabilities assigned to a given event do not match with its surprisal value, namely Equiprobable Events, Unexpected Consequences and Unknown Unknowns.

Equiprobable Events

Consider the scenario of a fair lottery on 100,000 tickets. If we are to model this situation via a probability space, we would define $P : \Omega \to [0, 1]$ to be a uniform probability, such that for any ticket, the probability that it is extracted is $\frac{1}{100000}$. If we take the thesis that an event is unexpected if its probability is low, then any ticket out of the 100000 - would be equally unexpected.

While it may be the case that any ticket has the same probability of being extracted, if the ticket 11111 were to be the winner, it would strike us as a more shocking event then if the ticket number 75296 were selected, even though from the point of view of the probability distribution via which the scenario is modelled, these two events are by definition equiprobable.

What may seem a rather informal intuition regarding which equiprobable events we humans find more surprising has been confirmed by experimental results, which suggest how we judge certain combinations, such as 11111 or 012345, to be more unexpected ([41]). As an example, consider the more recent experiment performed in [42], where a group of participants were asked to choose 2 among 14 possible combinations of six numbers ranging from 1 to

49 (possible entries in a Lottery). Though all combinations are equiprobable, all 26 subjects showed strong avoidance of the two simplest ones, namely [1 2 3 4 5 6] and [34 35 36 37 38 39]: if they had truly chosen randomly, there would be a probability of barely 3^{10-4} that they would have consistently avoided these two combinations.

Unexpected Consequences

Conversely, it is not the case that every event which is assigned high probability by a given agent is expected: as an example, consider the postulates of Einstein's Relativity theory. Within the scientific community at least, Einstein's model of physics is commonly accepted, meaning it is an established part of physicist's model of the world or, in our terms, it is a part of their schema. Nevertheless, many empirical tests were carried out to confirm Relativity Theory's predictions [43], and it would be incorrect to state that the fact that these empirical tests confirmed Einstein's theory was unsurprising [44]. In strictly probabilitstic terms, if we conditionalise on Einstein's theory (i.e. we assign probability 1 to the fact that Relativity theory is a correct model of our world), then the probability that those empirical tests would come out positive is 1, as they are logical consequences of the theory.

If we are to take seriously the claim that only low probability events can be considered unexpected, then we need to give up any surprise about a given event taking place, whenever said event is a postulate (or logical consequence) of a theory or claim we already believe to be true, since by conditioning on the claims we are certain of, their postulates are bound to be extremely likely, if not certain.

While General Relativity's prediction of the perihelion of Mercury may seem like a limit example, it is more representative of our experience than one may think. Think for example of the way progress within the field of mathematics is made: though we may be genuinely convinced in the correctness of a theorem, its logical consequences may still be considered surprising. As an example, consider the commonly accepted axiom of choice: insofar as we accept it as an axiom, we will assign probability one to it. Nevertheless, it is not true that we would not be surprised at the logical consequences which follow from the standard ZFC model, some of which may feel like a genuine discovery (think for example of the Banach-Tarski paradox, [45]). If low-probability is the defining criteria for unexpectedness, then logical consequences of accepted theories are by definition unsurprising: as they are consequences of a result we believe to be true, their probability must be 1.

Unknown Unknowns

We have discussed cases in which low probability events may be unsurprising, and similarly cases in which high probability events may be surprising, but there is a third case which is worth considering: what about those events to which we don't assign a probability at all?

As humans, our knowledge of the world around us is more often than not incomplete: we may possess faulty, imprecise beliefs about the state of the world which leads us, for example, to erroneously assess how likely an event is to happen. These kinds of mistakes however

regard those events which already belonged to the set Ω in our probability space $(P, \Omega, P(\Omega))$: we may have erroneously assigned them probability 0 (we thought they were never going to happen, but they did), or we might have been over confident about them happening (and they did not). These events are those towards which we had a dispositional attitude, though erroneous.

There is however a deeper sense in which our model of the world may be disaligned with the reality around us: there are events which are not assigned a probability value at all: they do not belong to the set Ω of possible outcomes, and thus they simply are not in the range of the probability function P: these are the events that we are not aware of, or *unknown unknowns* [46]. To give a concrete example, consider the advent of the internet, from the point of view of an agent located (spatio-temporally) in the 1920′s. While science fiction may have suggested flying cars and the rapid techno-industrial progress of the time may have led one to think that in the future we could have even greater technological advancements, but the omnipresence of a technological medium such as the internet would have likely not been entertained at all: one back then would have not deemed impossible, she just would not have considered it as a possibility at all.

It should be clear that if by definiton only those events which are assigned low probability are unexpected, then these events cannot be such, yet we would want to say that these events are precisely those which we expect the least: we weren't even aware of the possibility of them being events at all. It should be stressed that these events are fundamentally different from an epistemic point of view from those events which we assign probability zero: we possess a dispositional attitude towards the latter (we have considered them and believe them to be impossible), but not towards the former (we have not previously entertained the possibility of such events taking place).

While in the last two sections we have argued that a surprise in terms of low probabilities is unsatisfactory, we now move onto a different claim: even if we could associate surprising events with events of low probability, the Bayesian's[4] reliance on a closed space of events makes it unfit to predict surprise.

2.4 An argument from Awareness Growth

Through this section, we argue the following:

Thesis 2: The Bayesian interpretation fails to model surprise because it mandates probabilism.

Given any representation of the world, humans often acquire information of which they were previously not aware, and refine the space of events by also adding new instances and properties which they did not previously consider. This experience leads to an increase in the size of the space of events, which orthodox Bayesian methods as presented so far do not permit:

[4]From now on, we will use "Bayesian" to mean *orthodox* Bayesian, as defined above.

the almost universal assumption for this model is that the set Ω, representing the possibilities an agent entertains, is *fixed* [23].

In Section 2.4.1, we formally define this problem, which in the literature is called "Awareness Growth" (see for example [26]), and evaluate the two standard Bayesian responses to it. In Section 2.4.2, we show that neither is satisfactory when Bayesian models are tasked with predicting surprise, and that the reason for this is the fundamental reliance on a closed event space.

2.4.1 Awareness Growth

Awareness growth describes situations in which an agent comes to consider propositions they were not entertaining before, resulting in an increase of the range of propositions in an agent's algebra [47]. This leads to a change in the agent's credences that is not due to observing evidence, and thus, even for a Bayesian, it need not take place by conditionalization: these are scenarios in which one might come to know about a previously unconsidered state of affairs. This may be an previously *unknown* event, like the one described in Section 2.3, but it doesn't have to be; growth in awareness is a rather mundane experience: as Earman ([48], p.192) puts it "new observations, even of familiar scenes; conversations with friends; idle speculations; dreams—all of these and more are constantly introducing heretofore unarticulated possibilities".

Within philosophy of science, this problem as not gone unnoticed (see Salmon ([49]), and Earman [48]), and it is often referred to as the New Theory Problem: suppose that we come to realise the possibility of a new hypothesis H^*, which we had previously not considered: how can we confirm H^* based on evidence we are already certain of, and how are we to adjust our credences to other existing hypotheses?

Within Bayesian epistemology, interest in Awareness Growth has recently been rekindled: if Bayesianism is to be taken as an appropriate representation of rationality in human agents, it *must* have a way of accomodating this phenomena.

Concrete everyday instances of this phenomena include scenarios where agents learn new facts, or where they recall something they had forgotten. As an example, consider an agent, Anna, who is currently considering what her mother might have gotten her as a birthday present: she assigns a credence of 0.5 to the proposition "my mother got me tickets for a festival" and 0.5 to "my mother got me movie tickets". Upon further reflection, Anna realises that the movie could be either a thriller or an action movie , and that her mother might have decided to get her tickets to a musical instead. What credence is Anna to assign to the different propositions, and how are we to model her epistemic state?

Reverse Bayesianism

A common approach put forward by the Bayesians is the one presented by Bradley ([50]), which he fittingly calls Reverse Bayesianism[5]. Following this approach, we ought to model

[5]Other proponents of this view include Karni and Vierø [51], as well as [52]

the results of the above change in awareness either by refining or expanding on an agent's algebra. If we are refining over a coarse grained proposition already present agent's algebra (eg. Anna realises "movie" could be "thriller" or "action") the credence in the coarse grained proposition should not change, but its credal mass (in this case, 0.5) should be split among the fine grained propositions. If instead we are expanding the algebra by including a proposition disjoint from the ones already considering (eg. Anna realises "musical" is an alternative to "movie" or "festival"), any credence can be assigned to such proposition, as long as all the propositions already in the algebra ("movie", "festival") are kept in the same proportion as before and the credences of all propositions in the new algebra sum up to 1.

We can formalise the Reverse Bayesian approach to the problem of awareness growth following Steele and Stefannson [53]. Let $(P, \Omega, \mathbf{F})$ be an agent's probability space, where Ω is the set of basic propositions of which the agent is aware. Then we say that an agent's awareness *grows* when the set of propositions Ω shifts to Ω^+, where $\Omega \subset \Omega^+$: this in turn means that the agent will have to move to a new probability function $P^+ : \Omega^+ \to [0, 1]$: Reverse Bayesianism requires can be defined in terms of the constraints that one applies to the new probability distribution P^+, which can be formalised as follows.

Definition 2.5. (**Reverse Bayesianism**) For any $A, B \in \Omega$,

$$\frac{P(A)}{P(B)} = \frac{P^+(A)}{P^+(B)}$$

This means that the new probability distribution P^+ must maintain the same proportions between the probabilities assigned to propositions of which an agent was previously aware. Amongst reverse Bayesians, some, such as Karni [51] require that the principle be applied only to outcomes (i.e. the elements the set Ω), while others such as Bradley [50] do not apply this restriction and instead demand that it be applied to theories as well (the whole event space $\mathbf{F}$).

The problem with reverse Bayesianism (indipendently of whether apply the principle beyond basic propositions in Ω) is that it fails to satisfactorily explain how we are to make sense of scenarios where refinement and expansion take place at the same time. As both Mathani [54] and De Canson [26] have pointed out, reverse Bayesianism requires that a rational agent undergoing awareness growth must fix his credence in the propositions they were already aware of according to their credences in those propositions before undergoing awareness growth. Mathani [54] shows this via two counterexamples: we introduce the second ([54] , p.8).

Example 2.6. (**Other Tails**) A fair UK ten pence coin is about to be tossed, and our agent has a credence of 0.5 that it will land HEADS, and a credence of 0.5 that it will land TAILS. The agent is also convinced that the tails side always has a LION engraved, so naturally it holds a credence of 0.5 that it will land on the face with a LION engraving: relative to our

agent's awareness state, the propositions LION and TAILS are equivalent. Now suppose that our agent realises that ten pence coins, occasionally, may have something other than a lion engraved on the tails side, namely an engraving of STONEHENGE. How should the credences in these propositions be updated?

HEADS	TAILS	
	LION	STONEHENGE
?	?	?

Figure 1: New awareness state, [54]

If we are to follow reverse Bayesianism, one must give TAILS and LION the same credence, following growth in awareness. But since LION and STONEHENGE are incompatible, and STONEHENGE entails TAILS, we must give STONEHENGE a credence of zero: clearly, this solution is unsatisfactory.[6]

Catch-all

Alternative approaches to awareness growth mandate the inclusion in an agent's algebra of what is commonly called the "Catch-all Hypothesis", first introduced by Shimony [55]: roughly, we can think of a catch all-hypothesis as the proposition stating "something other than all of the propositions I am considering".

By mandating the inclusion of such a hypothesis, what we mean is that an agent's probability function should always assign a small, but non-zero probability to the theory $\neg X_1 \wedge \neg X_2 \wedge ...\neg X_n$, where $X_1, ..., X_n$ are all of the propositions appearing in Ω.

This "Catch-all" solution to the problem of awareness growth has been considered extensively ([48, 56, 57]), but it is commonly seen as defective: the common view regarding catch all propositions is that "there is no particularly principled way to decide how much … probability should be assigned to the[m]", as pointed out by Henderson ([58], p.190). Catch-all propositions are considered to be *uniquely* intractable, as argued by Wenmackers et al.: "Whereas the other [propositions] come with a—possibly very intricate—theoretical background story, the catch-all has no content other than "none of the explicitly formulated hypotheses"" ([59], pp.1233). The argument then is, that insofar as such propositions are unintelligible (and they are the *only* proposition of this kind), they should be disregarded entirely.

Nevertheless, the intractability of catch-all hypotheses is not unique: we routinely entertain propositions that could also be deemed unintelligible, as has been pointed out by De Canson ([26], p.13-14) . Consider an agent untrained in the field of physics, who is assessing the probability of a given theory of fundamental physics being the correct one: as I know that my knowledge is limited, the three possibilities are "super-symmetry", "quantum gravity",

[6]In the remainder of the article, Mathani shows that even by placing further restrictions on the Reverse Bayesianism principle, we fail to produce satisfactory results: principled approaches which follow this line of reasoning inevitably lead to counter-intuitive conclusions.

and “something else”. As our agent is not a theoretical physicist, their understanding of the proposition “quantum gravity” is not different from their understanding of the proposition “something else”: they are both equally unintelligible.

If we are to accept the introduction of a Catch-all hypothesis to model awareness growth, then we must obey to the following procedure. Whenever we come to realise an additional proposition $X_{n+1} \notin \Omega$ we were not aware of (what the reverse bayesian would call awareness growth by expansion), we must refine the Catch-all proposition $\neg X_1 \wedge \neg X_2 \wedge ...\neg X_n$ and partition it into two different events: the possibility X_{n+1} and the alternative $\neg X_1 \wedge \neg X_2 \wedge ...\neg X_n \wedge \neg X_{n+1}$. We can then redistribute our credence freely amongst the $n+2$ possibilities, making sure that the new Catch all hypothesis, $\neg X_1 \wedge \neg X_2 \wedge ...\neg X_n \wedge \neg X_{n+1}$ is assigned non-zero probability[7].

While this solution may be efficient at solving the problem of awareness growth, we maintain that, independently of which solution we adopt, the Bayesian approach fails to accurately predict unexpected events, and is thus unsatisfactory for our purposes.

2.4.2 Probabilism

We begin by recalling that, independently of whether we adopt a Catch-all hypothesis or a reverse Bayesian solution, the Bayesian approach clearly supports *probabilism*, as defined in Section 2.1. While this is a computationally efficient shortcut to take, which is easily justified when looking at rational (human or artificial) agents[8], it is an undesirable requirement to have if we want to detect which events are surprising.

Recall that, according to the thesis we presented, we say that an event is unexpected if and only if its probability is low. Now suppose, just like in the example in section Section 2.4.1, our agent Anna comes to realise scenarios she was previously not considering, that is: she moves from $(P, \Omega, P(\Omega))$ to $(P^+, \Omega^+, P(\Omega))$ - whether via reverse Bayesianism, or by refining the Catch-all hypothesis - as she comes to realise that the tickets she will get for her birthday could be not only for a movie or for a concert, but also for a musical. Suppose now that the lowest proability event, according to P, was the movie ticket event: according to our thesis, this event would have been the one surprising Anna the most.

As Anna finds herself having to redistribute the probability mass amongst three options instead of two, she must ensure that the function P^+ is indeed a probability measure: she cannot keep the probabilities assigned by P to *both* the events “movie tickets” and “concert tickets” unchanged, and just assign any value between 0 and 1 to the proposition “musical tickets”. Instead, she must find a way to redistribute her credal mass across the three options in a way that makes them sum to one.

What this naturally implies is that there will be situations in which the degree of unexpectedness of a given possibility (either “movie tickets”, or “concert tickets”) will have to change,

[7] While at the same time ensuring that we still obey all of the probability axioms.

[8] There are nevertheless critics of this principle even when it comes to modelling epistemic states: see for example Pollock [60].

even though the only thing that has changed is that Anna has realised that another possibility might take place as well. Clearly it is possible that by realising that another possibility exists, the scenarios we already considered might appear as more or less surprising, but it seems counteintuitive to demand a continuous readjustment of the degree to which a given option is unexpected.

Note that this problem is not dependent on the methods or rules by which we decide to revise our credences for propositions in Ω, and instead it depends solely on the requirements that $(P^+, \Omega^+, P(\Omega))$ must satisfy, in order to be called a probability space: as P^+ must conform to the probability axioms, the sum of the probabilities in Ω^+ must sum up to 1.

As probabilism is a main tenet of Orthodox Bayesianism, one must require that, whenever our (possibly incomplete) model of the world is updated to include more possibilities, then there is at least one option whose unexpectedness *must* change; said option will be more surprising, as its probability will become lower. If we aim to model surprise via a Bayesian model, and attempt to define surprise in terms of low probabilities, we cannot accomodate for the fact that the reasons for which we might find a possibility surprising may well be independent of the possible alternatives.

2.5 Simplicity Theory

In the subsections above we have presented reasons for which probability theory is not a good framework to individuate unexpected events. The aim of the following sections is to present the alternative framework of Simplicity Theory (ST) - in particular its definition of a situation's unexpectedness in terms of Complexity drops. We aim to show that this characterisation of unexpectedness is a better predictor of when an event is surprising to us humans, which in turn defines which events or situations we as humans find surprising.

Simplicity theory's main tenet is the claim that human beings are sensitive to drops in complexity [61]: we find a situation unexpected whenever it is simpler to describe than it is to bring about. This theory builds on the mathematical theory of Algorithmic Information Theory, relying on the computations of two different Kolmogorov Complexities. In Section 2.5.1, we will introduce the central notions of Algorithmic Information Theory which form the basis for this framework, while Section 2.5.2 will give a brief introduction to Simplicity Theory. Finally, Section 2.6 will show how ST addresses the difficulties that the Bayesian definition of surprise cannot overcome.

2.5.1 Kolmogorov Complexity

In the previous section, we addressed some of the limitations of Probability Theory, in particular the difficulties that we have in expressing the randomness or the unexpectedness of a single event, independently of the probability distribution that generated it. Similarly, Shannon Information theory will always assign to two equi-probable events the same level of informativeness [62], as the informativeness (that is, the entropy) of an event depends on the probability distribution that generates it.

