Proceedings of the 18th International Conference on Principles of Knowledge Representation and Reasoning
Main Track - Short paper

Introducing k-lingo: a k-depth Bounded Version of ASP System Clingo

Fabio Aurelio D'Asaro , Paolo Baldi , Giuseppe Primiero
Logic Group, Department of Philosophy, University of Milan
{fabio.dasaro, paolo.baldi, giuseppe.primiero}@unimi.it

Abstract

Depth-Bounded Boolean Logics (*DBBL* for short) are well-understood frameworks to model rational agents equipped with limited deductive capabilities. These Logics use a parameter $k \geq 0$ to limit the amount of *virtual information*, i.e., the information that the agent may temporarily assume throughout the deductive process. This restriction brings several advantageous properties over classical Propositional Logic, including polynomial decision procedures for deducibility and refutability. Inspired by DBBL, we propose a limited-depth version of the popular ASP system `clingo`, tentatively dubbed k-`lingo` after the bound k on virtual information. We illustrate the connection between DBBL and ASP through examples involving both proof-theoretical and implementative aspects. The paper concludes with some comments on future work, which include a computational complexity characterization of the system, applications to multi-agent systems and feasible approximations of probability functions.

1 Introduction

Depth-Bounded Boolean Logics (*DBBL* for short) are logic formalisms to reason about agents that have limited deductive power. These logics build upon definitions for *virtual* and *actual* information. The former is information an agent may temporarily assume in order to perform logical deductions, whereas the latter is information that is evident to the reasoning agent. Proof-theoretically, virtual information corresponds to applying the *cut* (or *Principle of Bivalence*) rule. Intuitively, an agent reasoning at depth k can apply at most k nested instances of the cut rule. This results in a hierarchy of tractable approximations of classical propositional logic (D'Agostino, Finger, and Gabbay 2013).

This paper aims to port the very idea of Depth-Bounded Reasoning to *Answer Set Programming* (*ASP* for short), a common logic programming paradigm. In particular, we propose a modification of the popular ASP system `clingo` (Gebser et al. 2014), tentatively dubbed k-`lingo` to emphasize its dependency on the parameter k which determines the maximum reasoning depth. In the following, we will refer e.g. to the 2-depth version of such system as 2-`lingo` (and similarly for other values of k).

The present work briefly shows how our implementation relates to Depth-Bounded Boolean Logics and discusses some differences, most notably that DBBL is monotonic in nature, whereas ASP is non-monotonic. We also demonstrate how this implementation may be used for practical applications and hint at future developments that we believe will push forward our understanding of DBBL by providing an efficient tool for the computation of depth-bounded consequence and clarifying the interaction between non-monotonic and depth-bounded reasoning.

θ	ϕ	$\theta \wedge \phi$	$\theta \vee \phi$	$\theta \rightarrow \phi$	$\neg\theta$
0	0	0	0	1	1
0	1	0	1	1	1
0	$\bot$	0	$\bot$	1	1
1	0	0	1	0	0
1	1	1	1	1	0
1	$\bot$	$\bot$	1	$\bot$	0
$\bot$	0	0	$\bot$	$\bot$	$\bot$
$\bot$	1	$\bot$	1	1	$\bot$
$\bot$	$\bot$	$\bot, 0$	$\bot, 1$	$\bot, 1$	$\bot$

Table 1: 3ND truth-tables

2 Background and Related Work

Although we are not concerned here with providing an in-depth formal discussion of the relation between DBBL and ASP, it is helpful nonetheless to recall some concepts from these fields.

