Annals of Mathematics and Artificial Intelligence (2022) 90:271–296
https://doi.org/10.1007/s10472-021-09730-w

Parameterised complexity of model checking and satisfiability in propositional dependence logic

Yasir Mahmood[1] · Arne Meier[1]

Accepted: 1 February 2021 / Published online: 27 February 2021

Abstract
Dependence Logic was introduced by Jouko Väänänen in 2007. We study a propositional variant of this logic ($\mathcal{PDL}$) and investigate a variety of parameterisations with respect to central decision problems. The model checking problem (MC) of $\mathcal{PDL}$ is **NP**-complete (Ebbing and Lohmann, SOFSEM 2012). The subject of this research is to identify a list of parameterisations (formula-size, formula-depth, treewidth, team-size, number of variables) under which MC becomes fixed-parameter tractable. Furthermore, we show that the number of disjunctions or the arity of dependence atoms (dep-arity) as a parameter both yield a paraNP-completeness result. Then, we consider the satisfiability problem (SAT) which classically is known to be **NP**-complete as well (Lohmann and Vollmer, Studia Logica 2013). There we are presenting a different picture: under team-size, or dep-arity SAT is **paraNP**-complete whereas under all other mentioned parameters the problem is **FPT**. Finally, we introduce a variant of the satisfiability problem, asking for a team of a given size, and show for this problem an almost complete picture.

Keywords Propositional dependence logic · Parameterised complexity · Model checking · Satisfiability

Mathematics Subject Classification (2010) 68Q25 · 03B70

1 Introduction

The logics of dependence and independence are a recent innovation studying the notion of dependencies occurring in several areas of research: computer science, logic, statistics, game theory, linguistics, philosophy, biology, physics, and social choice theory [27]. Jouko Väänänen [47] initiated this subfield of research in 2007, and nowadays, it is a vibrant area

✉ Yasir Mahmood
mahmood@thi.uni-hannover.de

Arne Meier
meier@thi.uni-hannover.de

1 Institut für Theoretische Informatik, Leibniz Universität Hannover, Hannover, Germany

of study [1]. Its focus widened from initially first-order dependence logic further to modal logic [48], temporal logics [33, 34], probabilistic logics [15], logics for independence [32], inclusion logics [24, 31], multi-team semantics [14], and poly-team semantics [29].

In this paper, we study a sub-logic of the modal variant which is called propositional dependence logic ($\mathcal{PDL}$) [30, 51]. The main concept also in this logic, the *dependence atom* $\mathsf{dep}(P; Q)$, intuitively states that values of variables $p \in P$ functionally determine values of variables $q \in Q$. As functional dependence only makes sense on sets of assignments, which Väänänen called *teams*, team semantics are the heart of the satisfaction relation $\models$ in this logic. Formally, a team T is a set of classical propositional assignments $t\colon \mathrm{VAR} \to \{0, 1\}$, and $T \models \mathsf{dep}(P; Q)$ if and only if for all $t, t' \in T$, we have that t and t' agree on variables in P implies t and t' agree on variables in Q.

The model checking question (MC), given a team T and a $\mathcal{PDL}$-formula φ, asks if $T \models \varphi$ is true. The satisfiability problem (SAT), given a $\mathcal{PDL}$-formula φ, asks for the existence of a team T such that $T \models \varphi$. It is known that MC as well as SAT are **NP**-complete by Ebbing and Lohmann [18], respectively, by Lohmann and Vollmer [37]. These authors classify the complexity landscape of even operator-fragments of $\mathcal{PDL}$ yielding a deep understanding of these problems from a classical complexity point of view. For an overview of how other atoms (e.g., inclusion, or independence) influence the complexity of these problems consider the tables in the work Hella et al. [31].

Example 1 We illustrate an example from relational databases providing understanding of team logics.

Table 1 depicts a database which can be expressed in $\mathcal{PDL}$ via binary encoding of the possible entries for the attributes. The set of rows then corresponds to a team T. The database satisfies two functional dependencies:

$$\mathsf{dep}(\{\texttt{Room}, \texttt{Time}\}; \{\texttt{Course}\}) \text{ and } \mathsf{dep}(\{\texttt{Instructor}, \texttt{Time}\}; \{\texttt{Room}, \texttt{Course}\}).$$

Table 1 (Up) An example database with 4 attributes and universe size 15

`Instructor`	`Room`	`Time`	`Course`
Antti	A.10	09.00	Logic
Antti	A.10	11.00	Statistics
Antti	B.20	15.00	Algebra
Jonni	C.30	10.00	LAB
Juha	C.30	10.00	LAB
Juha	A.10	13.00	Statistics
$i_1 i_2$	$r_1 r_2$	$t_1 t_2 t_3$	$c_1 c_2$
00	11	110	11
00	11	111	00
00	00	000	01
01	01	001	10
10	01	001	10
10	11	010	00

(Bottom) An encoding with $\lceil \log_2(3) \rceil + \lceil \log_2(3) \rceil + \lceil \log_2(5) \rceil + \lceil \log_2(4) \rceil$ many propositional variables

Whereas, it does not satisfy $\mathsf{dep}(\{\mathtt{Room}, \mathtt{Time}\}; \{\mathtt{Instructor}\})$ as witnessed by the tuples (Juha, C.30, 10, LAB) and (Jonni, C.30, 10, LAB). Formally, we have that

$$T \models \mathsf{dep}(\{\mathtt{Room}, \mathtt{Time}\}; \{\mathtt{Course}\}) \wedge \mathsf{dep}(\{\mathtt{Instructor}, \mathtt{Time}\}; \{\mathtt{Room}, \mathtt{Course}\}),$$

but

$$T \not\models \mathsf{dep}(\{\mathtt{Room}, \mathtt{Time}\}; \{\mathtt{Instructor}\}).$$

Notice that in propositional logic, we cannot express a table of so many values. As a result, we need to encode in binary the values of each column separately. This might cause a logarithmic blow-up (by binary encoding the universe values for each column) in the parameter values, for example, it influences the number of variables. Furthermore, one also has to rewrite variables in the occurring formulas accordingly. For instance, as in Table 1, for $\mathsf{dep}(\{\mathtt{Room}, \mathtt{Time}\}; \{\mathtt{Instructor}\})$ this would yield the formula $\mathsf{dep}(\{r_1, r_2, t_1, t_2, t_3\}; \{i_1, i_2\})$. The parameters discussed in this paper correspond to the already encoded values. This means that there is no need to consider the blow-up observed in the previous example.