Kolmogorov Complexity on the other hand allows us to quantify the informativeness of a given object independently of the sample distribution, by defining the informativeness of an object as the number of bits that are needed to fully describe the object. Here, we will provide a basic introduction to the central definitions[9] needed in order to grasp Simplicity Theory, but we refer the reader to [8] for an in-depth introduction to the field.

Formally, we can define the complexity of an object as follows:

Definition 2.7. (**Complexity**) Suppose that we have a set of objects S and assume some standard enumeration of objects $x \in S$ by natural numbers $n(x)$. We also have a specifying method f, which is a partial function assigning to each possible program a natural number $n \in \mathbb{N}$. Then for every $x \in S$, the complexity of object x with respect to the specifying method f is defined as:

$$C_{f(x)} = \min\{l(p\} : f(p) = n(x)\}$$

Where $l(p)$ is the length of the program p. Informally, the complexity of x with respect to f is the length of the shortest program p such that f outputs x when p is given as input.

Clearly, there exist infinitely many specifying methods $f_1, f_2, ...$ which we may use to specify objects in S. One way in which we may compare different specifying methods is by determining which function in a given class $\mathcal{C}$ of partially computable functions $f_1, f_2, ...$, is additively optimal:

Definition 2.8. (**Additive Optimality**) Let $\mathcal{C}$ be a class of partially computable functions, and let S be a set of objects. We say that a partial function f^* is additively optimal for $\mathcal{C}$ if $f^* \in C$ and for every $g \in C$ there is a constant $c_{\{f,g\}}$ such that: $C_f(x) \leq C_g(x) + c_{\{f,g\}}$ for all $x \in S$.

We can also define the conditional complexity of object x *given* object y as follows:

Definition 2.9. (**Conditional Complexity**) Let x, y, p be natural numbers. Any partial computable function φ, together with p and y, such that $\varphi(\langle y, p\rangle) = x$ is a **description** of x. The complexity C_φ of x **conditional to** $\boldsymbol{y}$ is defined as: $C_\varphi(x|y) = \min\{l(p) : \varphi\langle y, p\rangle) = x\}$, and we call p a program that computes x given y. $C_\varphi(x|y) = \infty$ if there is no such p.

We can recover the unconditional complexity of x as a special case of its conditional complexity by noting that $C_\varphi(x) = C_\varphi(x \mid \varepsilon)$, where ε is the empty string.

The central result which allows us to define Kolmogorov Complexity is the Invariance Theorem, which we give – without proof – below.

[9]The definitions used here are taken from [8]

Theorem 2.10. (**Invariance Theorem**) Let $\mathcal{C}$ be the class of partial computable functions, then there exists an additively optimal universal partial computable function φ_0, such that for all other universal partial computable functions φ_i, and for all x, y we have:

$$C_{\varphi_0}(x|y) \leq C_{\varphi_i}(x|y) + c_{\varphi_i}$$

where c_{φ_i} is a constant depending on φ_i but not on x or y.

What this means is not that the universal description method φ_0 gives the short description in each case, but that no other description method can improve on it infinitely often by more than a fixed constant. However, the following result, which we give without proof (see [8] for a discussion), severely constrains the applicability of such a computation:

Theorem 2.11. The function $C(x)$ as defined in Definition 2.7 is not computable. Moreover, no partial computable function $\varphi(x)$ defined on an infinite set of points can coincide with $C(x)$ over the whole of its domain of definition.

For the purposes of Simplicity Theory, we reason with resource bounded turing machines,just as us humans are limited by the computational capacity of our cognitive system. Even in the resource bounded case, we can provide similar invariance results, but the Kolmogorov Complexity computation is now computable, as we see below.

Bounded Kolmogorov Complexity

Until now we have considered unbounded computations, where the Turing Machine[10] we use to compute a string x has access to unlimited space and time resources. One can just as easily define a resource-bounded notion of Kolmogorov Complexity, where the time and space resources the Turing Machine has access to are limited (they might be finite, or be a function of input x).

Remark. Let φ_1,φ_2, ..., be an effective enumeration of partial computable functions. Let T_φ be a multitape Turing machine that computes φ. Let t, s be polynomial functions, and let $n = l(x)$ be the length of string x. If $T_\varphi(y)$ returns x using $t(n)$ steps and $s(n)$ tape cells, we write $\varphi^{\{t,s\}}(y) = x$

Definition 2.12. (**Conditional Complexity**) Let $x, y, p \in \mathbb{N}$. Every computable function φ together with p, y such that $\varphi(p, y) = x$ is a description of x. The resource-bounded Kolmogorov Complexity $C_\varphi^{\{s,t\}}$ of x with respect to y is defined by:

$$C_\varphi^{\{s,t\}} = \min\{l(p) \mid \varphi^{\{s,t\}}(p, y) = x\}$$

Even if we deal with resource bounds, we can still select a reference universal Turing machine, thanks to the resource-bounded invariance theorem:

[10]For a formal definition of a Turing Machine, see Definition 3.22

Theorem 2.13. (**Bounded Invariance Theorem**) There exists a universal partial computable function φ_0 such that for every other partial computable function φ, there is a constant c such that, for all x, y:

$$C^{\{ct\log t, cs\}}_{\{\varphi_0\}}(x|y) \leq C^{\{t,s\}}_{\varphi_{i(x|y)}} + c_i$$

Where the constant c_i depends on φ_i but not on x or y.

Note that, while the notion of *unbounded* Kolmogorov Complexity is uncomputable (see [8], p.127) , this is not so for the resource bounded case. The key observation here, which will form the basis for the discussion that follows, is that resource bounded Kolmogorov Complexity can be mapped to any domain, as long as one clearly defines the symbols and computations which one uses to compute: we can thus speak of a minimal program given a specific instance of a resource bounded Turing Machine.

2.5.2 Unexpectedness

We are now in a position to clearly define ST's main claim:

An event is unexpected if it is easier to describe than to generate [63].

According to Semplicity Theory, the unexpectedness of a given situation should be thought of as the difference between two properties of an object, its generation complexity, and its description complexity [7]. These two quantities are independent of each other, and are formalised as two different Kolmogorov Complexity computations which are performed on two different two different devices: one representing the World Model, and the other representing the Mind Model[11].

Formally, the two computations are performed by two different resource bounded Turing Machines, which given an event s, return its generation and description complexity respectively. We call the Turing Machine computing the complexity of causing a given instance the W-Machine ("World machine"): this machine will take as input an agent's representation of world. Similarly, we call the machine computing the complexity of describing a given instance uniquely the "O-Machine" ("Observation machine"): this machine will take as input an agent's knowledge. The W-Machine will compute the Kolmogorov Complexity of the parameters required to bring an instance about, while the O-Machine computes the Kolmogorov Complexity of an instance's unique description.

Definition 2.14. (**Generation Complexity**) Let s be an instance, then the generation complexity C^i_W of s is the minimal description of all parameters that have to be given for s to be brought about in the world - according to agent i's input to the W-machine.

[11] Note that, as humans are agents whose computational resources are finite, the Kolmogorov Complexities computations carried forward by the Turing Machines representing the World and the Mind model will be *resource bounded*, and thus computable.

Example 2.15. If we take as instance the spot in which a billiard ball will end up when hit, its generation complexity could be interpreted as the specification of its initial position, its initial velocity and its angular velocity.

Definition 2.16. (**Description Complexity**) Let s be an instance, then the generation complexity C_D^i of s is the length of the shortest available description which uniquely specifies s - according to the O-machine of agent i.

Example 2.17. If we take as instance the number 262144, a candidate unique description would be 2^{18}, since 262144 can be described as the number we obtain by computing two to the power of 18 within the decimal system.

Remark. From now on, we will drop the superscript i, and assume that any computation of an instance's generation and description complexity is relative to an agent's World and Observation machine respectively.

We can now formally define Unexpectedness as follows:

Definition 2.18. (**Unexpectedness**, [7]) Let x be an arbitrary instance, when the unexpectedness of x is defined as:

$$U(x) = C_W(x) - C_D(x)$$

Where $C_W(x)$ is the generation complexity, and $C_D(x)$ is the description complexity.

Note that, while there has been work which vindicates Simplicity Theory(ST) as a descriptively accurate cognitive model through experiments ([13, 42, 64]), for the purposes of this work we are only interested in its predictive accuracy. We treat ST as simply an alternative paradigm, non-strictly probabilistic, which may be more efficient at predicting which events are indeed unexpected to humans than orthodox bayesian epistemology.

2.6 Surprise according to Simplicity Theory

Through this section, we show that the difficulties presented in Section 2.3 and Section 2.4 are in fact easily solved, if we adopt a Simplicity Theoretic framweork instead of a probabilistic one.

Equiprobable Events

Consider a fair lottery, as introduced in the earlier sections. We have seen how, if we associate the surprisal value of a given outcome with its probability, then every outcome of a lottery would be equally surprising. Simplicity theory however gives us a straightforward way to individuate surprising events, which is in line with the experimental results of [42].

Since we are in the presence of a fair lottery, all of the possible outcomes are generated in the same way, meaning that, for all outcomes o, o', we will have: $C_W(o) = C_W(o')$.

Nevertheless, not all outcomes will have the same description complexity, as some sequences of numbers are easier to compress than others: sequences such as 11111 are easier to describe compared to seemingly random strings such as 83691, for which no pattern is easily available. Since we have that $C_D(83691) > C_D(11111)$, we also have that $U(11111) > U(83691)$, meaning that the string 11111 is more surprising than the string 83691. Again, this result is in line with numerous cognitive experiments, such as the ones carried out in [41] and [42]

Unexpected Consequences

We have mentioned before that a low probability is not a necessary requirement for an event to be unexpected, as high probability can be just as suprising. As a further example, consider the case of a lunar eclipse: astronomers can predict such events years, if not centuries in advance. While the probability of an eclipse given a correct astronomical computation approaches certainty, and we are aware of the fact that such an event can be predicted, we still find these advent of a lunar eclipse surprising.

While it may be the case that the probability of an eclipse given an astronomical model (or theory) approaches 1, the generation complexity (C_W) of an eclipse remains rather high: in order for a lunar eclipse to take place, the Sun, the Moon and the Earth must be perfectly aligned in the correct order at the correct time, meaning that the description of the parameters needed to generate this event will be long. At the same time, the description complexity (C_D) will be low: recall that we want the length of the shortest *unique* description for a lunar eclipse: as this is such a peculiar astronomical event, we can select a rather short description for it, without risking it being confused for some other, similar events. What this means in turn is that there will be a great difference between C_W and C_D, which according to ST, is the definition of an unexpected situation.

Unknown Unknowns

A third case we considered to hint at why probability theory cannot predict unexpectedness is the rather common case of "unknown unknowns": these events are evidently unexpected, but they are events towards which, by definition, we do not hold a propositional attitude.

While we saw that probability theoretic approaches to unexpectedness fail to account for such events, simplicity theory can easily make sense of them, providing an intuitive explanation of why such an event is unexpected.

As such an instance belonged to the category of unknown unknowns, we can expect the generation complexity of the event to be high according to a given agent's model. This is because the causal dependencies which would have brought about the given event have not been understood - the agent did not hold a propositional attitude towards the event at all - meaning that it would be unlikely to possess a brief description of the parameters that are required to generate the event.

At the same time, such an event has never happened before, at least according to the agent's model (or else it would not be an *unknown* unknown) meaning that its description complexity will be extremely low: any fitting description for the event will be by definition a unique description.

Awareness growth

In Section 2.4, we discussed a second, deeper sense in which the use of probabilities is not appropriate to measure unexpectedness: as the space of events in a probabilistic model is closed, considering propositions we were previously not aware of implies a reassessment of the probability mass assigned to any given event in our model. This means that if the unexpectedness of an event is tied to its probability, then the mere fact that an agent realises alternative scenarios are also possible influences the surprisal value of possible situations of which we were previously aware.

The framework proposed by Simplicity Theory does not hold such an assumption: the unexpectedness of a given event is an unbounded measure, meaning that the surprisal value of a possible outcome is independent of the surprisal value of all other alternatives. While the set of causal dependencies may be expressed in probabilistic terms, the computation of C_W is model agnostic, that is: the shortest descriptions of a set of parameters does not have to necessarily be given in terms of probabilistic causal dependencies - though this might be desirable in some scenarios.

2.7 Conclusion

Through this section, we have argued that Orthodox Bayesian Epistemology is not a good framework to predict the surprisal value of a given event. We briefly presented the main tenets of Bayesian epistemology, alongside a characterisation of surprise in terms of schema discrepancy; we then showed that low probability is not a sufficient or necessary condition for surprise: we did so by considering three different scenarios, namely low probability events, logical consequences, and unknown unknowns. We then argued that, due to probability's reliance on a fixed space of events, any approach which depends on probabilism cannot work, since it requires a credal mass redistribution every time we come to realise new possibilities. Finally, we presented Simplicity Theory, and showed how this model can easily predict surprise in those scenarios where probability theory fails.

Chapter 3

Technical Preliminaries

Before we introduce CompLogDL's formal framework, we will first review the basic concepts which will appear throughout the thesis. Section 3.1 will introduce Description Logics, which will form the basis of our CompLogDL system; Section 3.2 will introduce the basics of Complexity Theory which will be needed in Chapter 5, when we will analyse the computational complexity of CompLogDL's unexpectedness computation.

3.1 Description Logics

Description Logics (DLs) are a family of formal languages designed for structured knowledge representation, with a focus on balancing expressive power and computational tractability. These formalisms represent the knowledge of a given domain by first specifying the relevant concepts in the domain (the world), and then use these concepts to specify properties of objects and individuals in the domain (the world description). A DL knowledge base consists of a **TBox** (terminological component), and an **ABox** (assertional component): the former specifies the vocabulary of the application domain via concept definitions and inclusions, while the latter describes facts about named individuals, such as the concepts to which they belong.

Many descritpion logics have been introduced through the years (see [65] for a comprehensive introduction). Here, we present the language $\mathcal{EL}$, as in Chapter 3 we will use an annotated version of this DL to build our CompLogDL system.

Definition 3.1. (**$\mathcal{EL}$ Signature**) An $\mathcal{EL}$ signature is a triple of non-empty disjoint sets: $\Sigma = \langle C, I, R \rangle$ where:

1. $\Sigma_C = \{C_1, C_2, ...\}$ is a set of concepts, called 'atomic concepts'
2. $\Sigma_I = \{a, b, ...\}$ is a set of Instances
3. $\Sigma_R = \{R_1, R_2, ...\}$ is a set of Roles

Formally, a concept is defined recursively as:

Example 3.2. An **$\mathcal{EL}$ Signature** could be the following tuple $\Sigma' = \langle \Sigma_C, \Sigma_I, \Sigma_R \rangle$:

1. `C = [Patient, Fever, Cough, Vaccine, Human, Mammal].`
2. `I = [Alice, Bob, Carl, Fido].`
3. `R = [hasSymptom, hasDisease, receivesTreatment, hasTravelled].`

Definition 3.3. (**Concept**) Let $\Sigma = \langle \Sigma_C, \Sigma_I, \Sigma_R \rangle$ be an $\mathcal{EL}$ signature, and let A $\in \Sigma_C$ be an atomic concept. We can define $\mathcal{EL}$ concepts recursively as follows:

$$C \coloneqq \top \mid A \mid C \sqcap D \mid \exists R.D$$

We denote the set of $\mathcal{EL}$ concepts over Σ by $\mathcal{C}$: this is the the smallest set that contains Σ_C, and every concept C that can be constructed recursively from the atomic concepts in Σ_C using the above grammar.

Example 3.4. Given the signature of Example 3.2 we could construct the following concepts:

```
Severe_case_patient ≡ Patient ⊓ Fever ⊓ Cough
Treated_patient ≡ Patient ⊓ ∃receivesTreatment.Vaccine
```

In order to provide a formal semantics of $\mathcal{EL}$ concepts, we define $\mathcal{EL}$ Interpretations:

Definition 3.5. (**$\mathcal{EL}$ Interpretation**) Let $\Sigma = \langle \Sigma_C, \Sigma_I, \Sigma_R \rangle$ be an $\mathcal{EL}$ signature. An interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ consists of a non empty set $\Delta^{\mathcal{I}}$ (the domain of interpretation) and an interpretation function which assigns to every atomic concept $C_i \in \Sigma_C$ a set $A_i^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$, and to every role $R_j \in \Sigma_R$ a binary relation $R_j^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$. The interpretation function is interpreted recursively and extend to arbitrary concept descritpions as follows:

$$\top^I = \Delta^I$$
$$\bot^I = \emptyset$$
$$(C \sqcap D)^I = C^I \cap D^I$$
$$(\exists R.C)^I = \{a \in \Delta^I \mid \exists b((a,b) \in R^I \wedge b \in C^{\mathcal{I}})\}$$

Definition 3.6. (**Equivalent Concepts**) Let C, D be concepts. We say that C, D are equivalent if and only if $C^{\mathcal{I}} = D^{\mathcal{I}}$, that is: the interpretation function $\mathcal{I}$ assigns to both concepts the same set of elements.

One of the purposes of a description logic's knowledge base is to specify the concepts in the domain by means of other, atomic concepts: this is done via axioms which may take one of two forms:

1. **Concept Subsumption**: Statements of the form $C \sqsubseteq D$. We say that an interpretation $\mathcal{I}$ satisfies a concept inclusion if $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$.
2. **Concept Definition**[12]: Statements of the form $C' \equiv D'$. We say that an interpretation $\mathcal{I}$ satisfies a concept definition if $C'^{\mathcal{I}} = D'^{\mathcal{I}}$.

Definition 3.7. (**T-Box**) A T-Box $\mathcal{T}$ is a *finite* set of concept subsumptions and concept definitions.

[12]Note that $C' \equiv D'$ is an abbreviation for the two statements $C' \sqsubseteq D'$ and $D' \sqsubseteq C'$.

