CompLogDL and its Implementation in Answer Set Programming (Extended Abstract)

Lucrezia Mosconi and Johannes P. Wallner

Graz University of Technology
lucrezia.mosconi@tugraz.at, johannes.p.wallner@tugraz.at

Surprise is a universal human experience which has been a topic of research in psychology and related disciplines for quite some time. While the commonly accepted view considers the surprisal value of an event as related to its probability [12], there are theories which reject this claim, such as Simplicity Theory [4]. According to the latter, an event is surprising if it is simpler to describe than it is to generate: this cognitive theory has been tested in experimental settings [5], and has applications ranging from smart home design [7] to subgraph mining [3]. Here, we build on previous work [11], and introduce a computational framework called CompLogDL. This system relies on the specification of a mental and world model, on which the description and generation complexity of an instance are computed, respectively. The former is computed via a description logic knowledge base [1], while the latter is implemented as a shortest path computation. We provide complexity results and an implementation in answer set programming (ASP) [6, 2, 10]. This presentation is based on [9] and extends it.

CompLogDL

The scope of CompLogDL is to compute the extent to which a given event is unexpected: given an event s, we compute the unbounded quantity $U(s) := C_W(s) - C_D(s)$. 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 up of two different components, as the computations of C_D and C_W are independent of each other.

Computing C_D The computation of an event's description complexity depends on the *descriptive* program, which is a set of annotated $\mathcal{EL}$ A-Box and T-Box axioms, representing the agent's knowledge base; the program is then used to construct a description graph. The problem is then cast as a minimum referring expression computation which builds on similar graph-based approaches in the literature [8]. The descriptive program is defined as $\mathbb{P}_D := \langle(\mathcal{T}, \mathcal{A}),\ w : \mathcal{T} \cup \mathcal{A} \to (0, 1]\rangle$. $\mathcal{T}$ is a set of concept subsumptions and definitions (i.e., statements of the form $C \sqsubseteq D$, $C \equiv D$), while $\mathcal{A}$ is a set of concept assertions (i.e., statements of the form $C(a)$ for some instance a). Finally $w : \mathcal{T} \cup \mathcal{A} \to (0, 1]$ is a cost function, specifying the degree to which concept assertions, subsumptions and definitions are accessible to 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 $\mathfrak{C} : \mathcal{A} \to \mathbb{R}^+$. Its purpose is to assign complexity values to concept and role assertions which depend indirectly on the cost function w, but it will return a positive real number representing the complexity of a given assertion.

Given a descriptive program $\mathbb{P}_D$ and a complexity function, we can now compute a minimum referring expression for said instance. We construct a description graph where concept assertions are interpreted as weighted (and labelled) loops, while roles are directed, weighted (and labelled) edges. The problem is then formally defined as follows: given a description graph $\mathcal{G} = (V, E, l, n)$ an instance $i \in V$, return the least cost star subgraph F^* of $\mathcal{G}$ such that, for all other connected subgraphs F' of $\mathcal{G}$, F^* is not isomorphic to F'.

We show that the decision problem equivalent to this optimisation problem is *NP*-complete, independently of whether the parent graph only contains loops, or also edges. Similarly, the optimisation problem is FP^{NP}-complete.

Computing C_W The computation of an event's generation complexity depends on a set of annotated tuples (the *causal* program), which are used to form a graph representing the (actual) causal dependencies in a set of events: the problem is then cast as a multiple source shortest path problem. The causal program $\mathbb{P}_W = (E, R \subset E \times E, w : R \to (0, 1])$ specifies a set of dependencies over a set of events, together with a weight function representing the extent to which the events are causally related, according to the agent. In order to construct the causation graph, we make use of a generation complexity function $\mathfrak{C} : R \to \mathbb{R}^+$: the graph will then be an edge-weighted graph, where the cost of the edges is given by $\mathfrak{C}$. Given any node $e_i \in E_c$ in the causation graph, 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.

The unexpectedness of an instance s is then the difference between the cost of a shortest weighed path obtained via a program's causation graph, and the complexity of the minimum unique referring expression for instance s, obtained via the description graph. 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.

ASP Encoding

We provide an ASP encoding of the FP^{NP}-complete optimization problem of computing a minimum referring expression. In brief, the given graph $\mathcal{G}$ is represented as ASP facts, the subgraph is non-deterministically constructed using choice rules. Auxiliary rules and constraints ensure that the result is a star graph and rule out solutions with isomorphic copies. Finally, an ASP minimization statement ensures minimum cost.

Acknowledgements

This research was funded in whole or in part by the Austrian Science Fund (FWF) 10.55776/COE12.