Often, when a problem is shown to be intrinsically hard, the framework of parameterised complexity theory [13] provides a way to further unravel the true reasons for the intractability. Here, one aims for a more fine-grained complexity analysis involving the study of parameterisations and how they pin causes for intractability substantially. One distinguishes two runtimes of a different quality: $f(k) \cdot p(|x|)$ versus $p(|x|)^{f(k)}$, where f is an arbitrary computable function, p is a polynomial, $|x|$ the input length and k the value of the parameter. Clearly, both runtimes are polynomial in x for each fixed k but the first one is much better as the polynomial degree is independent of the parameter's value. Problems that can be solved with algorithms running in a time of the first kind are said to be fixed-parameter tractable (or **FPT**). Whereas, problems of category two are in the complexity class **XP**. It is known that $\mathbf{FPT} \subsetneq \mathbf{XP}$ [23]. Whenever runtimes of the form $f(k) \cdot p(|x|)$ are considered with respect to nondeterministic machines, one studies the complexity class $\mathbf{paraNP} \supseteq \mathbf{FPT}$. In between these two classes a presumably infinite **W**-hierarchy is contained: $\mathbf{FPT} \subseteq \mathbf{W[1]} \subseteq \mathbf{W[2]} \subseteq \cdots \subseteq \mathbf{paraNP}$. It is unknown whether any of these inclusions is strict. Showing **W[1]**-hardness of a problem intuitively corresponds to being intractable in the parameterised world.

The area of research of parameterised problems is tremendously growing and often provides new insights into the inherent difficulty of the studied problems [12]. However, the area of dependence logic is rather blank with respect to this direction of research, only Meier and Reinbold [41] investigated the (parameterised) enumeration complexity of a fragment of $\mathcal{PDL}$ recently. As a subject of this research, we want to initiate and to further push a study of the parameterised complexity of problems in these logics.

Applications The concept of a team in team semantics bears a close resemblance with the relations studied in relational database theory. Moreover, dependence atoms are analogous to functional dependencies in the context of database systems. The MC problem for dependence logic, for example, is analogous to determining whether a relation in the database satisfies a functional dependency.

The teams of $\mathcal{PDL}$ also relate to the information states of inquisitive logic [10]; a semantic framework for the study of the notion of meaning and information exchange among agents.

Contributions We study a wealth of parameters, also relevant from the perspective of database theory. Here, the database corresponds to a team and the query that one wishes to evaluate is a $\mathcal{PDL}$-formula. The parameter team-size corresponds to the number of entries in the database and #variables corresponds to the number of attributes. The parameter formula-tw denotes how much interleaving is present among the attributes in the query and dep-arity bounds the size of functional dependencies in the query. Furthermore, the parameter formula-team-tw bounds the interleaving between a query and the database, formula-size limits the size of the query, formula-depth restricts the nesting depth of the query, and #splits bounds the number of join-operations in the query. With respect to all parameters, we study MC and SAT. Furthermore, we introduce a satisfiability variant m-SAT, which has an additional unary input $m \in \mathbb{N}$, and asks for a satisfying team of size at least m.

In Table 2, we give an overview of our results. In this article, we prove dichotomies for MC and SAT: depending on the parameter the problem is either fixed-parameter tractable or **paraNP**-complete. Only the satisfiability variant under the parameters formula-tw and #splits resist a complete classification and are left for further research.

Related work The notion of treewidth is due to Robertson and Seymour [44]. The study of the complexity of bounded treewidth query evaluation is a vibrant area of research [6–9, 16, 28]. As stated earlier, the formulas of dependence logic correspond to the functional dependencies in the database context. Bläsius et al. [4] study the parameterised complexity of dependency detection. The problem is defined as, given a database T and a positive integer k whether there is a non-trivial functional dependency of size (dep-arity in our notion) at most k that is satisfied by T. These authors prove that this problem is **W[2]**-complete.

Prior work A preliminary version of this article appeared in the proceedings of the 11th International Symposium on Foundations of Information and Knowledge Systems [39]. The present paper provides a higher level of detail, in particular it includes full proofs for several theorems (Theorem 18, 19, 24, 25, 27) and more examples. The new material includes a new result on the problem m-SAT (Theorem 26), and also a more extended outlook in the conclusion.

Table 2 Complexity classification overview showing the results of the paper with pointers to theorems. All **paraNP**-results are completeness results. The question mark symbol means that the precise complexity is unknown

Parameter	MC	SAT	m-SAT
formula-tw	**paraNP**^{15}	**FPT**^{23}	?
formula-team-tw	**FPT**^{20}	see above	see above
team-size	**FPT**^{17}	**paraNP**^{22}	**paraNP**^{28}
formula-size	**FPT**^{19}	**FPT**^{25}	**FPT**^{27}
formula-depth	**FPT**^{19}	**FPT**^{25}	**FPT**^{27}
#variables	**FPT**^{19}	**FPT**^{25}	**FPT**^{27}
#splits	**paraNP**^{18}	**FPT**^{24}	?
dep-arity	**paraNP**^{18}	**paraNP**^{22}	**paraNP**^{28}

Organisation of the article At first, we introduce some required notions and definitions in (parameterised) complexity theory, dependence logic, and propositional logic. Then we study the parameterised complexity of the model checking problem. We proceed with the satisfiability problem and study a variant of it. Finally, we conclude and discuss open questions.