Definition 3.8. (**Normal Form**) We say that a T-Box $\mathcal{T}$ is in normal form if all of its concept inclusions have one of the following forms:

$$A \sqsubseteq B \mid A_1 \sqcap A_2 \sqsubseteq B \mid A \sqsubseteq \exists r.B \mid \exists r.A \sqsubseteq B$$

Definition 3.9. (**T-Box Model**) We say that an interpretation $\mathcal{I}$ satisfies a T-Box $\mathcal{T}$ iff $\mathcal{I}$ satisfies each element of $\mathcal{T}$. If $\mathcal{I}$ satisfies an axiom (resp. a set of axioms), then we say that it is a model of this axiom (resp. set of axioms). Two axioms or two sets of axioms are equivalent if they have the same models.

Ordinarily, DL Knowledge Bases pose restrictions on T-Box Axioms, by requiring that the set T of axioms be *acylic*:

Definition 3.10. (**Acyclic T-Box**) Let T be a set of T-Box axioms for a $\mathcal{EL}$ signature $\Sigma = \langle C, I, R\rangle$. Let A, B be atomic concepts occurring in C. We say that A *directly uses* B in T if B appears on the right-hand side of the definition of A, and we call "*uses*" the transitive closure of the relation *directly uses*. Then T contains a cycle iff there exists an atomic concept in T that uses itself. Otherwise, T is called acyclic.

Aside from concept subsumptions and definitions, a Knowledge Base $\mathcal{K}$ will also contain information a specific state of affairs in the given application domain.

Definition 3.11. (**A-Box**) Given an $\mathcal{EL}$ signature $\Sigma = \langle \Sigma_C, \Sigma_I, \Sigma_R\rangle$, an A-Box is a finite set $\mathcal{A}$ of concept and role assertions, that is statements which have one of the following two forms:
1. $C(a)$, where C is a concept and a is an instance.
2. $R(b, c)$, where R is a role and b, c are instances.

We extend the interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ of Definition 3.5 to account for individual names. From now on, the interpretation function not only maps atomic concepts and roles to sets and relations, but in addition maps each individual name a to an element $a^{\mathcal{I}} \in \Delta^{\mathcal{I}}$.

Remark. From now on, we will also make use of the unique name assumption (UNA): if if a, b are distinct individual names, then $a^{\mathcal{I}} \neq b^{\mathcal{I}}$.

Definition 3.12. (**A-Box Model**) The interpretation $\mathcal{I}$ satisfies the concept assertion $C(a)$ (i.e. $\mathcal{I} \models C(a)$) if $a^{\mathcal{I}} \in C^{\mathcal{I}}$. We say it satisfies the role assertion $R(a, b)$ (i.e. $\mathcal{I} \models R(a, b)$) if $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in R^{\mathcal{I}}$. An interpretation satisfies the A-Box A if it satisfies each assertion in A: we say that $\mathcal{I}$ is a *model* of the assertion or of the A-Box. Finally, $\mathcal{I}$ satisfies an assertion $\mathcal{A}$ or an A-Box A with respect to a T-Box T if in addition to being a model of $\mathcal{A}$ or of A, it is a model of T.

Definition 3.13. (**Knowledge Base**) Given an $\mathcal{EL}$ signature $\Sigma = \langle \Sigma_C, \Sigma_I, \Sigma_R \rangle$, a knowledge base is a pair $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ where $\mathcal{T}$ is a T-Box and $\mathcal{A}$ is an A-Box.

Definition 3.14. (**Consistent Knowledge Base**) Let $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ be a knowledge base consisting of an A-box $\mathcal{A}$ and a T-Box $\mathcal{T}$. We say that $\mathcal{K}$ is consistent if there is an interpretation $\mathcal{I}$ which is a *model* of $\mathcal{A}$ and $\mathcal{T}$

Another property of a knowledge base which we will require is that our knowledge base is *materialised* :

Definition 3.15. (**Materialised Knowledge Base**) Let $\mathcal{KB} = (\mathcal{T}, \mathcal{A})$ be a consistent knowledge base for a given $\mathcal{EL}$ signature $\Sigma = \langle \Sigma_C, \Sigma_I, \Sigma_R \rangle$. Let D be an arbitrary concept appearing in $\mathcal{KB}$, which is either atomic or derived. We say that $\mathcal{KB}$ is *concept materialised* when we have that:

$$(\forall \mathcal{I} \text{ models of } \mathcal{KB}, \mathcal{I} \models C(a)) \Rightarrow C(a) \in \mathcal{A}$$

Similarly, $\mathcal{KB}$ is *role materialised* if whenever

$$(\forall \mathcal{I} \text{ models of } \mathcal{KB}, \mathcal{I} \models R(a, b)) \Rightarrow R(a, b) \in \mathcal{A}$$

Finally, we say that $\mathcal{KB}$ is materialised if it is both role and concept materialised.

Intuitively, this means that all the information which is entailed by the knowledge base is already explicitly mentioned by the Knowledge Base.

in normal form (possibly using new concept names) such that:

Example 3.16. An example of a knowledge base could be the following:

```
Headache ⊑ ClinicalFinding
MildHeadache ≡ Headache ⊓ ∃hasSeverity.MildSeverity
CommonCold ≡ ViralRespiratoryInfection ⊓ ∃hasCausativeAgent.Rhinovirus
MildHeadache(Alice)
receivesTreatment(Alice, Antiviral)
```

Note that this knowledge base is *consistent*, but not materialised: for example, `headache(Alice)` is entailed by any model of this knowledge base, yet it is not in its A-Box.

The main inference problem concerning Description Logics is subsumption, which we cast as a decision problem, specifically for the $\mathcal{EL}$ description logic:

Definition 3.17. SUBSUMPTION ($\mathcal{EL}$)
Input : An $\mathcal{EL}$ signature $\Sigma = \langle \Sigma_C, \Sigma_I, \Sigma_R \rangle$, and a TBox $\mathcal{T}$ and two (atomic or derived) concepts C, D.
Question : Does $\mathcal{T} \models C \sqsubseteq D$ hold?

A well known result regarding this problem, given by Baader et al. [66], is the following:

Theorem 3.18. Subsumption($\mathcal{EL}$) can be decided in polynomial time.

To prove the theorem above, an algorithm (call it "SAT[·]") is given which not only computes subsumption between any two concept names (assuming that a T-Box is in normal form), but, given a T-Box, returns all the subsumption relationships between all pairs of concept names occurring in T.

Definition 3.19. (**T-Box Saturation**) Let $\mathcal{T}$ be a T-Box in normal form. We call a T-Box $\mathcal{T}'$ the *saturation* of $\mathcal{T}$ if, when $\text{SAT}[\mathcal{T}] = \mathcal{T}'$. We say that T is *saturated* when $\text{SAT}[\mathcal{T}] = \mathcal{T}$.

The results given in [66] can then be formalised in the following corollary:

Corollary 3.20. There exists a polynomial time algorithm which, given a normal form T-Box $\mathcal{T}$ as input, will compute its saturation $\mathcal{T}'$.

3.2 Computational Complexity Theory

Since in Chapter 5 we will present an computational complexity analysis of CompLogDL's inference problems, we provide the reader with a brief introduction to the essential concepts of complexity theory: for a more detailed introduction to the topics mentioned here, see the following [67–69].

Complexity theory is the area of theoretical computer science which concerns itself with the intrinsic difficulty of computational problems, and the resources that are required to solve them. In order to avoid getting caught in unnecessary details, computational complexity often deals with decision problems, which we can define as follows:

Definition 3.21. (**Decision Problems**) Let $x \in \Sigma^*$ be a string defined over some alphabet Σ, and let $Q \subseteq \Sigma^*$ be a property. The decision problem is then, given (x, Q) as input, to assess whether $x \in Q$; we say that an algorithm *solves* the problem Q if for all inputs x, the algorithm correctly decides whether $x \in Q$.

When we talk of the complexity of a decision problem, we usually speak of its space or time complexity: each of this notion is made precise by defining it relative to an abstract computation device, the simplest of which is a Deterministic Turing Machine, which can be defined following [67] as follows:.

Definition 3.22. (**Deterministic Turing Machine**) A Deterministic Turing Machine with one tape is a tuple

$$M = (S, \Sigma, \Delta, s_0, F),$$

where:

- S is the finite, non-empty set of states;
- $s_0 \in S$ is the initial state;

- $F \subseteq S$ is the set of accepting (final) states;
- Σ is the finite, non-empty input alphabet, containing \$ ("start symbol") and $\square$ ("blank symbol"), as well as 0 and 1; and
- $\Delta : S \times (\Sigma \cup \{\$, \square\}) \to S \times (\Sigma \cup \{\$\}) \times \{L, R, S\}$ is the transition function.

Deterministic Turing Machines are used to define the complexity class **P**:

Definition 3.23. (**Complexity Class P**) We say that a language L $\subseteq \{0,1\}^*$ is in **P** if it is computable in polynomial time by a Deterministic Turing Machine.

Another important complexity class is **NP**, which we define following [67]:

Definition 3.24. ((**Complexity Class NP**) We say that a language L $\subseteq \{0,1\}^*$ is in **NP** if there exists a polynomial $p : \mathbb{N} \to \mathbb{N}$ and a Deterministic Turing Machine M such that for every $x \in \{0,1\}^*$:

$$x \in L \iff \exists u \in \{0,1\}^{p(|x|)} \text{ such that } M(x,u) = 1$$

and the computation is done in a number of steps which is polynomial in the size of x

Remark. The string $u \in \{0,1\}^{p(|x|)}$ is called a *certificate* for x if $M(x,u) = 1$

A classical example of a decision problem which belongs to NP is SAT:

Definition 3.25. SAT

Input: A Propositional Logic formula φ, given in Conjunctive Normal Form, over a set of variables $X = \{x_1, ...x_n\}$
Question: Does there exist a variable assignment $\alpha : X \to [0,1]$ such that $\alpha(\varphi) = 1$?

Definition 3.26. We say that a language L $\subseteq \{0,1\}^*$ is polynomial time reducible to another language L' $\subseteq \{0,1\}^*$, denoted by L $\underset{p}{\leq}$ if there exists a polynomial time computable function $f : \{0,1\}^* \to \{0,1\}^*$ such that for every $x \in \{0,1\}^*$: $x \in L \iff \exists f(x) \in L'$. We then say that given any complexity class **C**, a language L is **C**-hard if if for any other language L' in **C**, $L' \underset{p}{\leq} L$. Finally, we say that a language L is **C**-complete if it is **C**-hard and L is in **C**

The following well known result is given here, without proof:

Theorem 3.27. (Cook-Levin, 1979). SAT is **NP**-complete.

Other decision problems which have been shown to be NP-Complete (see [70]) are HITTING SET and CLIQUE, which can be formalised as follows:

Definition 3.28. HITTING SET
Input: A finite universe $\mathcal{U}$, a collection of subsets $\mathcal{S} = \{S_1, ..., S_m\} \subseteq 2^{\mathcal{U}}$, and an integer

$k \in \mathbb{N}$.
Question: Does there exist a set $H \subseteq \mathcal{U}$ such that $|H| \leq k$ *and* $H \cap S_i \neq \emptyset \; \forall S_i$

Remark. This problem can be extended to a weighted version: we have $w : U \to \mathbb{Z}$ and we ask whether there exists a hitting set H s.t. $w[H] \leq k$

Definition 3.29. CLIQUE
Input: A graph $G = (V, E)$, $k \in \mathbb{N}$.
Question: Does there exist a clique K_n in $G = (V, E)$, where $n \geq k$?

Remark. This problem can also be exended to a weighted version: we have $w : V \to \mathbb{Z}$ and we ask whether there exists a clique[13] K_n s.t. $w[K_n] \geq k$

Another fundamental complexity class is **CoNP**:

Definition 3.30. We say that a language L $\subseteq \{0,1\}^*$ is in **CoNP** if its complement $\overline{\text{L}}$ is in **NP**, i.e. $\overline{\text{L}} = \{x \in \Sigma^* \mid x \notin L\} \in$ **NP**.

We can generalise the classes **NP** and **CoNP** by looking at all those languages definable via a combination of existential and universal quantifiers and a polynomial- time computable predicate.These are the languages which together form the *Polynomial Hierarchy*, which we can define as follows [67]:

Definition 3.31. For i ≥ 1, a language L is in Σ_i^p if there exists a polynomial-time TM M and a polynomial q such that:

$$x \in L \iff \exists u_1 \in \{0,1\}^{q(|x|)} \forall u_2 \in \{0,1\}^{q(|x|)} ... Q_i u_i \in \{0,1\}^{q(|x|)} M(x, u_1, ..., u_i) = 1$$

where Q_i denotes $\forall$ or $\exists$ depending on whether i is even or odd, respectively. Then the polynomial hierarchy **PH** = $\cup_i \Sigma_i^p$.

Remark. For every i, define $\Pi_i^p = \mathbf{Co}\Sigma_i^p = \left\{L : \overline{\text{L}} \in \Sigma_i^p\right\}$. Thus, $\Pi_1^p = \mathbf{CoNP}$.

An alternative characterisation of the polynomial hierarchy can be given in terms of Oracle Turing Machine [67]:

Definition 3.32. (**Oracle Turing Machine**) An oracle Turing machine is a Turing machine M that has a special read-write tape called the *oracle tape*, and three distinguished states: q_{query}, q_{yes}, and q_{no}.

To execute M, we specify, in addition to the input, a language $O \subseteq \{0,1\}^*$ that is used as the oracle for M. Whenever, during the execution, M enters the state q_{query}, the machine behaves as follows:

[13] By clique, we mean a a set of vertices $V' \subseteq V$ where every pair of distinct vertices is connected by an edge.

- If the string q currently written on the oracle tape belongs to O, then M transitions to the state q_{yes};
- If $q \notin O$, then M transitions to the state q_{no}.

Remark. For every $O \subseteq \{0,1\}^*$, $\boldsymbol{P^O}$ is the set containing every language that can be *decided* by a polynomial-time deterministic TMs with oracle access to $\mathbf{O}$ and $\boldsymbol{NP^O}$ is the set of every language that can be *verified* by a polynomial-time deterministic TM with oracle access to $\mathbf{O}$.

With this in mind, we can define the i-th level of the polynomial hierachy in terms of Oracles via the following proposition (see [67] for a discussion), which we give without proof.

Proposition 3.33. $\Sigma_i^p = \mathbf{NP}^{\Sigma_{i-1}^p}$.

While until now we have considered complexity classes which concern *decision problems*, complexity classes have been defined which analyse the complexity of returning an optimal solution to an optimisation problem. Instead of studying the complexity of a language L, we study the complexity of a function $f(x)$.

Definition 3.34. (**FP**) A function $f : \Sigma^* \to N$ is in $\mathbf{FP}^{SAT}$ if f is computable in polynomial time with an **NP** oracle. We say that f is in $\mathbf{FP}^{SAT}[z(n)]$ if $f \in \mathbf{FP}^{SAT}$ and f is computable using at most $z(n)$ queries on inputs of length n.

The class $\mathbf{FP}^{SAT}[z(n)]$ allows us to quantify the degree of complexity of a given **NP**-complete problem. A result which we will make use of in Chapter 4 is the following:

Theorem 3.35. ([71]) The weighted version of CLIQUE and VERTEX COVER are $\mathbf{FP}\ \left[n^{O(1)}\right]$ −complete.

Corollary 3.36. WEIGHTED HITTING SET is also $\mathbf{FP}\ \left[n^{O(1)}\right]$ −complete.

The corollary above follows immediately by the fact that VERTEX COVER is a special instance of HITTING SET in which all $S_i \in \mathcal{S}$ have cardinality 2.

Chapter 4

Introducing CompLogDL

In Chapter 2, we have motivated the choice of computing the surprisal value of a given event by computing its *unexpectedness* instead of its probability, following the framework of Simplicity Theory. In Chapter 3, we have briefly introduced the technical background which will be needed through the rest of the thesis. Here, we formally present CompLogDL: a graph based architecture whose function is to predict the extent to which an event is unexpected by computing its complexity drop, i.e. the extent to which an event is harder to generate than it is to describe. In Section 4.1, we present the problem of computing the description complexity of a given instance, while in Section 4.2 we show how to compute the generation complexity of an event; we then formally define a CompLogDL program in Section 4.3, and show how we can recover a notion of *conditional* complexity. Finally, in Section 4.4 we compare CompLogDL to CompLog, the complexity-based architecture given in [1].

General Idea

CompLogDL is inspired by probabilistic declarative languages such as ProbLog [15], but possesses its own characteristics. Instead of computing the probability of an event, CompLogDL aims to compute the unexpectedness of a given situation, where the unexpectedness is computed as described in Definition 2.18 , i.e.

$$U(s) = C_W(s) - C_D(s)$$

C_W is the bounded Kolmogorov complexity of the causal relations which bring about instance s (i.e. Generation Complexity), while C_D is the complexity of describing instance s (i.e. Description Complexity). A CompLogDL program is made of two different components, as the computation of C_D and C_W are independent of each other.

The computation of an instance's description complexity depends on an annotated knowledge base (the *descriptive* program), which is used to construct a description graph: the problem is cast as a minimum referring expression computation which expands on similar graph based-approaches in the literature [72, 73]. The second part is a set of annotated tuples (the *causal* program), which are used to form a graph representing the causal dependencies in a set of events, generalising the concept of Causal Diagram, [74, 75] : computing C_W then reduces to a multiple source weighted shortest path problem.

An Example

To provide intuition for this framework, consider the following scenario.

Example 4.1. Our agent is currently in a room with three other people, Alice, Eve and Bob, all of which are feeling sick with different symptoms. Suppose that we wanted to quantify how surprising it is that Alice, out of the three of them, is sick. If we were to interpret this probabilistically, we would compute the probability of Alice being sick; here, we compute the generation complexity first (what is the lowest cost chain of events that could have brought about this instance?) and then compute the complexity of describing this instance uniquely (what is the lowest cost description of this instance, which is such that it uniquely identifies *Alice* as *sick*?).