References

1. Baader, F., Calvanese, D., McGuinness, D.L., Nardi, D., Patel-Schneider, P.F. (eds.): The Description Logic Handbook: Theory, Implementation and Applications. Cambridge University Press, 2 edn. (2007)
2. Brewka, G., Eiter, T., Truszczynski, M.: Answer set programming at a glance. Communications of the ACM **54**(12), 92–103 (2011)
3. Delarue, S., Viard, T., Dessalles, J.: Unexpected attributed subgraphs: a mining algorithm. In: Prakash, B.A., Wang, D., Weninger, T. (eds.) Proc. ASONAM. pp. 171–178. ACM (2023)
4. Dessalles, J.L.: La Pertinence et Ses Origines Cognitives - Nouvelles Théories. Hermes-Science Publications (2008)
5. Dessalles, J.: A structural model of intuitive probability. CoRR **abs/1108.4884** (2011), http://arxiv.org/abs/1108.4884
6. Gelfond, M., Lifschitz, V.: The stable model semantics for logic programming. In: Proc. ICLP/SLP. pp. 1070–1080. MIT Press (1988)
7. Houzé, É., Dessalles, J., Diaconescu, A., Menga, D.: What should I notice? using algorithmic information theory to evaluate the memorability of events in smart homes. Entropy **24**(3), 346 (2022)
8. Krahmer, E., Erk, S.V., Verleg, A.: Graph-Based Generation of Referring Expressions. Computational Linguistics **29**(1), 53–72 (2003)
9. Mosconi, L.: The Unexpected side of Logic Programming: Introducing CompLog DL. Master's thesis, Universiteit van Amsterdam (2025)
10. Niemelä, I.: Logic programs with stable model semantics as a constraint programming paradigm. Annals of Mathematics and Artificial Intelligence **25**(3-4), 241–273 (1999)
11. Sileno, G., Dessalles, J.L.: From Probabilistic Programming to Complexity-Based Programming. In: Nowaczyk, S., Biecek, P., Chung, N.C., Vallati, M., Skruch, P., Jaworek-Korjakowska, J., Parkinson, S., Nikitas, A., Atzmüller, M., Kliegr, T., Schmid, U., Bobek, S., Lavrac, N., Peeters, M., van Dierendonck, R., Robben, S., Mercier-Laurent, E., Kayakutlu, G., Owoc, M.L., Mason, K., Wahid, A., Bruno, P., Calimeri, F., Cauteruccio, F., Terracina, G., Wolter, D., Leidner, J.L., Kohlhase, M., Dimitrova, V. (eds.) Artificial Intelligence. ECAI 2023 International Workshops. pp. 304–317. Springer Nature Switzerland (2024)
12. Teigen, K.H., Keren, G.: Surprises: Low probabilities or high contrasts? Cognition **87**(2), 55–71 (2003)

Appendix: ASP Encoding Details

We provide an ASP encoding of computing minimum weight referring expression below. Input facts represent an example for a given description.

```
% Input facts
vertex(i;a;b;c;d).
edge(i,a). edge(i,b). edge(i,c). edge(i,d).
label(i,a,la). label(i,b,lb). label(i,c,lb). label(i,d,lc).
cost(i,a,2). cost(i,b,1). cost(i,c,1). cost(i,d,3).
target(i).

% All edges are labelled and assigned a cost
← edge(X,Y), not label(X, Y, _).
← edge(X,Y), not cost(X, Y, _).

% Pick Edges in F
{ in(X,Y) : edge(X,Y) }.

%F must be a star graph
in(I) ← target(I).
in(X) ← in(X,_).
in(Y) ← in(_,Y).
← in(X,Y), target(I), X != I, Y != I.

%Minimize Cost
total_cost(T) ← T = #sum { C,Y : in(_,Y), cost(_,Y,C) }.
#minimize { C,Y : in(X,Y), cost(_,Y,C) }.

%Exclude solutions with isomorphic copies
label_count(L,N) ← label(_,_,L), N = #count { Y : in(X,Y), label(X,Y,L) }.
label_used(L) ← label_count(L,_).

label_capacity(J,L,N) ← label(_,_,L), vertex(J), target(I), J != I,
                        N = #count { Y : edge(J,Y), label(J,Y,L) }.

insufficient_label(J) ← label_count(L,Need), label_capacity(J,L,Have),
                        Have < Need.

isomorphic_star(J) ← vertex(J), target(I), J != I, label_used(_),
                     not insufficient_label(J).

← isomorphic_star(J).
```