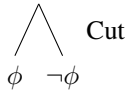
DBBL's semantics is given in terms of *three valued non-deterministic* (*3ND* for short) *truth tables* (see Table 1). Intuitively, the truth values represent *informational* versions of their classical counterparts. In other words, 1 (resp. 0) may be interpreted as "*holding* the information that a sentence is true" (resp. "false"). The third truth value, $\bot$ (not to be confused with the usual symbol for "falsum"), represents *informational indeterminacy*, that is, "*not holding* information about the truth value of a sentence". As it may be noted from the truth tables, non-determinism arises in cases where the value of two conjuncts (resp. disjuncts) is informationally indeterminate. In this case, the value of their conjunction (resp. disjunction) non-deterministically takes value $\bot$ or 0 (resp. $\bot$ or 1). This fact, which may look counterintuitive at first glance, has its justification in that we may carry additional information about the possibility of both conjuncts

(resp. disjuncts) to be *simultaneously* true. To put it with Quine (Quine 1976), we may see a small animal and we may not hold information about the two sentences "it is a mouse" and "it is a chipmunk". Nonetheless, we would surely deny their conjunction. However, we could suspend our judgement about the conjunction of two informationally indeterminate sentences "it is a mouse" and "it is in the kitchen".

The semantics of arbitrary formulas is given by *3ND-valuations*, i.e., mappings satisfying $V(\neg\theta) = f_{\neg}(\theta)$, $V(\theta \wedge \phi) = f_{\wedge}(\theta, \phi)$, $V(\theta \vee \phi) = f_{\vee}(\theta, \phi)$ and $V(\theta \rightarrow \phi) = f_{\rightarrow}(\theta, \phi)$ where functions $f_{\neg}$, $f_{\wedge}$, $f_{\vee}$ and $f_{\rightarrow}$ are defined according to Table 1. The associated semantic consequence relation, denoted by $\models_0$, is defined as usual, i.e., $X \models_0 \phi$ if all 3ND-valuations satisfying all formulas in the set X also satisfy ϕ.

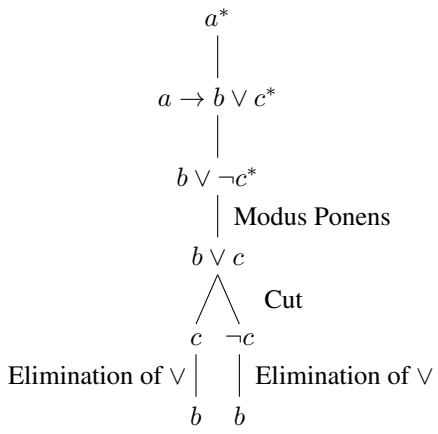
This semantic concept has a natural proof-theory that is a mixture of classic natural deduction and a tableau method. The full system can be found in (D'Agostino, Finger, and Gabbay 2013). This calculus consist of classical introduction and elimination rules for connectives (*intelim* rules for short). Intelim rules induce a deducibility relation $\vdash_0$ that is shown to be sound and complete with respect to $\models_0$, i.e., $X \models_0 \phi$ iff $X \vdash_0 \phi$.

It is possible to obtain completeness with respect to classical propositional logic by adding a *cut* rule (or *PB* rule, after the Principle of Bivalence) that corresponds to "guessing" a formula. The cut rule has the following "branching" form:



for some formula ϕ.

Separating intelim rules from the cut rule allows one to clearly distinguish between information the agent is aware of (*actual information*) and consequences of such information the agent may be not aware of, due to its limited deductive power. The latter kind of information may be extracted from actual information only through applications of the cut rule. Since it requires "guessing" the truth value of a formula A, DBBL regards applications of the cut rule as a "difficult" task both from a cognitive and computational point of view. This constitutes the rationale behind the choice of limiting the number of nested cuts that an agent is able to perform during the reasoning process, thus allowing for a generalization of $\vdash_0$ to $\vdash_k$, where k is the maximum number of nested cuts allowed[1]. An important result of DBBL is that for any fixed $k \geq 0$, deducibility and refutability of formulas in $\vdash_k$ is polynomial. A simple example of 1-depth derivation of B from a theory $\mathcal{T} = \{a, a \rightarrow b \vee c, b \vee \neg c\}$ (where a, b, c are propositional variables) goes as follows:



where we marked formulas in the theory $\mathcal{T}$ with $*$ (and we use this notation convention in the remainder of the paper). Since both branches of the above tree end with b (regardless of whether c holds or not) we deliberate that $\mathcal{T} \vdash_1 b$, where $\vdash_1$ stands for the 1-depth consequence relation. Note that the above tree also licenses that $\mathcal{T} \vdash_0 b \vee c$, as this is derivable from the theory $\mathcal{T}$ by a single application of Modus Ponens and no cuts.