4.1 Description Complexity

The language of CompLogDL as it currently stands draws from *CompLog*'s original blueprint [1], though it differs from the original framework in a fundamental respect: the computation of an instance's description complexity is kept indepentent of its generation complexity, and relies on the computation of a minimum referring expression for a given instance.

4.1.1 Referring Expression Generation

The problem of Referring Expression Generation (REG) has been discussed widely within the field of computational linguistics, following its first formalisation by Dale [76]. The problem is cast as a computation of noun phrases that successfully identify the intended referent, while at the same time do not convey any false conversational implicature (as is prescribed by Gricean pragmatics [77]): only those properties which are needed to refer to an instance uniquely, without unnecessary information. A notable formulation for this problem, given by Dale et al. [78], is given as follows:

- Given a target (or referent) object $r \in D$, find a set of attribute–value pairs L whose conjunction is true of the target but not of any of the distractors ($r' \in D, r' \neq r$), the domain objects different from the target). L is called a *distinguishing description* of the target.

Approaches to this problem have grown extensively since (a comprehensive survey is found here [79]), from those which consider the overall coherence of a candidate expression [80], to those which deal with gradable properties, and thus vague descriptions [81]. The approaches that perhaps are most relevant to the approach presented here are the ones given by Areces et al. [72], and the one given by Krahmer et al. [73]. The former models a referrring expression as a formula of description logic, casting REG as the problem of finding a description logic formula φ which uniquely refers to a target set of individuals:

- Given a $\mathcal{DL}$ instance $\langle C, I, R \rangle$, a knowledge base $\mathcal{KB}$ and a target set $S \subseteq I$, look for a Description Logic formula φ such that the set of instances $i \in I$ for which the formula φ holds is equal to the target set S.

The latter casts REG as a problem on labelled, directed graphs: objects are represented as the nodes (vertices) in a graph (called the *scene graph*), and the properties of and relations between these objects are represented as edges connecting the nodes. Given a target object (i.e., a node in the scene graph), look for a distinguishing referring graph that is a subgraph of the scene graph and uniquely characterizes the target.

What we propose in the rest of this section is an extension of the two approaches above: the relations between instances are specified by within the description logic $\mathcal{EL}$, while at the same time being annotated with a value between 0 and 1. The annotated knowledge base is then converted in a labelled, directed and weighted graphs, where the costs of each edge is determined by a function which depends on the annotations given in the knowledge base. Our approach presents two main advantages: by allowing the annotation of the knowledge base, we allow for uncertain concept and role assertions, while by associating the knowledge base to a weighted labelled graph, we are able to increase the expressivity of our referring expression beyond concepts to include roles as well.

4.1.2 Annotated Descritpion Logic

CompLogDL's descriptive program is expressed in the description logic $\mathcal{EL}$, which we introduced in Section 3.1. Recall that, for an arbitrary $\mathcal{EL}$ signature $\langle C, I, R\rangle$, a **concept** D in $\mathcal{EL}$ is defined recursively as:

$$D = C_0 \mid \top \mid C_1 \sqcap C_2 \mid \exists R_i.C_1$$

Remark. Following the presentation of Description Logics in Section 3.1, we require that T-Box axioms be **acyclic** and in **normal form**.

We extend the logic $\mathcal{EL}$ by allowing T-Box and A-box axioms of a given knowledge base $\mathcal{KB}$ to be annotated with values in the interval $(0, 1]$. However, unlike the approach of related work [82], we do not require that our annotated values conform to probability axioms, but instead interpret them as simple statements indicating the degree to which relations between between two instances, or two classes, are accessible to the agent. By relaxing the assumption that these annotations conform to probabilities, we avoid the problems we discussed in Section 2.4.2. If we drop the assumption that our weight function *must* conform to a probability, we are free to change the relations between our instances as we see fit (with no obligation to change existing epistemic relations), whenever new information is acquired.

Examples

In order to build some intuition regarding CompLogDL's syntax, we have decided to begin our discussion with some examples, motivating the choice of annotating the axioms in our Knowledge Base

Remark. In the examples that follow, we use # as syntactic sugar to differentiate between instances and concepts.

.

Example 4.2.

```
1 :: Yellow_Fever ⊑ Disease
0.7 :: Fever ⊑ Symptom
```

Here, we are specifying that whenever an instance (for example a patient) can be described as having yellow fever, then our agent is also immediately able to talk of said patient as one with a disease (the cognitive cost of associating these two descriptions as low as it can get). On the other hand, the second statement tells us that our agent does not immediately associate an individual experiencing a fever as one experiencing a symptom of an illness.

Example 4.3.

```
0.8 :: Cold_symptoms ≡ Fever ⊓ Runny_nose
```

Here, we are specifying the cognitive cost of defining the symptoms of a cold as a fever and a runny nose, according to our agent.

We use concept assertions to associate a particular instance to a class. For example, we may have:

Example 4.4.

```
1 :: Patient(#Alice)
0.7 :: Patient(#Bob)
```

Here we are asserting that Alice can immediately be identified as a patient, while for Bob, this requires some cognitive effort.

Aside from annotating, concept assertions, we may also annotate role assertions. These may be used to assert additional relations between specific instances, which we may be useful when we want to describe them. For example, we could assert the following:

Example 4.5.

```
0.6 :: Same_Room(#Alice, #Bob)
1  :: Travelled(#Alice, #Asia)
0.3 :: Travelled(#Bob, #Asia)
```

Here, we are stating that, for our agent, the degree to which the assertion "Alice has been in the same room as Bob" is accessible when thinking about alice is 0.6. Similarly, the assertion "Alice travelled to Asia recently" is immediately accessible, while the same assertion is not as accessible if we are talking about Bob.

4.1.3 Descriptive Program

Having given some intuition regarding the descriptive component of CompLogDL, we can formally CompLogDL's descriptive program as follows. Let the triple $\Sigma = \langle \Sigma_C, \Sigma_I, \Sigma_R \rangle$ be any $\mathcal{EL}$ signature of as defined in Section 3.1. We then have:

Definition 4.6. A **Descriptive Program** $\mathbb{P}_D$ is a tuple $\langle(\mathcal{T}, \mathcal{A}), w\rangle$ where:

1. $(\mathcal{T}, \mathcal{A})$ is a Knowledge Base consisting of T-Box and A-Box axioms constructed using the instances, roles and atomic concepts specified in $\Sigma = \langle\Sigma_C, \Sigma_I, \Sigma_R\rangle$
2. $w : \mathcal{T} \cup \mathcal{A} \longrightarrow (0, 1]$ is a cost function, which assigns to each T-Box and A-Box axiom a degree between 0 and 1.

A descriptive program $\mathbb{P}_D$ will specify a set of relations which hold between concepts, instances and roles, together with the degree to which they are accessible to the agent. An important requirement is that the relations between different concepts be consistent:

Definition 4.7. (**Consistent Program**) A descriptive program $\mathbb{P}_D = \langle(\mathcal{T}, \mathcal{A}), w\rangle$ is consistent if $(\mathcal{T}, \mathcal{A})$ is a *consistent* knowledge base (as defined in Section 3.1).

In addition to this, we require that our program be complete, meaning that all the relationships between concepts appearing in the T-Box $\mathcal{T}$ are made explicity in $\mathbb{P}_D$:

Definition 4.8. (**Complete Program**) We say that a descriptive program $\mathbb{P}_D = \langle(\mathcal{T}, \mathcal{A}), w\rangle$ is *complete* when its knowledge base $(\mathcal{T}, \mathcal{A})$ is saturated (as defined in Section 3.1)

Finally, we will require that our program's knowledge base be materialised, as defined below:

Definition 4.9. (**Materialised Knowledge Base**) Let $\mathcal{KB} = (\mathcal{T}, \mathcal{A})$ be a consistent knowledge base for a given $\mathcal{EL}$ signature $\Sigma = \langle\Sigma_C, \Sigma_I, \Sigma_R\rangle$. Let D be an arbitrary concept appearing in $\mathcal{KB}$, which is either atomic or derived. We say that $\mathcal{KB}$ is *concept materialised* when we have that:

$$(\forall \mathcal{I} \text{ models of } \mathcal{KB}, \mathcal{I} \models C(a)) \Rightarrow C(a) \in \mathcal{A}$$

Similarly, $\mathcal{KB}$ is *role materialised* if whenever

$$(\forall \mathcal{I} \text{ models of } \mathcal{KB}, \mathcal{I} \models R(a, b)) \Rightarrow R(a, b) \in \mathcal{A}$$

Finally, we say that $\mathcal{KB}$ is materialised if it is both role and concept materialised.

Intuitively, this means that all the information which is entailed by the knowledge base is already explicitly mentioned by the Knowledge Base.

Definition 4.10. (**Materialised Program**) A descriptive program $\mathbb{P}_D = \langle(\mathcal{T}, \mathcal{A}), w\rangle$ is consistent if $(\mathcal{T}, \mathcal{A})$ is a *materialised* knowledge base (as defined in Section 3.1).

Remark. From this point forwards, we will assume that any descriptive program is consistent, materialised and complete.

Example 4.11. We give here a simple example of a CompLogDL Descriptive Program, describing the instance we presented in Example 4.1.

```
1   :: Fever  ⊑ Symptom
1   :: Cough  ⊑ Symptom
0.9 :: SevereCase ≡ Patient ⊓ Fever ⊓ Cough

0.9 :: Patient(#Alice)
0.7 :: Patient(#Bob)
0.9 :: Patient(#Eve)

1     :: Symptom(#Alice)
0.85   :: Fever(#Alice)
0.6   :: Cough(#Alice)
0.1   :: SevereCasePatient(#Alice)

1     :: Symptom(#Bob)
0.5   :: Fever(#Bob)

1     :: Symptom(#Eve)
0.4   :: Cough(#Eve)

0.9 :: Travelled(#Alice, #Asia)
0.7 :: Travelled(#Bob, #Europe)

1 :: SisterOf(#Alice, #Eve)
1 :: SisterOf(#Eve, #Alice)
```

4.1.4 Description Graph

Suppose we are given a descriptive program $\mathbb{P}_D = \langle(\mathcal{T}, \mathcal{A}), w\rangle$, which is constructed via a an $\mathcal{EL}$ signature $\Sigma = \langle\Sigma_C, \Sigma_I, \Sigma_R\rangle$. The program can be regarded as an agent's mental model (representing the instances' descriptions via concepts and roles).

Recall that a descriptive program is equipped with a cost function $w : \mathcal{T} \cup \mathcal{A} \longrightarrow (0, 1]$, acting as an indicator of epistemic accessibility: it tells us how strongly concepts or instances are related in the mind of the agent.

In order to compute the description complexities of a given instance, we will need an intermediary function, which we call a descriptive complexity function. Its purpose is to assign complexity values to concept and role assertions: it will depend indirectly on the cost function w, but it will return a positive real number representing the complexity of a given assertion. We first definine the desirable properties that such a function ought to have.

Complexity Function

Let $\mathbb{P}_D = \langle(\mathcal{T}, \mathcal{A}), w\rangle$ be a descriptive program. We have:

Definition 4.12. (**Descriptive Complexity Function**) A function $\mathfrak{F} : \mathcal{A} \longrightarrow \mathbb{R}^+$ is a Complexity function if and only if it satisfies the following[14]:

1. **Non-Negativity**: For all concept assertions $C \in \mathcal{C}$:

$$\mathfrak{F}(C) \geq 0$$

and equivalently for all role assertions $R \in \mathcal{R}$:

$$\mathfrak{F}(R) \geq 0$$

2. **Super-additivity**: Let $C \equiv D_1 \sqcap D_2$. Then:

$$\mathfrak{F}(C) \geq \max(\mathfrak{F}(D_1), \mathfrak{F}(D_2))$$

Intuitively, this means that the function will always return a non-negative number, which depends on the degree of accessibility which the cost function in the given descriptive program $\mathbb{P}_D$ assigns to the TBox and ABox axioms. While any function satisfying the properties above will work, we provide an example of such a function, which we will show below satisfies the conditions specified above. Once more, let $\mathbb{P} = \langle(\mathcal{T}, \mathcal{A}), w\rangle$, be a CompLogDL Program associated with its $\mathcal{EL}$ signature.

Example 4.13. The function $\mathfrak{C} : \mathcal{A} \longrightarrow \mathbb{R}^+$ is defined as follows. We define it case by case, depending on the input it is given.

1. If the input is a role assertion $R(a, b) \in \mathcal{R}$, then

$$\mathfrak{C}(R(a, b)) := \log \frac{1}{w(R(a, b))} + 1$$

2. If the input is an concept assertion $C(a)$, and C is an *atomic* concept:
   - If $C(a)$ does not appear on the right hand side of any concept inclusion statements in $\mathbb{P}$, i.e. $C_1 \not\sqsubseteq C$ for any other concept C_1:

$$\mathfrak{C}(C(a)) = \log \frac{1}{w(C(a))} + 1$$

   - Otherwise, we have that for some C_1, $C_1 \sqsubseteq C \in \mathbb{P}$, and thus:

$$\mathfrak{C}(C(a)) = \max\left(\log \frac{1}{w(C(a))}, \mathfrak{C}(C_1)\right) + 1$$

3. The input is a role assertion $C(a)$, and C is a *derived* concept. Let $w(\equiv)$ be the uncertainty value that the cost function applies to the definition of C in terms of other concepts. We then have:
   - If $C(a)$ does not appear on the right hand side of any concept inclusion statements in $\mathbb{P}$:

$$\mathfrak{C}(C(a)) = \begin{cases} \log \frac{1}{w(\equiv)} + \log \frac{1}{w(C(a))} + \mathfrak{C}(D_1) + ... + \mathfrak{C}(D_n) + 1 \text{ if } C = D_1 \sqcap ... \sqcap D_n \\ \log \frac{1}{w(\equiv)} + \log \frac{1}{w(C(a))} + \mathfrak{C}(D) + 1 \text{ if } x = \exists R.D \end{cases}$$

[14] Abusing notation, we use C as shorthand for the concept assertion C(a), where a is an instance and C is a concept. Similarly, R is shorthand for R(a,b).

- If C appears on the right hand side of a concept inclusion, i.e. for some other C_1, $C_1 \sqsubseteq C \in \mathbb{P}$, the function is defined as

$$\mathfrak{C}(C(a)) = \max(\mathfrak{C}(C(a)), \mathfrak{C}(C_1) + 1)$$

Proposition 4.14. The function $\mathfrak{C} : \mathcal{A} \longrightarrow \mathbb{R}^+$ in Example 4.13 is a Complexity function.

Proof.

1. **Non-negativity**: Immediate by the definition of $\mathfrak{C}$, we only assign a function a sum of positive terms.
2. **Super Additivity**: Let $C \equiv D_1 \sqcap D_2$. Then we know by the definition of $\mathfrak{C}$ that:

$$\mathfrak{C}(C(a)) \geq \log \frac{1}{w(\equiv)} + \log \frac{1}{w(C(a))} + \mathfrak{C}(D_1) + \mathfrak{C}(D_2) + 1 \geq \max(\mathfrak{C}(D_1), \mathfrak{C}(D_2))$$

   where the second inequality follows by the fact that $\mathfrak{C}$ is non-negative.

□

Example 4.15. Given the program in Example 4.11, we compute the complexities of some of the role and concept assertions appearing, according to the function $\mathfrak{C}$ defined above:

- $\mathfrak{C}(\texttt{Fever(\#Alice)}) = \log\left(\frac{1}{0.85}\right) + 1 = 1.235$
- $\mathfrak{C}(\texttt{Symptom(\#Alice)}) = \max(\log\left(\frac{1}{1}\right), \mathfrak{C}(\texttt{Fever(\#Alice)})) + 1 = 2.235$

While this is only a particular instance of a suitable descriptive complexity function, our reasons for defining it as we have are as follows:

1. The cost of asserting *any* description is strictly more than zero, even if a concept is immediately accessible; there is a cognitive cost which is expended by just uttering a given description, which we have represented here by adding a $+1$ to the cost of any role and concept assertion.
2. The cost of describing an instance using derived concepts is always greater than the cost of their components; if a concept C is defined in terms of other, simpler concepts, then using C as a description for some instance should not be cheaper cognitively than the components which make up concept C.

Remark. In our current implementation, the cost of concept subsumptions (call it $w(\sqsubseteq)$) is left unutilized, however one could define alternative cost functions which add $w(\sqsubseteq)$ to the cost of the concept being subsumed (as we are currently doing for derived concepts). We see this being useful in situations where we might want a more specific concept to be cognitively more costly to use than a more general one.

Graph Construction

Given a descriptive program $\mathbb{P}_D = \langle(\mathcal{T}, \mathcal{A}), w\rangle$, the computation of a given instance's description complexity relies on the conversion of our program to what we call its description

graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, c, n)$. Intuitively, the instances which appear in our program will represent the set of nodes, while roles and concept membership will be represented as directed edges and loops.