2 Preliminaries

In this paper, we assume familiarity with standard notions in complexity theory [43] such as the classes **NP** and **P**.

2.1 Parameterised complexity

We will recapitulate some relevant notions of parameterised complexity theory, now. For a broader introduction consider the textbook of Downey and Fellows [13], or that of Flum and Grohe [23]. A parameterised problem (PP) $\Pi \subseteq \Sigma^* \times \mathbb{N}$ consists of tuples (x, k), where x is called the *instance* and k the *(value of the) parameter*.

Definition 2 (Fixed-parameter tractable and **paraNP**) Let Π be a PP over $\Sigma^* \times \mathbb{N}$. We say that Π is *fixed-parameter tractable* (or is in the class **FPT**) if there exists a deterministic algorithm $\mathcal{A}$ deciding Π in time $f(k) \cdot |x|^{O(1)}$ for every input $(x, k) \in \Sigma^*$, where f is a computable function. If $\mathcal{A}$ is a nondeterministic algorithm instead, then Π belongs to the class **paraNP**.

Let P be a PP over $\Sigma^* \times \mathbb{N}$. Then the *$\ell$-slice of P*, for $\ell \geq 0$, is the set $P_\ell := \{x \mid (x, \ell) \in P\}$. It is customary to use the notation $O^\star(f(k))$ to denote the runtime dependence only on the parameter and to ignore the polynomial factor in the input. We will use the following result from parameterised complexity theory to prove **paraNP**-hardness results.

Proposition 3 [23, Theorem 2.14] *Let P be a PP. If there exists an $\ell \geq 0$ such that P_ℓ is* **NP**-*complete, then P is* **paraNP**-complete.

Moreover, we will use the following folklore result to get several upper bounds.

Proposition 4 *Let Q be a problem such that (Q, k) is in* **FPT** *and let ℓ be another parameter such that $k \leq f(\ell)$ for some computable function f, then (Q, ℓ) is also in* **FPT**.

2.2 Propositional dependence logic

Let VAR be a countably infinite set of variables. The syntax of propositional dependence logic ($\mathcal{PDL}$) is defined via the following EBNF:

$$\varphi ::= \top \mid \bot \mid x \mid \neg x \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \mathsf{dep}(X; Y) \mid \neg\mathsf{dep}(X; Y),$$

where $\top$ is *verum*, $\bot$ is *falsum*, $x \in$ VAR is a variable, $X, Y \subset$ VAR are finite sets of variables, $\mathsf{dep}(\cdot;\cdot)$ is called the *dependence atom*, and the disjunction $\vee$ is also called *split-junction*. Observe that we only consider atomic negation. We let $\mathcal{PL}$ be defined as the $\mathcal{PDL}$-formulas without $\mathsf{dep}(\cdot;\cdot)$. Finally, the set X in $\mathsf{dep}(X; Y)$ can be empty, giving rise to formulas of the form $\mathsf{dep}(; Y)$. To simplify the notation, when either set in the arguments of $\mathsf{dep}(X; Y)$ is singleton then we write, for example, $\mathsf{dep}(x; y)$ instead of $\mathsf{dep}(\{x\}; \{y\})$.

Definition 5 (Team semantics) Let φ, ψ be $\mathcal{PDL}$-formulas and $P, Q \subset \mathrm{VAR}$ be two finite sets of variables. A *team* T is a set of assignments $t\colon \mathrm{VAR} \to \{0,1\}$. Furthermore, we define the satisfaction relation $\models$ as follows, where $T \models \top$ is always true, $T \models \bot$ is never true, and $T \models \neg\mathsf{dep}(P; Q)$ iff $T = \emptyset$:

$$\begin{array}{lll}
T \models x & \text{iff} & \forall t \in T : t(x) = 1 \\
T \models \neg x & \text{iff} & \forall t \in T : t(x) = 0 \\
T \models \varphi \wedge \psi & \text{iff} & T \models \varphi \text{ and } T \models \psi \\
T \models \varphi \vee \psi & \text{iff} & \exists T_1 \exists T_2 (T_1 \cup T_2 = T) : T_1 \models \varphi \text{ and } T_2 \models \psi \\
T \models \mathsf{dep}(P; Q) & \text{iff} & \forall t, t' \in T : \bigwedge\limits_{p \in P} t(p) = t'(p) \text{ implies } \bigwedge\limits_{q \in Q} t(q) = t'(q)
\end{array}$$

Observe that for the satisfaction of formulas of the form $\mathsf{dep}(; Q)$ the team has to be constant with respect to Q. That is why such atoms are called *constancy atoms*. Note that in literature there exist two semantics for the split-junction operator: *lax* and *strict* semantics (e.g., Hella et al. [31]). Strict semantics requires the "splitting of the team" to be a partition whereas lax semantics allow an "overlapping" of the team. We use lax semantics here. Notice that the computational complexity for SAT and MC in $\mathcal{PDL}$ are the same irrespective of the considered semantics. Furthermore, our proofs work for both semantics. Also note that allowing an unrestricted negation operator dramatically increases the complexity of SAT in this logic to **ATIME-ALT(exp, poly)** (alternating exponential time with polynomially many alternations) as shown by Hannula et al. [30]. That is one reason why we stick to atomic negation.