A taxonomy of tableau systems for ASP was given in (Gebser and Schaub 2006). These systems are similar to the proof-theory of DBBL in that they also consist of (non-monotonic) introduction and elimination rules, and a cut rule that is exactly similar to that of DBBL. Therefore, implementing the idea of k-depth reasoning in ASP by limiting the number of cuts appears quite natural. This constitutes the core idea of the present work. However, it must be noted that there are some important differences between the intelim rules of DBBL and the ASP tableau rules, mainly due to the non-monotonic nature of ASP and the monotonicity of DBBL. For instance, tableaux for ASP include a non-monotonic *Backward True Atom* rule of the form

$$\frac{a, \mathit{not}\ b_1, \ldots, \mathit{not}\ b_{i-1}, \mathit{not}\ b_{i+1}, \mathit{not}\ b_n}{b_i}\ \text{BTA}$$

where $b_1, \ldots, b_n$ are all bodies of rules with head a, and we use the operator *not* to distinguish non-monotonic negation and classical negation. This rule has no intelim equivalent in DBBL. Therefore, we must be careful when interpreting the meaning of truth values in depth-bounded ASP, which may be seen as an *informational* counterpart of truth values in non-monotonic logics. We touch upon these differences in the following section by means of examples.

3 The k-`lingo` System

Our depth-bounded ASP system is built upon the popular ASP grounder and solver `clingo`, which exploits the *conflict-driven nogood learning* (*CDNL* for short) solving strategy. The proof-system underlying CDNL is based on two expansion rules: a *decision* rule and a *propagation* rule. The decision rule is nothing but the cut rule discussed in Section 2. The propagation rule is used to deterministically assign atoms a truth value according to the ASP tableau deterministic rules, i.e., all tableau rules but the cut rule.

[1]The exact notation for k-depth consequence is $\vdash_k^f$, where f defines the *virtual space* of formulas that the agent is allowed to choose from when applying a cut. For simplicity, in this paper we follow the common ASP convention that such space consists of all atoms in the language, and simply write $\vdash_k$. Formal definitions of $\vdash_k^f$ and the associated semantic relation $\models_k^f$ can be found, e.g., in (D'Agostino 2015).

We use `clingo`'s Python API to limit the number of decisions/cuts to a fixed number $k \geq 0$. We dubbed the resulting system k-`lingo`[2]. The idea underlying k-`lingo` is to exploit `clingo`'s efficient solving strategy to look for an answer set of the input theory as we count the number of (nested) decisions the solver makes throughout the solving process, thus setting an upper limit on the *decision level* k-`lingo` can reach. If an answer set is found without exceeding decision level k, k-`lingo` returns it. Similarly, if the problem is shown to be unsatisfiable at decision level k, k-`lingo` returns `UNSATISFIABLE`. Otherwise, as soon as depth k is reached, we let the solver deterministically propagate literals until it reaches a *propagation fixpoint*, i.e., it is impossible to deduce new information unless a new nondeterministic decision is made. At this stage, `clingo` has assigned a truth value 0 or 1 to a subset of the atoms occurring in the input theory, with the other atoms left undetermined. Let A^1 be the set of atoms assigned truth value 1, A^0 be the set of atoms assigned truth value 0, and $A^\perp$ the set of atoms left undetermined. We then let k-`lingo` return a mapping from all atoms in the input theory to $\{0, 1, \perp\}$ such that $V(a) = 1$ if $a \in A^1$, $V(a) = 0$ if $a \in A^0$, and $V(a) = \perp$ if $a \in A^\perp$. We call this mapping a *3ND*-valuation* to distinguish it from standard 3ND-valuations in DBBL.