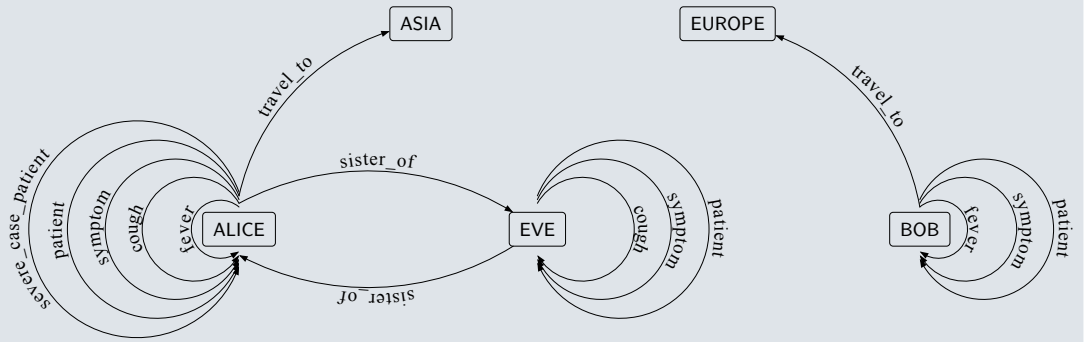
Formally, let $\mathbb{P}_D = \langle(\mathcal{T}, \mathcal{A}), w\rangle$ be a descriptive program, let $\Sigma = \langle\Sigma_C, \Sigma_I, \Sigma_R\rangle$ be the $\mathcal{EL}$ signature on which it is constructed, and let $\mathfrak{C}_D$ be a descriptive complexity function. We define its description graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, c, n)$ as follows:

Definition 4.16. (**Description Graph**) A description graph is a tuple $\mathcal{G} = (\mathcal{V}, \mathcal{E}, c, \boldsymbol{n})$, where:

- $\mathcal{V} = \mathcal{I}$, where $\mathcal{I}$ is the set of instances in $\Sigma = \langle\Sigma_C, \Sigma_I, \Sigma_R\rangle$.
- $\mathcal{E} = \{(a, b) \mid R(a, b) \in \mathcal{A}\} \cup \{(a, a) \mid C(a) \in \mathcal{A}\}$: the roles are represented as directed edges, while the concepts to which an instance belongs are represented by self-loops.
- $c : \mathcal{E} \to \mathbb{R}$ is a cost function
- The function $\boldsymbol{n}$ assigns to each edge its name, i.e. the name of the concept or role that generated a the given edge.

The description graph can be computed rather straightforwardly, once we are given the description program, an $\mathcal{EL}$ signature and a descriptive complexity function $\mathfrak{C}_D$. For every instance i in our $\mathcal{EL}$ signature, we add a node; for every concept assertion $C(i)$, we add a looping edge (i, i) to the graph, to which we assign the name "C" and whose cost is given by $c(i, i) \coloneqq \mathfrak{C}_D(C(i))$. Similarly, for any role assertion $R(i, j)$, we add an edge (i, j), to which we assign the name "R" and cost label is given by $c((i, j)) \coloneqq (\mathfrak{F}(R(i, j))$.

Example 4.17. Recall the program we gave in Example 4.11: its description graph can be given as follows[15]:



[15] We omitted annotations to ensure legibility: in practice, $c : \mathcal{E} \to \mathbb{R}$ assigns costs to each edge.

4.1.5 Computing C_D

The description complexity of an instance is computed as the cost of the minimum unique referring expression for that given instance, building on similar work [72, 73]. Note that, while CompLogDL is based on description logics, the referring expression we will generate will be a formula $\varphi \in \mathbf{FO}_{\wedge}$ (predicate logic restricted to conjunction)[16].

Let $\mathbb{P}$ be a descriptive progam, and $\mathcal{G}$ its description graph. We can define the first order interpretation $\mathcal{M} = (\mathcal{D}, I)$ by taking as domain $\mathcal{D}$ the set of vertices $\{v_1, ..., v_n\}$ in $\mathcal{G}$, and then translating each loop and edge of $\mathbb{P}$ into unary and binary predicate statements.

We can now specify what conditions a first order logic formula must meet to be considered a referring expression.

Definition 4.18. (**CompLogDL Referring Expression**) Let i be an instance in $\mathbb{P}_D$, and let $\varphi[x/i]$ denote the formula φ where all instances of variable x are substituted with i. Let $c_1, ..., c_{m+m'}$ be a set of constants. A First order logic formula ψ is a CompLogDL referring expression for instance i if and only if:

1. $\psi = C_1(x) \wedge ... \wedge C_n(x) \wedge R_0(x, c_1) \wedge ...R_m(x, c_m) \wedge R'_0(c_{m+1}, x) \wedge ... \wedge R'_{m'}(c_{m+m'}, x)$, and
2. $\mathbb{M} \vDash \psi[x/i]$

Intuitively, φ is a referring expression if it is a conjunction of of unary and binary predicates which, when thought of as loops and directed edges in he description graph $\mathcal{G}$, form a subgraph F such that all edges in F contain i as one (or both) of the endpoints.

In line with the intuition above, we can talk of a referring expression generated by a subgraph F of the description graph $\mathcal{G}$:

Definition 4.19. A referring expression φ for instance i is *generated* by a subgraph F of $\mathcal{G}$ if and only if F is the subgraph obtained by considering all unary predicates assertions C_i in φ as loops in G, and all roles R_i in φ as directed edges.

What we want is to find a unique referring expression, which can individuate a given instance:

Definition 4.20. (**Unique Referring Expression**) A first-order formula φ is a *unique* referring expression for a given instance i if :

1. φ is a referring expression as specified in Definition 4.18, i.e. $\varphi = C_1(x) \wedge ... \wedge C_n(x) \wedge R_0(x, c_1) \wedge ...R_m(x, c_m) \wedge R'_0(c_{m+1}, x) \wedge ... \wedge R'_{m'}(c_{m+m'}, x)$ *and* $\mathbb{M} \vDash \varphi[x/i]$
2. There is no set of constants $c_1, ..., c_{m+m'}$, and no instance $i' \neq i$ such that:

$$\mathbb{M} \vDash \psi[x/i][c_1, ..., c_{m+m'}/d_1, ..., d_{m+m'}]$$

[16] Our referring expression is a conjunction of binary and unary predicates which represent role and concept assertions. Note that concepts of the form $\exists R.C$ will also take the form of unary predicates, *not* existentially quantified first order formulas.

We can think of this in graph theoretic terms as well: a referring expression φ for instance i is unique if the subgraph F of $\mathcal{G}$ that *generates* φ does not have an isomorphic copy in the description graph $\mathcal{G}$ which is not "anchored" at instance i. Given the way in which the description graph is constructed (see Definition 4.16), it is immediate to see that the two definitions are equivalent.

What this means is that, for any set of instances T

We can now define the notion of the complexity of a referring expression, which will depend on its components, and on the instance to which it refers . Let $\mathbb{P}_D$ be a descriptive program, $\Sigma = \langle \Sigma_C, \Sigma_I, \Sigma_R \rangle$ its $\mathcal{EL}$ signature and $\mathfrak{C}$ a complexity function.

Definition 4.21. (**Formula Complexity**) Let $\psi = C_1(x) \wedge ... \wedge C_n(x) \wedge R_i(x, c_1) \wedge ... \wedge R_m(x, c_k) \wedge R'_j(y_{m+1}, x) \wedge ...R'_{m'}(y_{m+m'}, x)$ be a referring expression for some instance i. Then the complexity of ψ with respect to i (denoted by $\hat{C}(\varphi)_i$), is equal to:

$$\hat{C}(\varphi)_i = \sum_{\mathbb{M} \models C([x/i])} \mathfrak{C}(C(i)) + \sum_{\mathbb{M} \models R([x/i], c_j)} \mathfrak{C}\big(R(i, c_j)\big) + \sum_{\mathbb{M} \models R(c_j, [x/i])} \mathfrak{C}\big(R(c_j, i)\big)$$

That is, we sum over the complexities of the concept and role assertions which appear in formula φ.

What we want to construct is a formula φ which is a unique referring expression for a given instance i: φ should be the cheapest possible formula,which we call a *minimal* referring expression:

Definition 4.22. (**Minimal RE**) We say that a referring expression φ for an instance i is **minimal** iff, for all other referring expressions ψ of i, we have:

$$\hat{C}(\varphi)_i \leq \hat{C}(\psi)_i$$

We are finally in a position to clearly state our description complexity problem: we identify the description complexity of a given instance i with the complexity of its *minimum unique referring expression*. We can formalise the problem as follows:

Definition 4.23.
Input: A signature $\Sigma = \langle \Sigma_C, \Sigma_I, \Sigma_R \rangle$, a description graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, c, n)$ and an instance $i \in \Sigma_I$
Output: A FOL formula φ which is a minimal unique referring expression for $\{i\}$, if such a φ exists, and to return $\hat{C}(\varphi)_i$

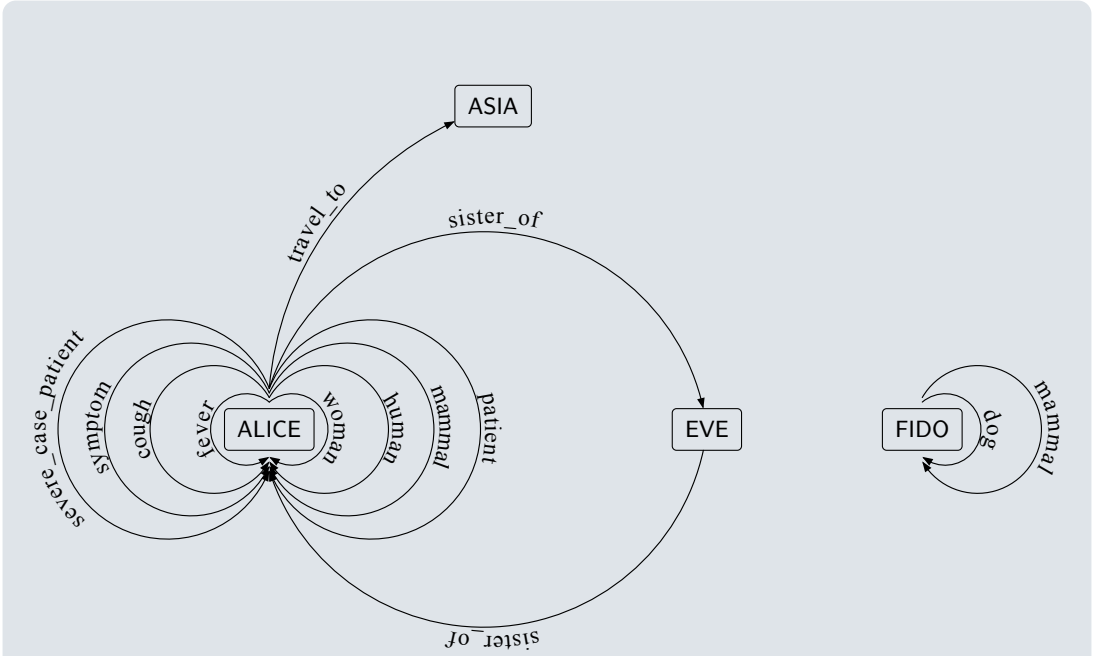
Example 4.24. Suppose that, as in Example 4.1, we want to compute the description complexity of the instance "Alice is Sick". Out of all of the edges leaving the node `#Alice`, we need to pick a combination of edges leaving the node #Alice which uniquely identifies

that node. In this case, we have that the minimal unique referring expression for instance `#Alice` is $\varphi = \text{Fever}(\#\text{Alice}) \wedge \text{Cough}(\#\text{Alice})$ and that $\hat{C}(\varphi) = 1.23 + 1.73 = 2.96$.

Example 4.25. To show how the descriptive complexity changes depending on the universe of discourse, consider the following example, where the target instance is once more $\#\text{Alice}$.

```
// ---------- T-Box ----------
1   :: Fever  ⊑ Symptom
1   :: Cough  ⊑ Symptom
0.9 :: SevereCase ≡ Patient ⊓ Fever ⊓ Cough
0.9 :: Woman  ⊑ Human

0.8 :: Dog  ⊑ Mammal
1   :: Human ⊑ Mammal

// ---------- A-Box ----------
1   :: Patient(#Alice)
1     :: Symptom(#Alice)
0.85   :: Fever(#Alice)
0.6   :: Cough(#Alice)
0.1   :: SevereCasePatient(#Alice)

1   :: Woman(#Alice)
0.9   :: Human(#Alice)
1     :: Mammal(#Alice)

0.9 :: TravelledTo(#Alice, #Asia)

1 :: SisterOf(#Alice, #Eve)
1 :: SisterOf(#Eve, #Alice)

1     :: Dog(#Fido)
0.9   :: Mammal(#Fido)
```

All of the descriptions of Alice that were true before are true now, but our context has changed. The graph is constructed as follows:



Clearly, aside from the edges labelled `Mammal`, no other edge is shared between instance `#Alice` and instance `#Fido`. For this reason, we are free to choose a simpler description of `#Alice`: the cheapest edge leaving the node is `Woman(#Alice)`, which has a complexity of 1, meaning that the referring expression for Alice is $\varphi =$`Woman(#Alice)`, and that $\hat{C}(\varphi) = 1$. Note that an equivalent referring expression would have been $\psi =$`Patient(#Alice)` , as $\hat{C}(\psi) = 1$.

4.2 Generation Complexity

While the description complexity of an instance depends an annotated $\mathcal{EL}$ knowledge base, the generation complexity of an instance is computed via a causal program, which specifies of a set of causal relations and a degree of causal dependency between any instances. From the causal program, we then recover a causation graph, and cast the generation complexity problem as a multiple source shortest problem to a target instance.

4.2.1 Causal graphical Models

Since Pearl's seminal work on causality [83], interest in causal models has grown into a vast area of research, which spans from philosophy to artificial intelligence. The scope of a causal model is to encode the truth value of sentences dealing with causal relations between events (if we do A, B will happen), counterfactuals (If A had been different, B would have been different), as well as general causal utterances ("A may cause B"). Pearl's definition of a causal model can be given as follows[17]:

[17] We follow the definitions as presented in [83].

Definition 4.26. (**Causal Model**) A causal model is a triple $M = \langle U, V, F \rangle$ where

1. U is a set of background variables, that are determined by factors outside the model
2. V is a set $\{V_1, ...V_n\}$ of *endogenous* variables that are determined by variables in the model, i.e. variables in $U \cup V$
3. $F = \{f_1, ..., f_n\}$ is a set of functions such that each $f_i : U_i \cup PA_i \to V_i$ (where $U_i \subseteq U$ and $PA_i \subseteq V/V_i$). The entire set F must form a mapping from U to V.

To each causal model we can associate a directed graph $G(M)$, in which each node corresponds to a variable and the directed edges point from members of PA_i and U_i towards V_i, and this graph is called a *causal diagram*. Note that the causal diagram identifies the background variables and endogenous variables which have an influence on V_i, without specifying each f_i. This definition of causal model above is easily extended to a probabilistic causal model:

Definition 4.27. (**Probabilistic Causal Model**) A causal model is a pair $M = \langle M, P(u) \rangle$ where

1. M is a Causal Model as in Definition 4.26.
2. $P(u)$ is a probability function defined over the domain of U.

Remark. The function $P(u)$, together with the fact that each endogenous variable is a function of U, defines a probability distributions over the endogenous variables.

The concept of Probabilistic Causal Models has been and is to this day an extremely influential framework, and it is beyond the scope of this section to provide a full review: we refer the reader to the following introductions ([83–85]).

In the following section, we propose a modification of the above model, which is neither qualitative nor probabilistic, and which aims at describing actual causation. By actual causation (also found in the literature under the names of "token" or "singular" causation), we mean that we are interested in defining the causal relations which take place between singular instances. As an example, consider the difference between the statement "Smoking causes lung cancer" and the statement "Alice smoking 10 sigarettes a day for ten years is the cause of Alice getting diagnosed with lung cancer today".

The metaphysics of actual causation and its relation to general causation has been vastly discussed in the philosophical literature [86], and it is beyond the scope of this work to discuss its interpretation in detail. To build further intuition regarding our approach, we can begin by thinking about actual causation in a probabilistic framework. Let $(P, \Omega, P(\Omega))$ be a probability space, and let $X : \Omega \to \mathbb{R}^+, Y : \Omega \to \mathbb{R}^+$ be two random variables. Instead of assessing the relation between X and Y, to determine whether a change in X leads to a change in Y as is done in a Causal Model, we want to assess the relation between instance $X = x$ and instance $Y = y$. Models of probabilistic actual causation follow this intuition [87], and Pearl's framework has been adapted to this setting as well [75].

Our approach to generate the generation complexity of a given instance is quantitative, yet not probabilistic: we construct a causation diagram, starting from a set of cause-effect instance pairs. Our model presents the following three features:

- Each cause-effect instance pair is annotated with a degree between zero and one - which in epistemic terms, represents the degree to which the agent thinks that the two instances are causally related.
- The annotations of each cause effect pair for any given cause e_i do not form a probability measure, as they might sum to less/more than 1.
- Similarly, the annotation of each cause effect pair for any given effect e_i do not form a probability measure, as they might sum to less/more than 1.

4.2.2 Causal Program

The specification of annotated causal relations between events constitutes CompLogDL's Causal program, which we can define as follows:

Definition 4.28. (**Causal Program**) A Causal program $\mathbb{P}_W$ is defined as a tuple (E, R, w), where:

1. $E = \{e_1, ..., e_n\}$ is a set of events (or equivalently, instances)
2. $R \subset E \times E$ is a set of ordered pairs (e_i, e_j), which are interpreted as event e_i is a cause of e_j.
3. $w : R \to (0, 1]$ is a weight function, which represents the extent to which the events are causally related, according to the agent.

Remark. Note that, though $w : R \to (0, 1]$ maps tuples to a number between 0 and 1, this does not define a set of probability functions. Thus it might be the case that

$$\sum_{e_j \in E, (e_i, e_j) \in R} (e_i, e_j) \neq 1 \text{ , and also } \sum_{e_i \in E, (e_i, e_j) \in R} (e_i, e_j) \neq 1$$

Example 4.29. As an example, we can consider the following causal program, which describe relations between events generating the instance in which Alice is sick, following Example 4.1:

1. Let the set of events be $E = \{$Sick_Alice, Cold, Low_Immunity, Stress, Travel, Airborne_Exposure, Contamination, Pneumonia$\}$
2. While the cause effect pairs and their weights are given in the following table:

w(Cold, Sick_Alice) = 0.48	w(Low_Immunity, Cold) = 0.08	w(Stress, Low_Immunity) = 0.65
w(Travel, Stress) = 0.81	w(Stress, Sick_Alice) = 0.20	w(Pneumonia, Sick_Alice) = 0.38

w(Contamination, Pneumonia) = 0.88	w(Airborne_exposure, Pneumonia) = 0.06	w(Travel, Airborne_exposure) = 0.23

4.2.3 Causation Graph

Much like in the case of our description complexity, we use our program $\mathbb{P}_W$ to specify a graph (which we call "causation graph"[18]): we then use the graph to compute the generation complexity. Our target instance $e_i \in E$ is represented as a vertex in a causation graph: given such a vertex v_{e_i}, we want to find a sequence of events which can bring about e_i and has lowest complexity amongst all sequences.