In the following, we define three well-known formula properties which are relevant to results in the paper. A formula ϕ is *flat* if, given any team T, we have that $T \models \phi \iff \{s\} \models \phi$ for every $s \in T$. A logic $\mathcal{L}$ is *downwards closed* if for every $\mathcal{L}$-formula ϕ and team T, if $T \models \phi$ then for every $P \subseteq T$ we have that $P \models \phi$. A formula ϕ is *2-coherent* if for every team T, we have that $T \models \phi \iff \{s_i, s_j\} \models \phi$ for every $s_i, s_j \in T$. The classical $\mathcal{PL}$-formulas are flat. This also implies that for $\mathcal{PL}$-formulas, the truth value is evaluated under each assignment individually, consequently, the semantics is the usual Tarski semantic. Moreover, $\mathcal{PDL}$ is downwards closed and every dependence atom is 2-coherent.

2.3 Representation of inputs as graphs

As we will consider specific structural parameters, we need to agree on a representation of formulas, respectively, teams. Classically, propositional formulas were represented via different kinds of graphs (e.g., Gaifman graph, primal graph) [45]. However, in this setting usually CNF-formulas are considered. Coping with this restriction, Lück et al. [38] defined syntax circuits for temporal logic formulas that also allow arbitrary formulas. In our setting, we continue in this direction and define the syntax (or formula) structure with respect to a $\mathcal{PDL}$-formula.

An important observation regarding the graph representation for the $\mathcal{PDL}$-formulas is due to Grädel [26]. In the usual setting for logics with team semantics, we take the syntax tree and not the associated syntax structure, that is, we distinguish between different occurrences of the same subformula. The reason for this choice is that a formula $\phi \vee \phi$ is not equivalent to ϕ, and in its evaluation, different teams are entitled to the two occurrences of

ϕ in the formula. Consequently, the well-formed formulas of $\mathcal{PDL}$ can be seen as binary trees with leaves as atomic subformulas (variables and dependence atoms).

Example 6 The team {00, 01, 10, 11} satisfies $\mathsf{dep}(x; y) \vee \mathsf{dep}(x; y)$, even though it does not satisfy $\mathsf{dep}(x; y)$.

Notice that when a $\mathcal{PDL}$-formula is considered a tree, as discussed above, the parameter treewidth (Def. 9) is not meaningful anymore.

For this reason we consider the syntax structure rather than the syntax tree as a graph structure to consider treewidth as a parameter. Moreover, in the case of MC, one might include assignments in a graph representation. In the latter case, one considers the Gaifman graph of the structure that models both, the team and the input formula.

Definition 7 (Syntax structure) Let $\langle T, \Phi \rangle$ be an instance of the model checking problem, where Φ is a $\mathcal{PDL}$-formula with propositional variables $\{ x_1, \ldots, x_n \} \subseteq \mathrm{VAR}$ and $T = \{ s_1, \ldots s_m \}$ is a team of assignments $s_i : \mathrm{VAR} \to \{ 0, 1 \}$. The *syntax structure* $\mathcal{A}_{T,\Phi}$ over the vocabulary

$$\tau_{T,\Phi} := \{ \mathsf{VAR}^1, \mathsf{SF}^1, \succcurlyeq^2, \mathsf{DEP}^2, \mathsf{inTeam}^1, \mathsf{isTrue}^2, \mathsf{isFalse}^2, r, c_1, \ldots, c_m \},$$

where superscripts denote the arity of each relation, then is defined as follows.

The universe of $\mathcal{A}_{T,\Phi}$ is $A := \mathrm{SF}(\Phi) \cup \mathrm{VAR}(\Phi) \cup \{ c_1^{\mathcal{A}}, \ldots, c_m^{\mathcal{A}} \}$, where $\mathrm{SF}(\Phi)$ and $\mathrm{VAR}(\Phi)$ denote the set of subformulas and variables appearing in Φ, respectively.

– SF and VAR are unary relations representing 'is a subformula of Φ' and 'is a variable in Φ' respectively.
– $\succcurlyeq$ is a binary relation such that $\phi \succcurlyeq^{\mathcal{A}} \psi$ iff ψ is an immediate subformula of ϕ and r is a constant symbol representing Φ.
– DEP is a binary relation which connects each dependence atom with the used variables.
– The set $\{ c_1, \ldots, c_m \}$ encodes the team T, where each c_i is interpreted as $c_i^{\mathcal{A}} \in \mathcal{A}$ and each c_i corresponds to an assignment $s_i \in T$ for $i \leq m$.
– $\mathsf{inTeam}(c)$ is true if and only if $c \in \{ c_1, \ldots, c_m \}$.
– isTrue and $\mathsf{isFalse}$ relate variables with the team elements. $\mathsf{isTrue}(c, x)$ (resp., $\mathsf{isFalse}(c, x)$) is true if and only if x is mapped 1 (resp., 0) by the assignment interpreted by c.

Analogously, the *syntax structure* $\mathcal{A}_\Phi$ over a respective vocabulary τ_Φ is defined. In this case, the team related relations are not present and the universe does not contain the m-many constants $c_i^{\mathcal{A}}$ for $1 \leq i \leq m$.

In the following we write $\mathcal{A}$ instead of $\mathcal{A}_{T,\Phi}$ when it is clear that our input instance is $\langle T, \Phi \rangle$.

Definition 8 (Gaifman graph) Given a team T and a $\mathcal{PDL}$-formula Φ, the *Gaifman graph* $G_{T,\Phi} = (A, E)$ of the $\tau_{T,\Phi}$-structure $\mathcal{A}_{T,\Phi}$ is defined as

$$E := \Big\{ \{u, v\} \;\Big|\; u, v \in A, \text{ such that there is an } R \in \tau_{T,\Phi} \text{ with } (u, v) \in R \Big\}.$$

Analogously, we let G_Φ to be the *Gaifman graph* for the τ_Φ-structure $\mathcal{A}_\Phi$.