In the following we show a few example theories, discuss the corresponding output of k-`lingo`, and occasionally compare it to k-depth DBBL. Note that we refer to 0-depth k-`lingo` as 0-`lingo`, 1-depth k-`lingo` as 1-`lingo`, and similarly for other values of k.

Example 1. *Consider the following theory:*

$$T_1 = \left\{ \begin{array}{c} a \leftarrow b \\ b \end{array} \right\}$$

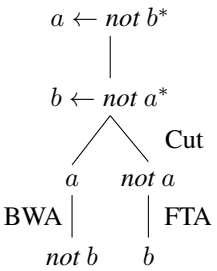
In this case, 0-`lingo` *computes the only 3ND*-valuation satisfying T_1, namely $V(a) = 1$ and $V(b) = 1$. Note that in this case the output of* 0-`lingo` *is consistent with 0-depth DBBL, as it may be easily verified from the 3ND truth table of implication in Table 1.*

In the next example, we show a subtle difference between the tableau systems of DBBL and ASP:

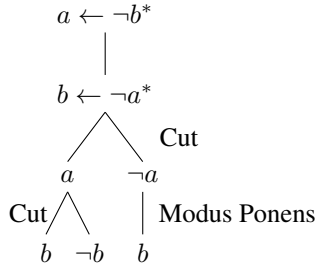
Example 2. *Let T_2 be as follows:*

$$T_2 = \left\{ \begin{array}{c} a \leftarrow \mathit{not}\ b \\ b \leftarrow \mathit{not}\ a \end{array} \right\}$$

0-`lingo` *finds the 3ND*-valuation $V(a) = \perp$ and $V(b) = \perp$.* 1-`lingo` *computes the 3ND*-valuation $V(a) = 0$ and $V(b) = 1$, which corresponds to the rightmost branch of the following ASP tableau:*



where BTA is the Backward True Atom rule discussed in Section 2, and FTA is the Forward True Atom rule (similar to Modus Ponens) discussed in (Gebser and Schaub 2006). Note that, in this case, the tableau of DBBL would be different:



The rightmost branch of the DBBL tableau is equivalent to the rightmost branch in the ASP tableau above. However, the leftmost branch of the DBBL tableau which corresponds to the 3ND-valuation $V'(a) = V'(b) = 1$ has no ASP equivalent. This is because of the different meaning assigned to negation and implication by ASP and DBBL.

k-`lingo` inherits `clingo`'s support to advanced constructs such as choice rules. In the example below, we briefly show how k-`lingo` behaves in their presence.

Example 3. *Let*

$$T_3 = \left\{ \begin{array}{c} \{a; b; c\} \\ d \leftarrow \mathit{not}\ c \end{array} \right\}$$

where $\{a; b; c\}$ is a choice rule.

1-`lingo` *finds the following valuation: $V(a) = \perp$, $V(b) = \perp$, $V(c) = 0$, $V(d) = 1$, which corresponds to "guessing" the falsity of c and deducing d.*

In the example below, we discuss how k-`lingo` behaves when it deals with unsatisfiable theories.

Example 4. *Consider the following unsatisfiable ASP theory:*

$$T_4 = \left\{ \begin{array}{c} d(a; b; c) \\ r(a; b) \\ 1\{f(X, Y) : r(Y)\}1 \leftarrow d(X) \\ \leftarrow 2\{f(X, Y)\}, r(Y) \end{array} \right\}$$

which encodes an instance of the pigeon hole problem, i.e. it looks for an injective mapping from a set $d = \{a, b, c\}$ to a set $r = \{a, b\}$. Interestingly, 0-`lingo` *is unable to detect that such problem is unsatisfiable, and outputs the 3ND*-valuation $V(f(a, a)) = V(f(b, a)) = V(f(c, a)) = V(f(a, b)) = V(f(b, b)) = V(f(b, c)) = \perp$. However, running* 1-`lingo` *on this theory produces output* `UNSATISFIABLE`.