Once the causal program is specified, we can immediately recover its causation graph. A causation graph will be an edge-weighted graph, where the cost of the edges is given by a generation complexity function. We begin by defining the conditions that a function must satisfy in order to be considered a generation complexity function.

Definition 4.30. (Generation Complexity Function) Let $\mathbb{P}_C = (E, R, w)$ be a causal program. $\mathfrak{C} : R \to \mathbb{R}^+$ is a generation complexity function if and only if it satisfies the following:

1. **Monotonicity**: For all $(e_i, e_j), (e_s, e_t) \in R$: if $w(e_i, e_j) \leq w(e_s, e_t)$ then $\mathfrak{C}(e_i, e_j) \geq \mathfrak{C}(e_s, e_t)$
2. **Non-Negativity** For all (e_i, e_j), $\mathfrak{C}(e_i, e_j) \geq 0$

As an example, consider the following function:

Example 4.31. Let $\mathfrak{C} : R \to \mathbb{R}^+$ be such that

$$\mathfrak{C}(e_i, e_j) = \log \frac{1}{w(e_i, e_j)} + 1$$

. In the examples that follow, we will make use of the function specified here[19], but in principle, we are free to specify a complexity function as we please.

In the examples that follow, we will make use of the function specified in the example above, but in principle, we are free to specify a complexity function as we please.

Definition 4.32. (Causation Graph) The causation graph $\mathcal{G}_c = (V_c, E_c, \mathfrak{C})$ is a tuple where:

$$V_c := \{e_i |\ e_i \text{ is an event}\}$$

$$E_c := \{(e_i, e_j)\ |(e_i, e_j) \in \mathbb{P}_C\}$$

[18] We use the term "causation graph" to distinguish our approach from causal diagrams as presented in Definition 4.26.

[19] Note that this is equivalent to the fomula complexity computation we defined in Definition 4.21, if we interpret the tuple (v_j, v_{j+1}) as an unlabelled role assertion.

$$\text{and } \mathfrak{C} : E_c \to \mathbb{R}^+$$

is a generation complexity function assigning a cost to each edge.

Remark. Note that, contrary to other causation graphical models in the literature [74], we do not require that our graph be acyclic: since our computation relies on a shortest weighted path computation, cycles will never be part of a causal path.

Intuitively, given any node $e_i \in E_c$ in the causal subgraph, its generation complexity is given by the cost of the weighted shortest path from all possible initial conditions: we interpret this as the minimum description of the parameters which are necessary to bring about e_i. By initial conditions, we mean those events which, according to our program, are not themselves caused by something else: we could for example interpret them as the possible "current situations" the agent considers. Formally, we define them as follows.

Definition 4.33. (Initial Condition) Given a causation graph $\mathcal{G}_c = (V_c, E_c, \mathfrak{C})$, a node $v \in V_c$ is an initial condition if and only if it has degree at least 1 *and*:
1. There is no v' such that $(v', v) \in E_c$;
2. There is at least one v'' such that $(v, v'') \in E_c$

Then let $\mathcal{G}_c = (V_c, E_c, \mathfrak{C})$ be a causation graph, which has and $S = \{s_1, ..., s_k\}$ as the set of initial conditions.

Definition 4.34. (Path Complexity) Let $\pi = ((v_0, v_1), (v_1, v_2), ..., (v_n, v_i))$ be a path in $\mathcal{G}_c = (V_c, E_c, \mathfrak{C})$ reaching node v_i. Then its complexity, denoted $\hat{C}(\pi(i))$, is defined as:

$$\hat{C}(\pi(i)) = \sum_{(v_j, v_{j+1}) \in \pi} \mathfrak{C}\big((v_j, v_{j+1})\big)$$

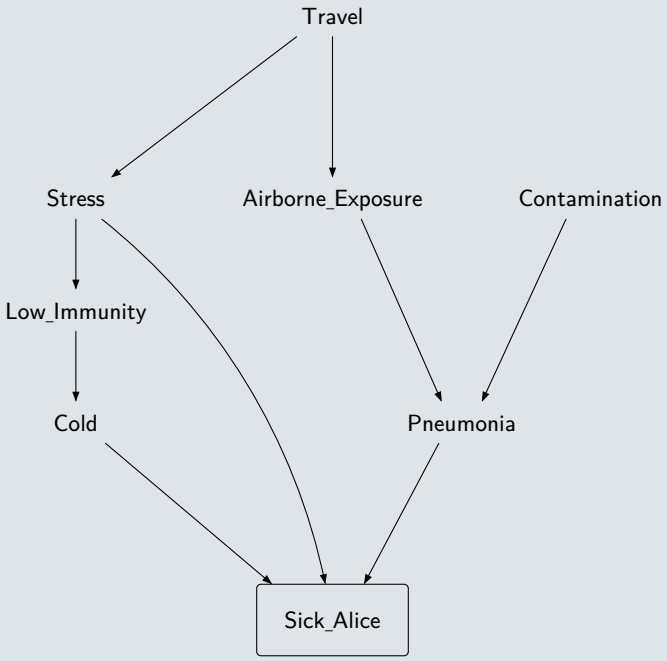
Definition 4.35. (Minimal Path Complexity) Let v_i be an arbitraty node in $\mathcal{G}$. Then we say that a path formula $\pi(i)$ is minimal if and only if for all other paths $\pi'(i)$ reaching the same target v_i from a different initial condition, we have: $\psi_{\pi'}(i)$

$$\hat{C}((\pi(i))) \leq \hat{C}((\pi'(i)))$$

We are now in a position to formalise the generation complexity problem as follows:

Definition 4.36.
Input: A causation graph $\mathcal{G}_c = (V_c, E_c, \mathfrak{C})$ and an event $e_i \in V_c$
Output: The cheapest path π^* from any initial condition $s_j \in S$ to e_i, together with its complexity $\hat{C}(\varphi_\pi(i))$.

Example 4.37. Consider the causal program we gave in Example 4.29. Its causation graph can be given as[20]:



We take Sick_Alice to be the target event, and compute the shortest path to it from the initial conditions, namely Travel and Contamination. In this case, the shortest weighted path is $\pi = (\text{Contamination}, \text{Pneumonia}), (\text{Pneumonia}, \text{Sick\_Alice})$, which has complexity:

$$\hat{C}(\pi(i)) = \sum_{(v_j, v_{j+1}) \in \pi} \mathfrak{C}\big((v_j, v_{j+1})\big) = 2 + \log\left(\frac{1}{0.88}\right) + \log\left(\frac{1}{0.38}\right) = 3.58$$

Remark. Note that, as we do not require that the cost of outgoing edges from any given vertex formed a probability measure (i.e. summed up to 1), we are free to add and remove new causal relations as we please. This means that any change in awareness regarding the possible causes of an event does not necessarily influence the generation complexity (and thus the unexpectedness) of a given event.

4.3 Computing Unexpectedness

In Section 4.1 and Section 4.2, we have showed how the problem of computing the description and generation complexity of an instance can be formalised, given a descriptive program and a causal program, respectively.

[20]Once more, we omit the cost annotations given by the function $\mathfrak{C} : E_c \to \mathbb{R}$ for legibility.

We are finally in the position to define a CompLogDL program:

Definition 4.38. (CompLogDL Program) A CompLogDL Program is a tuple $(\mathbb{P}_D, \mathbb{P}_C, \mathfrak{C}_D, \mathfrak{C}_C)$, where:

- $\mathbb{P}_C$ is a descriptive program
- $\mathbb{P}_D$ is a causal program
- $\mathfrak{C}_D$ is a descriptive complexity function
- $\mathfrak{C}_C$ is a generation complexity function

Now recall that, within simplicity theory, the unexpectedness of an instance s is defined as:

$$U(s) = C_w(s) - C_D(s)$$

Within the framwework of CompLogDL, this means that unexpectedness of an instance $s \in \mathbb{P}$ is the difference between the complexity of a minimal path $\pi^*(s)$ obtained via a program's causation graph $\mathcal{G}$, and the complexity of the minimum unique referring expression for instance s. Using the notation defined above, we can thus re-cast the problem as:

$$U(s) = \hat{C}(\varphi^*_\pi(s)) - \hat{C}(\varphi)_s$$

This means that an event is *unexpected* if its cost of its cheapest path is strictly greater than the cost of its minimum unique referring expression, i.e. $\hat{C}(\varphi^*_\pi(s)) > \hat{C}(\varphi)_s$.

Example 4.39. Let us return back to our initial instance, where we wanted to compute the unexpectedness of the instance "Alice is Sick". We know from Example 4.11 that this instance has description complexity 2.96, and we know by Example 4.29 that it has generation complexity 3.58. This means that

$$U(\text{Alice is Sick}) = 3.58 - 2.97 = 0.61$$

Suppose now that instead of our agent being in a room wih Alice, Eve and Bob, we are now in the situation of Example 4.25, where our agent is standing in a room with Alice and a dog. The unexpectedness of alice being sick is now:

$$U(\text{Alice is Sick}) = 3.58 - 1 = 2.58$$

Even though the causal chain of events leading to Alice being sick did not change, we can still say that the fact that Alice is sick is less surprising.

4.3.1 Conditional Unexpectedness

Just like in the framework of probability theory we may want to compute the *conditional* probability of an event x *given* an event y, we can define a similar conditional unexpectedness computation. If within the probabilistic framework we are only allowed to condition on events, within CompLogDL we are allowed to condition on three different components: concepts, roles and events.

Conditioning on concepts and roles

When computing the descritption complexity of a given instance s , we are searching for a description of s in terms of the "cheapest" concepts and roles: epistemically, this cost represents how accessible a given concept or role is to an agent.

We can thus think of conditioning on concepts and roles as asserting that they are immediately accessible to the agent; as such, it only influences the computation of an instance's description, since the accessibility of a role or a concept does not influence the causal dependencies between isntances. Formally, we can define it as follows. Let $\mathbb{P}_D$ be a descriptive program and $\mathfrak{C} : \mathcal{A} \to \mathbb{R}^+$ its complexity function: when we condition on a given concept, we update our function $\mathfrak{C}c$ to its contitioned form, $\mathfrak{C}^U$: this function is equivalent to $\mathfrak{C}$, except for the complexity it assigns to the concept C we are conditioning on, and those concepts which are defined in terms of C.

Definition 4.40. Conditioning on a Concept Let C be an arbitrary concept in $\mathbb{P}_D$. Then if we condition on C, we have that, for all instances a in $\mathbb{P}_D$:

$$\text{if } C(a) \in \mathbb{P}_D, \text{then } \mathfrak{C}^U(C(a)) = 1$$

If C appears on the right hand side of any concept definition D (i.e $D = C \sqcap C'$), we still require super-additivity, by requiring that:

$$\mathfrak{C}^U(D(a)) = \max(\mathfrak{C}^U(C(a)), \mathfrak{C}^U(C'(a)))$$

Remark. Note that, once we condition on a concept C and drop the cost of all concept assertions including C to one[21], the function $\mathfrak{C}^U$ may not respect super-additivity with respect to the concepts that are used to define C.

Example 4.41. Assume that, given the example program in Example 4.11, we condition on the role `Severe_Patient()`. Since Alice is the only instance for which this role applies, it clearly is a unique referring expression. This means that the minimum referring expression for Alice is now φ' = `Severe_Patient(#Alice)`, and $\hat{C}(\varphi') = 2$.

Similarly, we can define role conditioning as follows:

Definition 4.42. Conditioning on a Role Let R be an arbitrary role in $\mathbb{P}_D$, then if we condition on R, we have that, for all instances a, b in $\mathbb{P}_D$:

$$\text{if } R(a,b) \in \mathbb{P}_D, \text{then } \mathfrak{C}^U(R(a,b)) = 1$$

Conditioning on events

We saw that conditioning on instances and roles influences the computation of a situations' description complexity: we also have a second kind of conditioning which only

[21] The cost of this assertion drops to 1, not zero, to match the intuition that even though a concept may be immediately available, there is still a non-zero cost in uttering it.

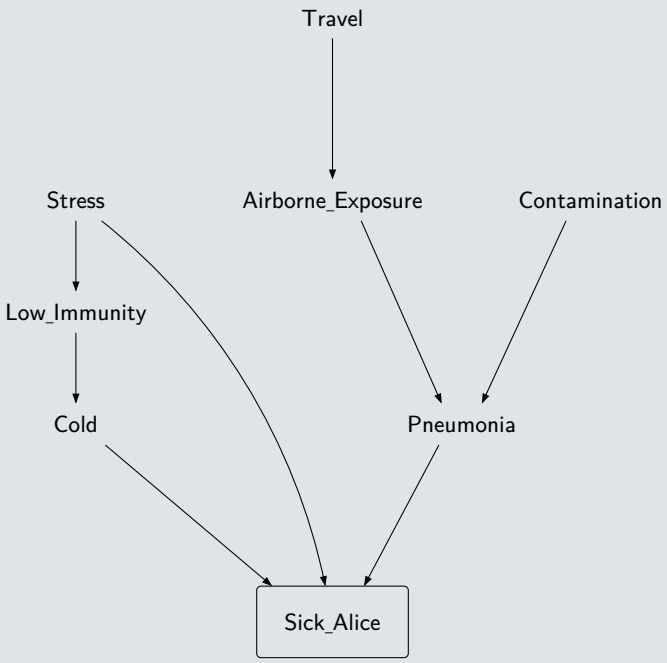
influences its generation complexity. This is the situation in which we conditionalise on an event: epistemically, this means that we are assuming that the event in question is now an initial condition, and disregard any causal dependencies which are needed to bring it about. Formally, we can think of this as a shrinking of the causation graph, and define it as follows. Let $\mathbb{P}_C$ be a causal program, $\mathfrak{C}$ its complexity function, and let $\mathcal{G}_c = (V_c, E_c, \mathfrak{C})$ be its causation graph:

Definition 4.43. (**Conditioning on an event**) Let s be an arbitrary event in $\mathbb{P}_C$. Then if we condition on s, we update the set E_c to a new set E_c^U where:

$$E_c^U = E_c \setminus \{(t, s) \mid t \in V_c, (t, s) \in E_c\}$$

This means that, when we condition on an event s, we remove all incoming edges from other events into s, meaning that s is now an initial condition.

Example 4.44. Suppose we know that Alice is stressed, and thus we condition on this event, we then have the following causation graph:



Stress is now an initial condition, and the shortest weighted path from any initial condition to `Sick_Alice` is now $\pi' = (\text{Stress, Sick\_Alice}) = 1 + \log\left(\frac{1}{0.20}\right) = 1 + 2.32 = 3.32$.

4.4 CompLog and CompLogDL

We have mentioned before that CompLogDL draws from an existing blueprint given in [1], which first introduces the concept of complexity based programming. Here, we briefly

compare the two proposed architectures, to show in what ways CompLogDL maintains the intuition of the original CompLog in [1].

The computation performed by both systems is the same, that is: both CompLog and CompLogDL compute the difference between the generation and description complexity of an instance, and return a real number representing the unexpectedness of said instance. Nevertheless, the two systems differ vastly in the way in which they compute both complexity values.

We have seen how CompLogDL presents two different components, each of which presents its own syntax. The description complexity component revolves around an annotated knowledge based which is specified using the description logic $\mathcal{EL}$, while the generation complexity is computed based on an input consisting of annotated cause-effect pairs, which form a causation graph. CompLog, on the other hand, utilizes the same syntax for both computations, via a propositional language which is inspired by ProbLog's syntax and which specifies both causal and descriptive events.

In CompLogDL, we allow for a distinction between instances (or events), roles and concepts, and any $\mathcal{EL}$ signature $i \in I$ appearing the descriptive program $\mathbb{P}_D$ need not have an equivalent counter part in the causal program $\mathbb{P}_C$. Similarly, the concepts and roles are only present in the descriptive program $\mathbb{P}_D$, as they rerpesent possible descriptions of an instance. CompLog instead distinguishes *conditions* from *events*: the former are used to construct *declarative rules* of the form $\mathtt{x} \rightarrow \mathtt{y}$, while the latter are used to form causal rules[22] $\#\mathtt{x} \rightarrow \#\mathtt{y}$.

In CompLog, rules are annotated with a cost (a positive real number), which represents the descriptive or causal complexity of a given descriptive or causal rule, respectively. Each set of rules should then form a graph (see Figure 2 below), and both complexity computations are defined as shortest path computations on weighted graphs.

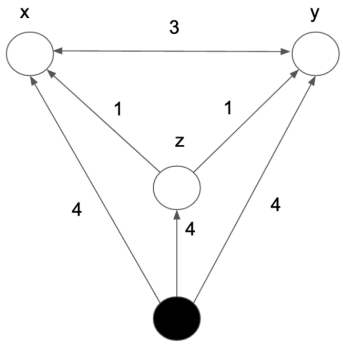


Figure 2: Graph constructed from a descriptive model, where the colored node represents the available resources, and the number on the edges specify the complexity of coloring in the target node, if the source node is colored (see [1] for details)

[22]The symbol # is used as syntactic sugar in CompLog to distinguish between the case in which item x is considered as a *condition*, and when it is considered an *event*

Whereas CompLog was meant to illustrate the working principles of the inferential system based on complexity in a simplistic propositional setting, CompLogDL is the first attempt to connect it to contemporary automated reasoning standards as description logics and the formal study of computational complexity.

Chapter 5

Computational Complexity

Now that we have presented the framework and formalised our inference problems, we provide a detailed analysis of their computational complexity. In section Section 5.1, we show that the complexity of computing C_D does not depend on whether role assertions appear in our CompLogDL program: generating the description complexity has an equivalent **NP** -complete decision problem, independently of whether we limit ourselves to concepts and roles: we show that the optimisation version of this problem is $\mathbf{FP}^{SAT}[O(n)]$-complete. In Section 5.2, we will show that the computation of C_W can be easily carried out in polynomial time, as it is equivalent to an instance of a single source shortest path problem.