Note that for G_Φ we have $E = \mathsf{DEP} \cup \succcurlyeq$ and for $G_{T,\Phi}$ we have that $E = \mathsf{DEP} \cup \succcurlyeq \cup\, \mathsf{isTrue} \cup \mathsf{isfalse}$.

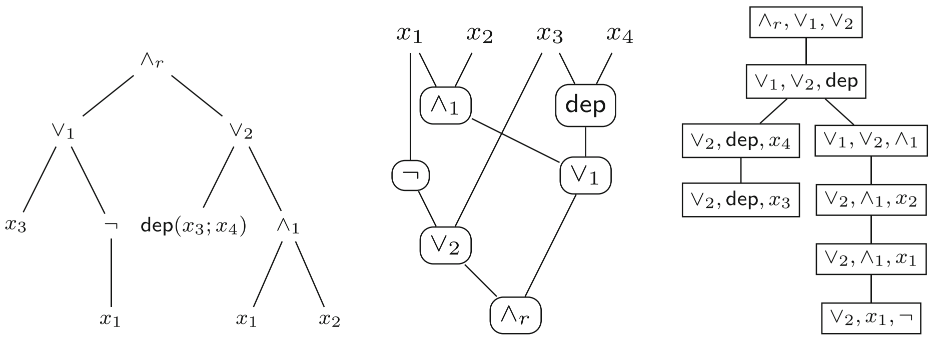


Fig. 1 An example syntax tree (left) with the corresponding Gaifman graph (middle) and a tree decomposition (right) for $(x_3 \vee \neg x_1) \wedge \big(\mathsf{dep}(x_3; x_4) \vee (x_1 \wedge x_2)\big)$. Note that we abbreviated subformulas in the inner vertices of the Gaifman graph for presentation reasons

Definition 9 (Treewidth) The *tree decomposition* of a given graph $G = (V, E)$ is a tree $T = (B, E_T)$, where the vertex set $B \subseteq \mathcal{P}(V)$ is called *bags* and E_T is the edge relation such that the following is true.

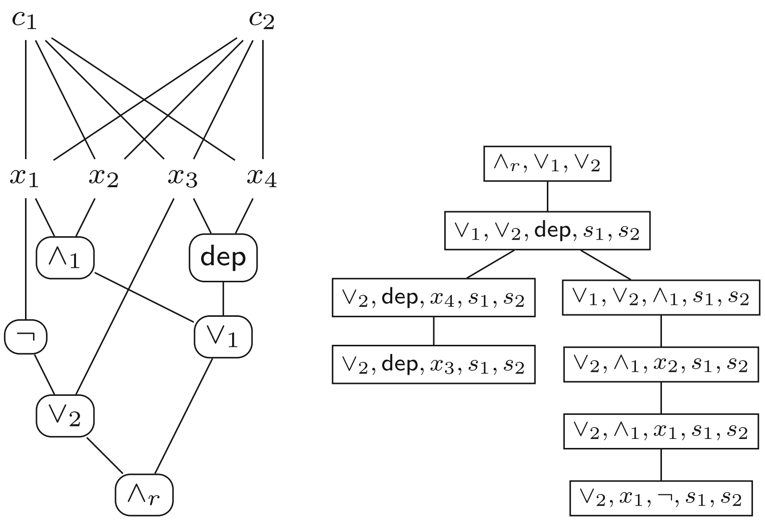
- $\bigcup_{b \in B} = V$,
- for every $\{u, v\} \in E$ there is a bag $b \in B$ with $u, v \in b$, and
- for all $v \in V$ the restriction of T to v (the subset with all bags containing v) is connected.

The *width* of a given tree decomposition $T = (B, E_T)$ is the size of the largest bag minus one: $\max_{b \in B} |b| - 1$. The *treewidth* of a given graph G is the minimum over all widths of tree decompositions of G.

Observe that if G is a tree then the treewidth of G is one. Intuitively, one can say that treewidth accordingly is a measure of tree-likeness of a given graph. The decision problem to determine whether the treewidth of a given graph $\mathcal{G} = (V, E)$ is at most k, is **NP**-complete [2]. See Bodlaender's Guide [5] for an overview of algorithms that compute tree decompositions. Further note that, opposed to our parameterised setting here, where instances of PPs are pairs of strings and numbers (the latter being the value of the parameter), there exists an approach, in which one works with parameterisation functions that map the input string to the value of the parameter (e.g., Flum and Grohe [23]). Clearly, in such a setting it could be tempting to think that treewidth is not a reasonable parameter because of the mentioned hardness of computing it. However, one can circumvent this pitfall by letting the input string contain the treewidth. The parameterisation function then essentially maps to a substring of the input avoiding the need to compute it from scratch.

Example 10 Figure 1 represents the Gaifman graph of the syntax structure $\mathcal{A}_\Phi$ (in middle) with a tree decomposition (on the right). Since the largest bag is of size 3, the treewidth of the given decomposition is 2. Figure 2 presents the Gaifman graph of the syntax structure $\mathcal{A}_{T,\Phi}$, that is, when the team $T = \{s_1, s_2\} = \{0011, 1110\}$ is also part of the input.