[2]The latest version of k-`lingo` can be found on GitHub at `http://github.com/dasaro/klingo`

7	2358	2358	45	345	35	2568	1	9
4	6	2358	1	9	357	258	2358	238
1359	135	359	6	8	2	7	35	4
1358	9	23458	458	12456	1568	1268	2368	7
18	1278	278	3	126	1689	4	2689	5
1358	123458	6	7	1245	1589	1289	2389	1238
35689	34578	1	589	356	35689	25689	2456789	268
2	358	3589	589	7	4	15689	5689	168
5689	4578	45789	2	156	15689	3	456789	168

Figure 1: A "hard" sudoku. The small gray digits are the candidates for that cell as computed by 0-lingo.

In the following example we and show an application of k-lingo to a standard puzzle solving problem.

Example 5. *Let us now consider a* 9×9 *sudoku solver, which we encode with the following logic program:*

```
x(1..9). y(1..9). n(1..9).

1{ sudoku(X,Y,N): n(N) }1 :-
    x(X) ,y(Y).

subgrid(X,Y,A,B) :-
    x(X), x(A), y(Y), y(B),
    (X-1)/3 == (A-1)/3,
    (Y-1)/3 == (B-1)/3.

:- sudoku(X,Y,N), sudoku(A,Y,N), X!=A.
:- sudoku(X,Y,N), sudoku(X,B,N), Y!=B.
:- sudoku(X,Y,V), sudoku(A,B,V),
    subgrid(X,Y,A,B), X != A, Y != B.
```

where `sudoku(X,Y,N)` *means that cell* $(\texttt{X},\texttt{Y})$ *contains value* `N`*. The following sudoku instance cannot be solved by a* 0*-depth agent:*

```
sudoku(1,1,7). sudoku(1,8,1).
sudoku(1,9,9). sudoku(2,1,4).
sudoku(2,2,6). sudoku(2,4,1).
sudoku(2,5,9). sudoku(3,4,6).
sudoku(3,5,8). sudoku(3,6,2).
sudoku(3,7,7). sudoku(3,9,4).
sudoku(4,2,9). sudoku(4,9,7).
sudoku(5,4,3). sudoku(5,7,4).
sudoku(5,9,5). sudoku(6,3,6).
sudoku(6,4,7). sudoku(7,3,1).
sudoku(8,1,2). sudoku(8,5,7).
sudoku(8,6,4). sudoku(9,4,2).
sudoku(9,7,3).
```

In this case, 0-lingo *is only able to compute some straightforward consequences of the input problem. For instance, it figures cell* $(1,2)$ *cannot contain* 1*,* 4*,* 6*,* 7 *or* 9*. However, it remains agnostic as to whether it contains a* 2*,* 3*,* 5 *or* 8 *(see Figure 1). Finding a solution therefore requires quite some guesswork. Indeed, players sometimes need to apply advanced strategies to solve "hard" sudokus like this one, e.g., the "forcing chain" technique which requires using virtual information*[3]*. Note that inferences of* 0-lingo *depend on the input theory. For instance, in the given example, adding axioms expressing that each number must appear exactly once in every row, column and square would allow a* 0*-depth reasoning agent to conclude that cell* $(1,6)$ *contains a* 6*.*

Examples like this one suggest that we may take the minimum depth at which a problem can be satisfactorily solved as a coarse measure of its cognitive difficulty.

4 Conclusion and Future Work

In this short paper we introduce and present an ASP system based on clingo and inspired by DBBL. To the best of our knowledge, this is the first system implementing the idea at the core of DBBL, i.e. limiting the number of cuts throughout the reasoning process. In future work, we plan to establish a formal connection between ASP and DBBL and expand k-lingo to a flexible and fully-fledged framework for Depth-Bounded reasoning.