5.1 Analysis for the description complexity

5.1.1 Allowing only Concepts

We have seen in that computing the description complexity of an instance i in a descriptive program $\mathbb{P}_D$ can be described as a minimum unique referring expression computation, where we aim to find the cheapest FOL formula φ representing a set of edges and vertices on the description graph.

Through this section, we restrict ourselves to the case in which our descriptive program does not have role assertions, i.e. an instance $\mathbb{P}_D = \langle(\mathcal{T}, \mathcal{A}), w\rangle$ where the $\mathcal{A}$ does not contain any role assertions. This means that in our description graph $\mathcal{G}$ will consist of a set of nodes representing the instances in $\mathcal{I}$, and a set of loops: there will be no edges connecting any set of vertices.

Recall the original formulation of the problem, given in Definition 4.23:

Definition 5.1.
Input: A signature $\Sigma = \langle\Sigma_C, \Sigma_I, \Sigma_R\rangle$, a description graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, c, n)$ and an instance $i \in \Sigma_I$
Output: A FOL formula φ which is a minimal unique referring expression for $\{i\}$, if such a φ exists, and to return $\hat{C}(\varphi)_i$

As the formula φ was obtained by a conjunction of edges involving our instance, this is equivalent to outputting the subgraph subgraph F of $\mathcal{G}$ that generates it, that is:

1. Our target instance $i \in F$
2. The formula φ obtained by taking the conjunction of all edges in F is a unique minimal referring expression for instance i (see Definition 4.20).

Condition 2 implies that our output will be the cheapest labelled subgraph F of $\mathcal{G}$ which is not isomorphic to any other subgraph F'. This is an optimisation problem (for brevity, call it SIMPLE REFERRING EXPRESSION),and we first show that the decision problem equivalent to it is NP complete. The decision problem (which we call SIMPLE K-REFERRING EXPRESSION) we analyse it as follows:

Definition 5.2.
Input: A description graph $\mathcal{G} = (V, E, l, n)$ an instance $i \in V$, and a $k \in \mathbb{N}$
Question: Is there a connected subgraph F^* of G such that:
1. All loops e in F^*, have i as endpoint
2. For all other connected subgraphs F' of $\mathcal{G}$, $F \not\equiv F'$, and
3. The cost of the formula φ generated by F is less than k.

Lemma 5.3. The decision problem described above is in the complexity class NP.

Proof. A certificate for this problem will be a connected subgraph F of $G = (V, E, l, n)$, for which we have to check the following:
1. The sum of the costs of the edges in F less than k: this step can be done in time linear in the size of E.
2. F is a subgraph of all looping edges leaving the target node v_*: this can be done in time linear in the size of F.
3. For any vertex $j \in V$ which is not equal to i, we check whether there is a set of loops with j as an endpoint which is exactly the same as F; this is equivalent to checking whether the set of looping edges L of F is a subset of the looping edges leaving the other vertices in G. This step can be carried out in time $|V| \times |L|$.

If all three conditions are satisfied, then we return yes. □

We now show that this decision problem is also NP-hard, via reduction from HITTING SET (which we have formally presented in Definition 3.28):

Remark. The SIMPLE K-REFERRING EXPRESSION problem defined above is equivalent to the description logic referring expression problem in [72]. If our FOL Referring Expression φ is restricted to concepts, then it is just a reformulation of a DL concept formed as the intersection of all concepts appearing in φ. Therefore, we are providing a formal complexity analysis for the problem presented there, for the case in which the DL used is $\mathcal{EL}$.

Lemma 5.4. HITTING SET is polynomial time reducible to SIMPLE K-REFERRING EXPRESSION.

Proof. Let $U = \{u_1, ..., u_n\}$, **S** = $\{S_1, ...S_m\} \subseteq P(U), k \in \mathbb{N}$ be an instance of HITTING SET as defined in Section 3.2. We first give a reduction to an instance of SIMPLE K-REFERRING EXPRESSION as follows:

The graph $G = (V, E, l, n)$ is constructed as follows:

- The vertices V are the set **S**, together with an additional node S_*
- For every $S_i \in \mathbf{S}$, we add as many looping edges as there are elements in U/S_i: each loop is labeled with an element $u_i \in U/S_i$. The node S_* will get n loops, each labeled with an element $u_i \in U$.
- For all edges e, l(e) = 1
- Our target node is S_*, and k is the same as in our HITTING SET instance

This reduction can be carried out in time linear in the size of our HITTING SET instance's input, as we construct V in |**S**| steps, and then take the set difference between U and every S_i once to add edges. The labelling can be done as the edges are constructed, and the cost is a constant time operation.

We now show that this reduction is indeed correct.

Assume that there is a solution to HITTING SET: this means that there is a set $H \subseteq U$ such that for all $S_i \in \mathbf{S}$, $H \cap S_i \neq \emptyset$.

What this means is that for all S_i, there exists a u_i that is not in U/S_i, meaning that the vertex S_i does not have a loop leaving S_i which is labeled u_i. Now let F be the subgraph of G which is obtained by considering the set of all loops around S_* which have as label some $u_i \in H$. It is clear that by construction,there are no isomorphic copies of F in G, as for every other vertex s_i we are guaranteed that there is at least one labelled loop in F which does not appear amongst the loops leaving s_i. Since $|H| \leq k$, it is clear that $l(F) = \sum_{e \in F} l(e) \leq k$, which concludes this direction of the proof.

Now assume that there is a solution to our instance of SIMPLE K-REFERRING EXPRESSION, meaning that there is a subgraph F such that $l(F) \leq k$ and that there are no isomorphic copies of F in G which are rooted at a vertex other than S_*. Now consider the set $H = \{u_i \mid l(e) = u_i, e \in F\}$. Since D has no other isomorphic copies in G, for every node $s_i \in G$ we have (at least) an edge labelled with some $u_j \in H$ which does not appear as a loop around node S_i, but does appear around S_*. Since the labels around every S_i were exactly those $u_j \in U/S_i$, we are guaranteed that for every $S_i, S_i \cap H \neq \emptyset$. Finally, since $l(F) = \sum_{e \in F} l(e) \leq k$, we know that $|H| \leq k$: H is therefore a solution to our arbitrary HITTING SET instance, concluding our proof. □

Theorem 5.5. Let $\mathcal{G} = (V, E, l, n)$ be a description graph for some description program $\mathbb{P}_D$. If the set E contains only loops, then the decision problem equivalent of generating a minimum unique referring expression is NP-Complete.

Proof. Immediately follows from Lemma 5.3 and Lemma 5.4, together with the observation that generating a referring expression related to a given graph F and its cost is a linear time operation. □

We can now consider the optimisation version of this problem, which we call OPT-SIMPLE K-REFERRING EXPRESSION which we present here:

Definition 5.6.
Input: A description graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, l, n)$ an instance $i \in V$, and a $k \in \mathbb{N}$
Question: Return the least connected subgraph F^* of G such that:
1. All loops e in F^*, have i as endpoint
2. For all other connected subgraphs F' of $\mathcal{G}$, $F \not\equiv F'$

We will now show that the above is a P^{NP}-complete problem. We begin by showing membership:

Proposition 5.7. OPT-SIMPLE K-REFERRING EXPRESSION belongs to the class $\mathbf{FP}^{SAT}[O(n)]$.

Proof.

We construct a polynomial time algorithm that, using a linear amount of NP oracle calls, can solve this optimisation problem. Let $\mathcal{G} = (V, E, l, n)$, $v_i \in V$ be our input:

INPUT: $(\mathcal{G} = (V, E, l, n), v_i \in V)$):

let $L \leftarrow 1$
let $U \leftarrow \sum l(e)$ ▷ U is the total cost of graph $\mathcal{G}$
while L < U:
    $M \leftarrow \lfloor \frac{L+U}{2} \rfloor$
    Query Oracle: Is there a solution **for** v_i on G, when $k = M$?
    if Yes:
        $U \leftarrow M$
    else:
        $L \leftarrow M + 1$
let $k \leftarrow L$ ▷ We found the cost of a solution
if Oracle never answered Yes:
    Return "*No solution*"
else: ▷ Construct candidate subgraph
    let $G = (V, E)$
    $S \leftarrow \{\}$
    $j = 1$
    while $j \neq |E|$:
        if e_j is a loop on node v_i:
            $G = \left(V, E - \{e_j\}\right)$
            Query Oracle: Is there a solution **for** v_i on $\mathcal{G}$ **and** $k = L$?
            if not:
                let $G = \left(V, E + e_j\right)$ ▷ add edge e_j back
                let $j \leftarrow j + 1$
            else:
                let $j \leftarrow j + 1$
                $S += e_j$
        else:
            j = j + 1
Return $(\{v_i\}, S)$

The above algorithm will first make at most $\log\left(|\sum_{e \in E} l(e)|\right)$ calls to the Oracle, in order to find the least k (if any) for which the decision problem has a solution.

Once that is done, the algorithm then iterate through the list of edges: it uses oracle calls to verify whether any given loop on our target instance v_i appears in an optimal solution to the decision problem. If an edge does appear, then we add it to the set of edges S which will form part of our goal subgraph. Once we have iterated through all edges, we return the subgraph $(\{v_i\}, S)$.

As the Oracle Calls are assumed to take constant time, our algorithm is polynomial: finding the ideal k takes time $O\big(\log\big(\sum_{e\in E} l(e)\big)\big)$, while returning an optimal solution can be done in $O(|E|)$ steps, i.e. using $O(n)$ oracle calls.

□

In order to show $\mathbf{FP}\big[n^{O(1)}\big]$-hardness, we will reduce it to an instance of another optimisation problem, namely weighted HITTING SET (formally introduced in Section 3.2), which we know to be $\mathbf{FP}\big[n^{O(1)}\big]$–complete .

Proposition 5.8. weighted HITTING SET is polynomial time reducible to OPT-SIMPLE REFERRING EXPRESSION

Proof.

Let $\langle \mathcal{U}, \mathcal{S} \subseteq P(\mathcal{U}), w : \mathcal{U} \to \mathbb{R}^+ \rangle$ be an instance of weighted HITTING SET.We use the same reduction as in Lemma 5.4, with the addition that each loop on a vertex (v_i, v_i) that is labelled with a given u_j, also gets assigned the cost of that $u_j \in \mathcal{U}$ as it is given in the weighted HITTING SET istance:

- The vertices V are the set **S**, together with an additional node S_*
- For every $S_i \in \mathbf{S}$, we add as many looping edges as there are elements in U/S_i: each loop is labeled with an element $u_i \in U/S_i$. The node S_* will get n loops, each labeled with an element $u_i \in U$.
- For all edges e, l(e) = $w(u_i)$ where u_i is the label of edge e.
- Our target node is S_*, and k is the same as in our HITTING SET instance

Now suppose that our instance of weighted HITTING SET has a solution, that is: we found a HITTING SET $H^* \subseteq \mathcal{S}$ such that, for all other candidate HITTING SETS $H', w(H^*) \leq w(H')$. We already know that the graph F^* generated by H^* following the strategy outlined in Lemma 5.4 is indeed a graph whose edges contain the target instance as an endpoint, and that it is unique. It remains to show that it is the graph with the lowest cost which satisfies these conditions. Assume for a contradiction that this were not the case, then there would be a graph F' (containing i and with no isomorphic copies in the parent graph) such that $l(F') < l(F^*)$ but then let $H_{F'}$ be the set containing those u_i that label the edges of the graph F'. By the arguments in Lemma 5.4, we know that H' would be a hitting set, and by construction we would have that $l(F') = w(H_{F'}) < w(H^*) = l(F^*)$, contradicting the minimality of H^*.

Now suppose that we have a solution to OPT-SIMPLE REFERRING EXPRESSION, that is we have a graph F containing i and with no isomorphic copies in the parent graph such that $l(F') < l(F)$ for all other subgraphs F' which contain i and are also unique up to isomorphism. Let H_F be the set generated by F following the strategy in Lemma 5.4, then we know that H_F is indeed a hitting set. Suppose that it were not minimal: then there would exist a hitting set H^* such that $w(H^*) < w(H_F)$. But then by letting F^* be the graph containing as edges those loops on S^* labeled with u_i, we would have unique subgraph (again by the

arguments in Lemma 5.4) and also: $w(H^*) = l(F^*) < l(F) = w(H_F)$. As this contradicts the minimality of F, we know that H_F must in fact me a minimum hitting set. □

Finally, we can now conclude the following:

Theorem 5.9. OPT-SIMPLE REFERRING EXPRESSION is $\mathbf{FP}\left[n^{O(1)}\right]$-complete

Proof.

Proposition 5.7 gives us memership in $\mathbf{FP}\left[n^{O(1)}\right]$, while by Proposition 5.8, we know that any $\mathbf{FP}\left[n^{O(1)}\right]$-complete problem can be reduced to an instance of OPT-SIMPLE REFERRING EXPRESSION, giving us hardness. □

5.1.2 Allowing Roles

Through this section, we consider the unrestricted version of our problem, i.e. we no longer assume that the set of role assertions is empty, and instead it allow it to be a set of arbitrary size.

Let $\mathcal{G} = (V, E, l, n)$ be a description graph containing role and concept assertions, and $i \in V_c$ an instance for which we want to compute a minimum referring expression. Before proceeding with the complexity analysis of this problem, we note that any subgraph F of G generating a referring expression will have the same form:

Proposition 5.10. If F is a subgraph of $\mathcal{G}$ which generates a referring expression for i, then F is a star graph, which may only have self-loops around node i

Proof.

This is easily seen by noting that a referring expression, in line with Definition 4.18, is generated only by graphs whose edges have the target instance i as one (or both) of their endpoints. □

Given this observation, we will now show that introducing roles does not increase the complexity of our problem, by reducing it to an instance of SIMPLE K-REFERRING EXPRESSION. We begin by formally defining our decision problem, which we call K-REFERRING EXPRESSION:

Definition 5.11.
Input: A description graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, c, \boldsymbol{n})$ an instance $i \in V$, and a $k \in \mathbb{N}$
Question: Is there a connected subgraph F^* of G such that:
1. For all edges e in F^*, i is one (or both) of the endpoints of e,
2. For all other subgraphs F' of $\mathcal{G}$, $F \not\equiv F'$, and
3. The cost of the formula φ generated by F is less than k.

We now proceed by showing that this instance is NP-Hard:

Lemma 5.12. SIMPLE K-REFERRING EXPRESSION is polynomial time reducible to K-REFERRING EXPRESSION

Proof.

This is seen immediately by noting that SIMPLE K-REFERRING EXPRESSION is a special case of K-REFERRING EXPRESSION, one in which the set E contains no edges of the form (u, v) with $u \neq v$. □

Given the above, we can easily observe that K-REFERRING EXPRESSION is NP-Complete.

Theorem 5.13. K-REFERRING EXPRESSION is NP-Complete.

Proof.

Lemma 5.12 gives us NP-Hardness, while membership in NP can be given by a counting argument analogous to the one given in Lemma 5.3, as shown here: A certificate for this problem will once more be a connected subgraph F of $G = (V, E)$, for which we have to check the following:

1. The sum of the costs of the edges in F less than k: this step can be done in time linear in the size of E.
2. Every edge in F contains the target node v_* as one of its endpoints: this can be done in time linear in the size of F.
3. For any vertex $j \in V$ which is not equal to i, we check whether there is a set of edges with j as an endpoint which have exactly the same labels as the edges in F; will take time at most $O(|V| \times |E|)$.

□

5.2 Analysis for the generation complexity

The previous section focused on computing C_D, showing that its computational complexity of this problem is contingent on the structure of our description graph $\mathcal{G}$. Through this section we will show that C_W can be computed efficiently in polynomial time, as it is easily expressible as a single source shortest path computation on the causation graph G_c.

We begin by recalling the problem as it was given in Section 4.2.3:

Definition 5.14.
Input: A causation graph $\mathcal{G}_c = (V_c, E_c, \mathfrak{C})$ and an instance $i \in V_c$
Output: The cheapest path to $\pi^*(i)$ from any initial condition, together with its cost $\hat{C}(\varphi_\pi(i))$.

We recall that the complexity of a path $\pi^*(i)$ for an instance $i \in V_c$ as defined in Definition 4.34 is nothing more than the sum of all costs along the path π: this means that we are looking for the cheapest possible path from any possible initial condition to the target instance i.

To solve this problem in polynomial time, we can simply add an additional source node s^*, and a set of cost zero edges from s^* to every initial condition node s_i. Formally, we construct the following graph:

$$\mathcal{G}_c' = (V^*, E^*, l, n)$$

where $V^* = V_c \cup \{s^*\}$, $E^* = E_c \cup \{(s^*, s_i) \mid s_i \text{ is an inital condition}\}$, the functions l, n are unchanged for all edges already in $\mathcal{G}_c$ and $l((s^*, s_i)) = 0$ for all initial conditions s_i.

Proposition 5.15. Computing the weighted shortest path from s^* to a target instance i is equivalent to computing the cheapest path from any given s_i to i.

Proof. The proof is straightforward: let π^* be the weighted shortest path π^* from s^* to i, then since the cost of the first edge on the path is 0, we know that the cost of π^* is the least amongst all possible paths from s^* to i. Since the cost of the path formula is defined by summing over the edges in π^*, we know that

$$\sum_{e \in \pi^*} l(e) = l(s^*, i) + \sum_{e \in \pi^* - (s^*, i)} = 0 + \sum_{e \in \pi^* - (s^*, i)} l(e) = \hat{C}\Big(\varphi_{\pi^* - (s^*, i)}(i)\Big)$$

Note that the minimality of the path formula $\varphi_{\pi^* - (s^*, i)}$ follows immediately by the minimality of path π^*. □

Remark. Since we have reduced our C_D computation to a weighted shortest path problem, we can use a variety of polynomial time algorithms to find a cheapest path formula: as an example, we may want to use Dijsktra's algorithm, which will return an exact solution in $O(|V^*|^2)$.