In Lemma 12, we will prove that the treewidth increases when the team is also part of the input.



c_1
c_2
x_1
x_2
x_3
x_4
$\wedge_1$
dep
$\neg$
$\vee_1$
$\vee_2$
$\wedge_r$
$\wedge_r, \vee_1, \vee_2$
$\vee_1, \vee_2$, dep, s_1, s_2
$\vee_2$, dep, x_4, s_1, s_2
$\vee_1, \vee_2, \wedge_1, s_1, s_2$
$\vee_2$, dep, x_3, s_1, s_2
$\vee_2, \wedge_1, x_2, s_1, s_2$
$\vee_2, \wedge_1, x_1, s_1, s_2$
$\vee_2, x_1, \neg, s_1, s_2$

Fig. 2 The Gaifman graph for $\langle T, \Phi\rangle$ (as given in Example 10) with a possible tree decomposition. Note that we abbreviated the subformulas in the inner vertices of the Gaifman graph for presentation reasons

2.4 Considered parameterisations

We consider eight different parameters for all three problems of interest (MC, SAT and m-SAT). These include formula-tw, formula-team-tw, team-size, formula-size, #variables, formula-depth, #splits and dep-arity. All these parameters arise naturally in problems we study. Let T be a team and Φ a $\mathcal{PDL}$-formula then the parameters are defined as follows:

– #splits denotes the number of times a split-junction ($\vee$) appears in Φ and #variables denotes the total number of propositional variables.
– formula-depth is the depth of the syntax tree of Φ, that is, the length of the longest path from root to any leaf in the tree.
– team-size is the size of the team T.
– The arity of a dependence atom dep$(P; Q)$ is the size of P and dep-arity is the maximum arity of a dependence atom in Φ.

Regarding treewidth, recall that for the MC problem, we can also include the assignment-variable relation in the graph representation. This yields two treewidth notions formula-tw and formula-team-tw, the name emphasises whether the team is also part of the graph.

– formula-tw is the treewidth of G_Φ.
– formula-team-tw is the treewidth of $G_{T,\Phi}$.

Clearly, formula-team-tw is only relevant for the MC problem.

The following lemma proves relationships between several of the aforementioned parameters. The notation $\kappa(T, \Phi)$ stands for the parameter value of the input instance (T, Φ). This is also visualised in Fig. 3.

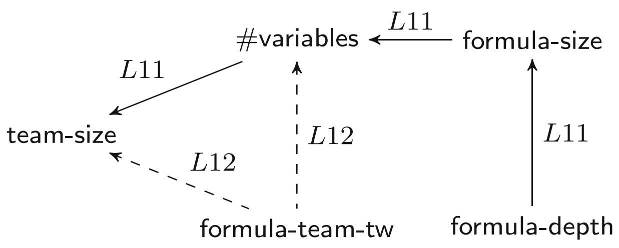


Fig. 3 The relationship among different parameters. The direction of arrow in $p \leftarrow q$ implies that bounding q results in bounding p. The dashed line indicates that the parameter bounds either (minimum) of the given two. Li means Lemma i

Lemma 11 *Given a team T and a formula Φ then*

1. $\textsf{team-size}(T, \Phi) \leq 2^{\textsf{\#variables}(T,\Phi)}$
2. $\textsf{team-size}(T, \Phi) \leq 2^{\textsf{formula-size}(T,\Phi)}$
3. $\textsf{formula-size}(T, \Phi) \leq 2^{2\cdot\textsf{formula-depth}(T,\Phi)}$

Proof If a $\mathcal{PDL}$-formula Φ has m variables then there are 2^m many assignments and the maximum size for a team is 2^m. As a result, we have $\textsf{team-size} \leq 2^{\textsf{\#variables}}$. Furthermore, the number of variables in a $\mathcal{PDL}$-formula Φ is bounded by the formula-size and as a consequence, we have $2^{\textsf{\#variables}} \leq 2^{\textsf{formula-size}}$. This proves the second claim.

If a formula Φ has $\textsf{formula-depth} = d$ then there are $\leq 2^d$ leaves in the (binary) syntax tree of Φ and $\leq 2^d$ internal nodes. Then $\textsf{formula-size} \leq 2^{2d}$ is true. □

Now we prove the following non-trivial lemma stating that treewidth of the structure $\mathcal{A}_{T,\Phi}$ bounds either the team size or the number of variables. This implies that bounding the treewidth of the structure also bounds either of the two parameters. Recall that for formula-team-tw, we talk about the treewidth of the Gaifman graph underlying the structure $\mathcal{A}_{T,\Phi}$ that encodes the MC question.

Lemma 12 *Let $\langle T, \Phi\rangle$ be a given* MC *instance. Then the following relationship between parameters is true,*

$$\textsf{formula-team-tw}(T, \Phi) \geq \min\{\,\textsf{team-size}(T, \Phi), \textsf{\#variables}(T, \Phi)\,\}$$

Proof We prove that if formula-team-tw is smaller than the two then such a decomposition must have cycles and hence cannot be a tree decomposition. The proof uses the fact that in the Gaifman graph $G_{T,\Phi}$, every team element is related to each variable. As a consequence, in any tree decomposition, the assignment-variable relations 'isTrue' and 'isFalse' cause some bag to have their size larger than either the team size or the number of variables (based on which of the two values is smaller). We consider individual bags corresponding to an edge in the Gaifman graph due to the relations from $\tau_{T,\Phi}$. Let $\{\, x_1, \dots, x_n \,\}$ denote variables that also appear as leaves in the formula tree $\langle \mathrm{SF}(\Phi), \succcurlyeq, \Phi\rangle$.

Consider a minimal tree decomposition $\langle \mathcal{B}_T, \prec\rangle$ for the Gaifman graph of $\mathcal{A}$. Denote by $B(x_i, c_j)$ the bag that covers the edge between a variable x_i and an assignment-element c_j, that is, either $\textsf{isTrue}(x_i, c_j)$ or $\textsf{isFalse}(x_i, c_j)$ is true. Moreover, denote by $B(x_i, \alpha)$ the bag covering the edge between a variable x_i and its immediate $\succcurlyeq$-predecessor α. Recall that in the formula part of the Gaifman graph, there is a path from each variable x_i to the formula Φ due to $\succcurlyeq$. This implies that there exists a minimal path between any pair of variables in the Gaifman graph, and this path passes through some subformula Ψ of Φ. Let $B(x, \alpha_1)$, $B(\alpha_1, \alpha_2), \ldots, B(\alpha_q, \Psi)$, $B(\Psi, \beta_r), \ldots, B(\beta_2, \beta_1)$, $B(\beta_1, y)$ be the sequence of bags that covers $\succcurlyeq^{\mathcal{A}}$-edges between x and y (where $q, r \leq |\mathrm{SF}(\Phi)|$). Without loss of generality, we assume that all these bags are distinct. Now, for any pair x, y of variables, the bags $B(x, c_i)$ and $B(y, c_i)$ contain c_i for each $i \leq m$ and as a consequence, we have either of the following two cases.