As a next step, we intend to provide a formal description of the logical framework, and characterize its computational complexity in the light of the solving techniques of clingo, i.e. CDNL. Efficient solving in a k-depth bounded setting may indeed bring several advantages. For example, as it was shown in (Baldi, D'Agostino, and Hosni 2020), it is possible to approximate probability functions using DBBL. Thus, k-lingo may serve as a tool to perform such approximation in practice. Moreover, since k-lingo is compatible with clingo's syntax, it can be applied to most existing ASP programs and frameworks straight out-of-the-box or, conversely, if one wants to test the practical implications of reasoning at k-depth in complex realistic domains, e.g. in the case of probabilistic temporal reasoning (D'Asaro et al. 2017; D'Asaro et al. 2020).

A further application of k-lingo is to multi-agent systems, where each agent is bound to a certain reasoning depth but can interact and receive additional information from agents who can reason at different depths (Cignarale and Primiero 2020). We aim to investigate these aspects further in future work.

[3]The "forcing chain" technique can be applied whenever two or more cells have multiple candidate values. In these cases, one may guess a candidate value for one of these cells and then verify whether the guessed value yields a contradiction. If this is the case, the guessed value can be removed from the set of candidates for that cell, eventually forcing one specific value.

Acknowledgments

This research was funded by the Department of Philosophy "Piero Martinetti" of the University of Milan under the Project "Departments of Excellence 2018-2022" awarded by the Ministry of Education, University and Research (MIUR). The authors also thankfully acknowledge the support of the Italian Ministry of Education, University and Research (PRIN 2017 project n. 20173YP4N3). The authors wish to thank Marcello D'Agostino for suggestions on how to improve parts of this paper, as well the three anonymous reviewer for some valuable comments on a previous version of this work.

References

Baldi, P.; D'Agostino, M.; and Hosni, H. 2020. Depth-bounded approximations of probability. In Lesot, M.-J.; Vieira, S.; Reformat, M. Z.; Carvalho, J. P.; Wilbik, A.; Bouchon-Meunier, B.; and Yager, R. R., eds., *Information Processing and Management of Uncertainty in Knowledge-Based Systems*, 607–621. Cham: Springer International Publishing.

Cignarale, G., and Primiero, G. 2020. A multi-agent depth bounded boolean logic. In Cleophas, L., and Massink, M., eds., *Software Engineering and Formal Methods. SEFM 2020 Collocated Workshops - ASYDE, CIFMA, and CoSim-CPS, Amsterdam, The Netherlands, September 14-15, 2020, Revised Selected Papers*, volume 12524 of *Lecture Notes in Computer Science*, 176–191. Springer.

D'Agostino, M.; Finger, M.; and Gabbay, D. M. 2013. Semantics and proof-theory of depth bounded boolean logics. *Theor. Comput. Sci.* 480:43–68.

D'Agostino, M. 2015. An informational view of classical logic. *Theor. Comput. Sci.* 606:79–97.

D'Asaro, F. A.; Bikakis, A.; Dickens, L.; and Miller, R. 2017. Foundations for a probabilistic event calculus. In Balduccini, M., and Janhunen, T., eds., *Proceedings of the 14th International Conference Logic Programming and Nonmonotonic Reasoning, LPNMR 2017*, 57–63. Springer.

D'Asaro, F. A.; Bikakis, A.; Dickens, L.; and Miller, R. 2020. Probabilistic reasoning about epistemic action narratives. *Artificial Intelligence,* https://doi.org/10.1016/j.artint.2020.103352. 287:103352.

Gebser, M., and Schaub, T. 2006. Tableaux calculi for Answer Set Programming. *Proceedings of the 20th Workshop on Logic Programming, WLP 2006* 1–11.

Gebser, M.; Kaminski, R.; Kaufmann, B.; and Schaub, T. 2014. Clingo = ASP + control: Preliminary report. *CoRR* abs/1405.3694.

Quine, W. V. 1976. The roots of reference. *British Journal for the Philosophy of Science* 27(1):93–96.