Chapter 6

Conclusion and Future Work

This thesis provided a formal introduction to CompLogDL, a computational framework to assess the unexpectedness of a given event in terms of bounded Kolmogorov Complexities instead of probabilities. We built on existing work by Sileno and Dessalles [1], which first introduced a blueprint for this a programming framework rooted in Simplicity Theory [7, 63]. Our contributions can be summarised as follows:

We first presented a conceptual argument to justify the need for a quantitative, but not probabilistic framework to predict surprise; we did so by arguing that surprise should not be identified with low probability events, and that the phenomena of awareness growth renders Bayesian approaches unable to deal with predicting unexpectedness.

As an alternative, we introduced the main tenets of Simplicity Theory and presented CompLogDL, a novel computational framework capable of calculating unexpectedness through the difference between descriptive and generation complexities, which need not be interpreted probabilistically. We revisited earlier design choices for computing unexpectedness, gave a formal specification of a CompLogDL program and its inference problems, which build on existing work in Referring Expression Generation [72, 73]. Finally, we analysed the complexity of CompLogDL's inference by looking at each of its components.

Future Work

Finally, we highlight possible directions for future work:

1. The computation of CompLogDL relies on annotated description logics: we decided to use $\mathcal{EL}$ to guarantee efficient reasoning, but the same computations could in principle be performed using more expressive description logics [65], in particular those which include negation and union.
2. We have assumed that a CompLogDL (descriptive) program is materialised, but this need not be the case: an interesting further line of research could be setting conditions for how an arbitrary annotated knowledge base could be materialised: a possible starting point could be similar work with subsumption in the case of probabilistic description logics (see for example [88–90]).
3. We have given a particular example of a complexity function to build some intuition, but the framework as presented leaves room to explore different measures: it could be interesting to explore how different complexity measures impact unexpectedness computations.
4. We have given a framework which is quantitative in nature, yet not probabilistic: an interesting question would be to explore a version of CompLogDL where the annotated

knowledge base is constructed via a probabilistic description logic, such as Prob-$\mathcal{EL}$[91], or to use other annotated description logics, such as fuzzy [92] or typicality [93] oriented approaches; we hypothesize that the latter may be a more plausible cognitive representation. Similarly, we could replace our causation graph with a probabilistic counterpart, such as the ones presented in [74].

5. At the moment, we have defined our referring expression in such a way that the referring graph generating it will be a star, but this need not be the case in principle. It could be an interesting line of research to relax this assumption, and assessing the computational complexity of this problem for various types of graphs. We hypothesize that, for the case in which the generating subgraph is allowed to be any connected subgraph, the problem lies above the class **NP**: similar results have shown that for the restricted case in which the parent subgraph is a tree [94], this problem is **NP**-Complete.
6. Finally, it would be interesting to develop an implementation of CompLogDL: for this purpose, we suggest to model Referring Expression Component in ASP, as this framework handles NP problems elegantly. It would be particularly interesting to integrate this with existing ontologies, as running the full pipeline on such realistic inputs will move the framework from proof-of-concept to experimentally validated tool, paving the way for user studies on perceived surprise and for embedding CompLogDL in larger AI systems.

Chapter 7

Bibliography

1. Sileno G, Dessalles J-L (2023) From Probabilistic Programming to Complexity-based Programming
2. Aristotle, Roberts WR (2004) Rhetoric. Courier Corporation
3. Reisenzein R, Horstmann G, Schützwohl A (2019) The Cognitive-Evolutionary Model of Surprise: A Review of the Evidence. Topics in Cognitive Science 11:50–74. https://doi.org/10.1111/tops.12292
4. Teigen KH, Keren G (2003) Surprises: Low Probabilities or High Contrasts?. Cognition 87:55–71. https://doi.org/10.1016/s0010-0277(02)00201-9
5. Ortony A, Partridge D (1987) Surprisingness and Expectation Failure: What's the Difference?. In: Proceedings of the 10th International Joint Conference on Artificial Intelligence - Volume 1. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, pp 106–108
6. Meyer W-U, N Michael, R Udo, and Schützwohl A (1991) An Experimental Analysis of Surprise. Cognition and Emotion 5:295–311. https://doi.org/10.1080/02699939108411042
7. Dessalles J-L (2008) La Pertinence et Ses Origines Cognitives - Nouvelles Théories. Hermes-Science Publications
8. Li M, Vitányi P (2019) An Introduction to Kolmogorov Complexity and Its Applications. Springer International Publishing, Cham
9. Delarue S, Viard T, Dessalles J-L (2023) Unexpected Attributed Subgraphs: A Mining Algorithm. In: Proceedings of the International Conference on Advances in Social Networks Analysis and Mining. ACM, Kusadasi Turkiye, pp 171–178
10. Houzé É, Dessalles J-L, Diaconescu A, Menga D (2022) What Should I Notice? Using Algorithmic Information Theory to Evaluate the Memorability of Events in Smart Homes. Entropy 24:346
11. Murena PA, Al-Ghossein M, Dessalles JL, Cornuéjols A (2020) Solving Analogies on Words Based on Minimal Complexity Transformation. In: International Joint Conference on Artificial Intelligence. IJCAI, pp 1848–1854
12. Dessalles J-L (2011) Simplicity Effects in the Experience of Near-Miss. In: Society CS (ed) Proceedings of the 33rd Annual Conference of the Cognitive Science Society. Austin, TX, United States, pp 408–413

13. Dessalles J-L (2008) Coincidences and the Encounter Problem: A Formal Account. In: Society CS (ed) Proceedings of the 30th Annual Conference of the Cognitive Science Society. Washington, United States, pp 2134–2139
14. Sileno G, Saillenfest A, Dessalles JL (2017) A Computational Model of Moral and Legal Responsibility via Simplicity Theory. In: The 30th International Conference on Legal Knowledge and Information Systems
15. De Raedt L, Kimmig A, Toivonen H (2007) ProbLog: A Probabilistic Prolog and Its Application in Link Discovery. In: Proceedings of the 20th International Joint Conference on Artifical Intelligence. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, pp 2468–2473
16. Savage LJ (1972) The Foundations of Statistics. Dover Publications
17. Murphy KP, Murphy KP (2012) Machine Learning: A Probabilistic Perspective. MIT Press, Cambridge, MA
18. Brémaud P (2020) Probability Theory and Stochastic Processes, First Edition. Springer Cham
19. Hájek A (2023) Interpretations of Probability. The Stanford Encyclopedia of Philosophy
20. Linde W (2016) Probability Theory: A First Course in Probability Theory and Statistics. Walter de Gruyter GmbH & Co KG
21. Blitzstein JK, Hwang J (2014) Introduction to Probability. CRC Press
22. Foundations of the Theory of Probability : A. N. Kolmogorov : Free Download, Borrow, and Streaming : Internet Archive
23. Lin H (2024) Bayesian Epistemology. The Stanford Encyclopedia of Philosophy
24. William M. Bolstad JMC (2016) Introduction To Bayesian Statistics, Third Edition. John Wiley & Sons
25. Zabell S (2009) Philosophy of Inductive Logic: The Bayesian Perspective. The Development of Modern Logic 0
26. De Canson C (2024) The Nature of Awareness Growth. Philosophical Review 133:1–32. https://doi.org/10.1215/00318108-10880419
27. Leitgeb H, Pettigrew R (2010) An Objective Justification of Bayesianism II: The Consequences of Minimizing Inaccuracy. Philosophy of Science 77:236–272. https://doi.org/10.1086/651318
28. Ramsey FP (2016) Truth and Probability. Readings in Formal Epistemology: Sourcebook 21–45
29. Finetti B de (1937) La prévision : ses lois logiques, ses sources subjectives. Annales de l'institut Henri Poincaré 7:1–68
30. Ranyard RH (1977) Risky Decisions Which Violate Transitivity and Double Cancellation. Acta Psychologica 41:449–459. https://doi.org/10.1016/0001-6918(77)90003-8

31. SURPRISE Definition & Meaning - Merriam-Webster
32. Smith A (1982) Essays on Philosophical Subjects. Indianapolis : Liberty Classics
33. Descartes R (1649) The Passions of the Soul. H. Legras, Paris
34. Horstmann G, Schützwohl A (2024) Surprise. Emotion Theory: The Routledge Comprehensive Guide
35. Charlesworth WR (1969) The Role of Surprise in Cognitive Development. Studies in cognitive development 257–314
36. Rumelhart DE (1984) Schemata and the Cognitive System. Handbook of Social Cognition, Vol 1. 161–188
37. Alba JW, Hasher L (1983) Is Memory Schematic?. Psychological Bulletin 93:203–231. https://doi.org/10.1037/0033-2909.93.2.203
38. Dahlin B (2001) Critique of the Schema Concept. Scandinavian Journal of Educational Research 45:287–300. https://doi.org/10.1080/00313830120074215
39. Thorndyke PW, Yekovich FR (1980) A Critique of Schema-Based Theories of Human Story Memory. Poetics 9:23–49. https://doi.org/10.1016/0304-422X(80)90011-X
40. Anderson JR (2005) Cognitive Psychology and Its Implications. Worth Publishers
41. Savoie D, Ladouceur R (1995) Évaluation et Modification de Conceptions Erronées Au Sujet Des Loteries. [Evaluation and Modification of Misconceptions Concerning Lotteries.]. Canadian Journal of Behavioural Science / Revue canadienne des sciences du comportement 27:199–213. https://doi.org/10.1037/0008-400X.27.2.199
42. Dessalles J-L (2011) A Structural Model of Intuitive Probability
43. Schiff LI (1960) On Experimental Tests of the General Theory of Relativity. American Journal of Physics 28:340–343. https://doi.org/10.1119/1.1935800
44. Baum R, Sheehan W (2003) In Search Of Planet Vulcan: The Ghost In Newton's Clockwork Universe. Basic Books
45. Banach S, Tarski A (1924) Sur la décomposition des ensembles de points en parties respectivement congruentes. Fundamenta Mathematicae 6:244–277
46. Modica S, Rustichini A (1994) Awareness and Partitional Information Structures. Theory and Decision 37:107–124. https://doi.org/10.1007/BF01079207
47. Pettigrew R (2024) How Should Your Beliefs Change When Your Awareness Grows?. Episteme 21:733–757. https://doi.org/10.1017/epi.2022.33
48. Earman J (1996) Bayes or Bust?, Second Edition. The MIT Press
49. Salmon WC (2005) Rationality and Objectivity in Science. In: Reality And Rationality. Oxford University PressNew York, NY, pp 93–116
50. Bradley R (2017) Decision Theory with a Human Face. Cambridge University Press

51. Karni E, Vierø M-L (2013) "Reverse Bayesianism": A Choice-Based Theory of Growing Awareness. American Economic Review 103:2790–2810. https://doi.org/10.1257/aer.103.7.2790
52. Roussos J (2021) Awareness Growth and Belief Revision
53. Steele K, Stefánsson HO (2021) Belief Revision for Growing Awareness. Mind 130:1207–1232. https://doi.org/10.1093/mind/fzaa056
54. Mahtani A (2020) Awareness Growth and Dispositional Attitudes. Synthese 198:8981–8997. https://doi.org/10.1007/s11229-020-02611-5
55. Shimony A (1970) Scientific Inference
56. Glymour CN (1980) Theory and Evidence. Princeton, N.J. : Princeton University Press
57. Dowe P, Salmon MH (2005) Rationality and Objectivity in Science. In: Reality And Rationality. Oxford University PressNew York, NY, pp 93–116
58. Henderson L, Goodman ND, Tenenbaum JB, Woodward JF (2010) The Structure and Dynamics of Scientific Theories: A Hierarchical Bayesian Perspective. Philosophy of Science 77:172–200. https://doi.org/10.1086/651319
59. Wenmackers S, Romeijn J-W (2016) New Theory about Old Evidence. Synthese 193:1225–1250. https://doi.org/10.1007/s11229-014-0632-x
60. Pollock JL (1983) Epistemology and Probability. Synthese 55:231–252. https://doi.org/10.1007/BF00485070
61. Dessalles J-L (2008) La Pertinence et Ses Origines Cognitives - Nouvelles Théories. Hermes-Science Publications
62. Shannon CE (1948) A Mathematical Theory of Communication. Bell System Technical Journal 27:379–423. https://doi.org/10.1002/j.1538-7305.1948.tb01338.x
63. Simplicity Theory
64. Dessalles J-L (2008) How Can an Eclipse Be Unexpected?
65. Baader F, Calvanese D, McGuinness DL, et al (2007) The Description Logic Handbook: Theory, Implementation and Applications, 2nd ed. Cambridge University Press, Cambridge
66. Baader F, Brandt S, Lutz C (2005) Pushing the EL Envelope. In: Proceedings of the 19th International Joint Conference on Artificial Intelligence. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, pp 364–369
67. Arora S, Barak B (2009) Computational Complexity: A Modern Approach. Cambridge University Press
68. Computers and Intractability; A Guide to the Theory of NP-Completeness: | Guide Books | ACM Digital Library
69. Papadimitriou CH (2003) Computational Complexity. Encyclopedia of Computer Science 260–265

70. Garey MR, Johnson DS (1990) Computers and Intractability; A Guide to the Theory of NP-Completeness. W. H. Freeman & Co., USA

71. Gasarch WI, Krentel MW, Rappoport KJ (1995) OptP as the Normal Behavior of NP-complete Problems. Mathematical Systems Theory 28:487–514. https://doi.org/10.1007/bf01204168

72. Areces C, Koller A, Striegnitz K (2008) Referring Expressions as Formulas of Description Logic. In: White M, Nakatsu C, McDonald D (eds) Proceedings of the Fifth International Natural Language Generation Conference. Association for Computational Linguistics, Salt Fork, Ohio, USA, pp 42–49

73. Krahmer E, Erk SV, Verleg A (2003) Graph-Based Generation of Referring Expressions. Computational Linguistics 29:53–72. https://doi.org/10.1162/089120103321337430

74. Pearl J (1995) Causal Diagrams for Empirical Research. Biometrika 82:669–688. https://doi.org/10.2307/2337329

75. Halpern JY, Pearl J (2005) Causes and Explanations: A Structural-Model Approach. Part I: Causes. The British Journal for the Philosophy of Science 56:843–887. https://doi.org/10.1093/bjps/axi147

76. Dale R (1989) Cooking up Referring Expressions. In: Proceedings of the 27th Annual Meeting on Association for Computational Linguistics. Association for Computational Linguistics, USA, pp 68–75

77. Grice HP (1975) Logic and Conversation

78. Dale R, Reiter E (1995) Computational Interpretations of the Gricean Maxims in the Generation of Referring Expressions. Cognitive Science 19:233–263. https://doi.org/10.1016/0364-0213(95)90018-7

79. Krahmer E, van Deemter K (2012) Computational Generation of Referring Expressions: A Survey. Computational Linguistics 38:173–218. https://doi.org/10.1162/COLI_a_00088

80. Kees van Deemter AG (2007) Incremental Generation of Plural Descriptions: Similarity and Partitioning. Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning 102–111

81. Van Deemter K (2006) Generating Referring Expressions That Involve Gradable Properties. Computational Linguistics 32:195–222. https://doi.org/10.1162/coli.2006.32.2.195

82. Riguzzi F, Bellodi E, Lamma E, Zese R (2015) Probabilistic Description Logics under the~Distribution Semantics. Semantic Web 6:477–501. https://doi.org/10.3233/SW-140154

83. Pearl J (2009) Causality. Cambridge University Press

84. Imbens GW, Rubin DB (2015) Causal Inference for Statistics, Social, and Biomedical Sciences: An Introduction. Cambridge University Press, Cambridge

85. Peters J, Janzing D, Schlkopf B (2017) Elements of Causal Inference: Foundations and Learning Algorithms. The MIT Press

86. Lewis D (1973) Causation. The Journal of Philosophy 70:556–567. https://doi.org/10.2307/2025310

87. Halpern JY (2016) Actual Causality. The MIT Press

88. Ceylan İİ, Lukasiewicz T (2018) A Tutorial on Query Answering and Reasoning over Probabilistic Knowledge Bases. Reasoning Web. Learning, Uncertainty, Streaming, and Scalability: 14th International Summer School 2018, Esch-sur-Alzette, Luxembourg, September 22–26, 2018, Tutorial Lectures 35–77

89. Koller D, Levy A, Pfeffer A (1997) P-CLASSIC: A Tractable Probablistic Description Logic. In: Proceedings of the Fourteenth National Conference on Artificial Intelligence and Ninth Conference on Innovative Applications of Artificial Intelligence. AAAI Press, Providence, Rhode Island, pp 390–397

90. Lukasiewicz T, Straccia U (2008) Managing Uncertainty and Vagueness in Description Logics for the Semantic Web. Journal of Web Semantics 6:291–308. https://doi.org/10.1016/j.websem.2008.04.001

91. Gutiérrez Basulto V, Jung JC, Lutz C, Schröder L (2011) A Closer Look at the Probabilistic Description Logic Prob-EL. Proceedings of the AAAI Conference on Artificial Intelligence 25:197–202. https://doi.org/10.1609/aaai.v25i1.7856

92. Straccia U (2001) Reasoning within Fuzzy Description Logics. Journal of Artificial Intelligence Research 14:137–166. https://doi.org/10.1613/jair.813

93. Lieto A, Pozzato GL (2018) A Description Logic of Typicality for Conceptual Combination. In: Ceci M, Japkowicz N, Liu J, et al (eds) Foundations of Intelligent Systems. Springer International Publishing, Cham, pp 189–199

94. Wang Y-K, Chen W-M, Li C-T, Lin S-D (2015) Identifying Smallest Unique Subgraphs in a Heterogeneous Social Network. In: 2015 IEEE International Conference on Big Data (Big Data). IEEE, Santa Clara, CA, pp 757–766