Case 1. The two bags are equal, that is $B(x, c_i) = B(y, c_i)$ and as a consequence, we have $|B(x, c_i)| \geq 3$ because $B(x, c_i)$ contains at least x, y and c_i. Moreover, if this is true (otherwise case two applies) for each pair of variables, then there is a single bag, say $B(c_i)$, that contains all variables and the element c_i. This means the maximum bag size must be larger than the total number of variables, a contradiction.

Case 2. Every bag in the path between $B(x, c_i)$ and $B(y, c_i)$ contains c_i.

We know that if a $B(x, \alpha_1)$-$B(\beta_1, y)$-path between x and y due to $\succcurlyeq$ exist, then the bags $B(x, c_i)$ and $B(y, c_i)$ cannot be adjacent because this will produce a cycle, a contradiction again.

Moreover, for two different assignment-elements c_i, c_j, consider the bags $B(y, c_i)$ and $B(y, c_j)$. If these two bags are adjacent then $B(x, c_j)$ and $B(y, c_j)$ cannot be adjacent and the path between $B(x, c_j)$ and $B(y, c_j)$ must contain c_j. Notice that both $B(y, c_i)$, $B(y, c_j)$ and $B(x, c_j)$, $B(y, c_j)$ cannot be adjacent since this would, again, create a cycle. Consequently, the two possible cases are (see Fig. 4 explaining this situation as well): first, $B(y, c_i)$ and $B(y, c_j)$ are not adjacent and every path between these bags contains y. Second, $B(x, c_j)$ and $B(y, c_j)$ are not adjacent and every path between these bags contains c_j. Finally, since this is true for all variables and all elements c_i with $i \leq m$ this proves that either there is a bag that contains all variables, or there is one that contains all c_i's. The remaining case that there are cycles in the tree decomposition is not applicable.

This proves the claim and completes the proof to the lemma. □

The following corollary is immediate due to previous lemma.

Corollary 13 *Let $\Phi \in \mathcal{PDL}$ and T be a team. Then* formula-team-tw(T, Φ) *bounds* team-size(T, Φ).

Proof If formula-team-tw $\geq$ #variables then bounding formula-team-tw bounds #variables which in turn bounds team-size because team-size $\leq 2^{\#\text{variables}}$. Otherwise we already have formula-team-tw $\geq$ team-size according to Lemma 12. □

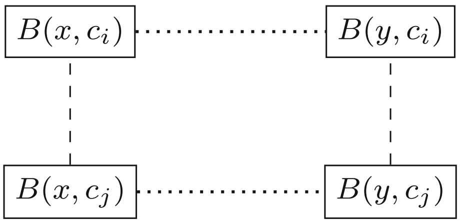


Fig. 4 The rectangles represent bags corresponding to a variable-assignment relation. If the c_i-bags do not contain c_j-nodes, then there can be only either dotted or dashed edges between the bags to avoid cycles

3 Parameterised complexity of model checking in PDL

In this section, we study the model checking question under various parameterisations. Classically, the problem is **NP**-complete (Prop. 14). Table 2 contains a complete list of our results.

Proposition 14 [18, Thm. 3.2] MC *is* **NP**-*complete.*

Theorem 15 MC *parameterised by* formula-tw *is* **paraNP**-*complete.*

Proof The upper bounds follows from Proposition 14. For the lower bound, we prove that the 1-slice of the problem is **NP**-hard by reducing from 3SAT. The reduction provided by Ebbing and Lohmann (Prop. 14) uses Kripke semantics (as they aim for a modal logic related results). We slightly modify it to fit our presentation (the correctness proof is the same). Let $\Phi := C_1 \wedge \ldots \wedge C_m$ be an instance of 3SAT over $\{x_1, \ldots, x_n\}$ and each C_i be a clause, that is, a set of literals (variables or negation of variables). We define an instance $\langle T, \Psi\rangle$ of $\mathcal{PDL}$-MC such that $\mathrm{VAR}(\Psi) = \{p_1, \ldots, p_n, r_1 \ldots, r_n\}$. The team $T = \{s_1, \ldots, s_m\}$ contains m assignments, where each assignment $s_i \colon \mathrm{VAR}(\Psi) \to \{0, 1\}$ is defined as follows,

$$\begin{aligned} &s_i(p_j) = s_i(r_j) = 1, && \text{if } x_j \in C_i,\\ &s_i(p_j) = 0, s_i(r_j) = 1, && \text{if } \neg x_j \in C_i,\\ &s_i(p_j) = s_i(r_j) = 0, && \text{if } x_j, \neg x_j \notin C_i. \end{aligned}$$

That is, there is an assignment s_i per each clause. The variable r_j encodes whether or not x_j appears in the clause C_i, whereas, p_j encodes whether x_j appears positively or negatively in C_i. Finally, let $\Psi := \bigvee_{j=1}^{n} \big(r_j \wedge \mathsf{dep}(; \{p_j\})\big)$. The proof of $\Phi \in \mathrm{SAT} \iff T \models \Psi$ is similar to the one of Ebbing and Lohmann [18, Thm. 3.2].

"$\Rightarrow$". Let θ be a satisfying assignment for Φ. We construct T_j for each $j \leq n$ such that,

$$T_j := \begin{cases} \{s_i \mid s_i(p_j) = 1 = s_i(r_j)\} & \text{if } \theta(x_j) = 1,\\ \{s_i \mid s_i(p_j) = 0, s_i(r_j) = 1\} & \text{if } \theta(x_j) = 0. \end{cases}$$

That is, T_j contains an assignment s_i if and only if the clauses C_i is satisfied by $\theta(x_j)$ where $i \leq m$. Clearly, $T_j \models r_j \wedge \mathsf{dep}(; p_j)$. Moreover, since every clause is satisfied, this implies that $\bigcup_{j \leq n} T_j = T$ and consequently $T \models \Psi$.

"$\Leftarrow$". Suppose that $T \models \Psi$, then there are $T_1, \ldots, T_n$ such that $T = \bigcup_{j \leq n} T_j$ and $T_j \models r_j \wedge \mathsf{dep}(; p_j)$. Clearly p_j is fixed by each T_j. We construct a satisfying assignment for Φ by considering each variable in turn. For $j \leq n$, let $V_j = \{i \mid s_i \in T_j\}$. Now, $s_i(p_j) = 1$ implies $x_j \in C_i$, and we set $\theta(x_j) = 1$, otherwise set $s_i(p_j) = 0$. This implies that $\neg x_j \in C_i$ and we set $\theta(x_j) = 0$. Since for every $s_i \in T$ there is a $j \leq n$ with $s_i \in T_j$, we have an evaluation that satisfies every clause $C_i \in \Phi$ and, as a consequence, $\theta \models \Phi$.

Notice that the parameter formula-twis fixed in advance. This is because G_Ψ is a tree; as a consequence, formula-tw $= 1$. This completes the proof. □

Notice that the formula in the reduction from 3SAT has fixed arity for any dependence atom (that is, dep-arity $= 0$). As a consequence, we obtain the following corollary.

Corollary 16 MC *parameterised by* dep-arity *is* **paraNP**-*complete.*

Algorithm 1 Recursive bottom-up algorithm solving MC parameterised by team-size.

```
Input  : A $\mathcal{PDL}$-formula $\Phi$ and a team $T$
Output: true if $T \models \Phi$, otherwise false
foreach non-root node $v$ in the syntax tree do $L_v = \{\emptyset\}$
foreach atomic/negated atomic $\ell \in \mathrm{SF}(\Phi)$ do  // find all sub-teams for $\ell$
    $L_\ell = \{\emptyset\}$
    foreach $P \subseteq T$ do
        if $\ell = x$ and $\forall s \in P : s(x) = 1$ then  $L_\ell \leftarrow L_\ell \cup \{P\}$
        else if $\ell = \neg x$ and $\forall s \in P : s(x) = 0$ then  $L_\ell \leftarrow L_\ell \cup \{P\}$
        else if $\ell = \neg\mathrm{dep}(P; Q)$ then  $L_\ell \leftarrow L_\ell$    // because $\emptyset \models \neg\mathrm{dep}(P; Q)$
        else if $\ell = \mathrm{dep}(P; Q)$ and $\forall s_i \forall s_j \bigwedge_{p \in P} s_i(p) = s_j(p) \Rightarrow \bigwedge_{q \in Q} s_i(q) = s_j(q)$ then
            $L_\ell \leftarrow L_\ell \cup \{P\}$
foreach $\alpha_1, \alpha_2$ with $\alpha = \alpha_1 \circ \alpha_2$ and $L_{\alpha_i} \neq \{\emptyset\}$ for $i = 1, 2$ do
    foreach $P \in L_{\alpha_1}$, $Q \in L_{\alpha_2}$ do
        if $\circ = \wedge$ and $P = Q$ then  $L_\alpha \leftarrow L_\alpha \cup \{P\}$
        else if $\circ = \vee$ then  $L_\alpha \leftarrow L_\alpha \cup \{P \cup Q\}$
if $T \in L_\Phi$ then return true else return false
```

The main source of difficulty in the model checking problem seems to be the split-junction operator.

For a team of size k and a formula with only one split-junction there are 2^k many candidates for the correct split and each can be verified in polynomial time. As a result, an exponential runtime in the input length seems necessary. However, if the team size (k) is considered as a parameter then the problem can be solved in polynomial time with respect to the input size and exponentially in the parameter. We consider both parameters (team-sizeand #splits) in turn.

Theorem 17 MC *parameterised by* team-size *is* **FPT**.

Proof We claim that Algorithm 1 solves the task in fpt-time. The correctness follows from the fact that the procedure is simply a recursive definition of truth evaluation of $\mathcal{PDL}$-formulas in bottom-up fashion.

Recall that the input formula Φ is a binary tree. The procedure starts by checking whether for each atomic (or negated atomic) subformula α and each subteam $P \subseteq T$, $P \models \alpha$. Notice that this step also takes care of negated atomic suformulas because we only allow atomic negations. Then recursively, if $P \models \alpha_i$ for $i = 1, 2$ and there is a subformula α such that $\alpha = \alpha_1 \wedge \alpha_2$ then it answers that $P \models \alpha$. Moreover, if $P_i \models \alpha_i$ for $i = 1, 2$ and there is a subformula α such that $\alpha = \alpha_1 \vee \alpha_2$ then it answers that $P \models \alpha$ where $P = P_1 \cup P_2$.

The first loop runs in $O^\star(2^k)$ steps for each leaf node and there are $|\Phi|$ many iterations, which gives a running time of $|\Phi| \cdot O^\star(2^k)$, where team-size $= k$. At each inner node, there are at most 2^k candidates for P and Q and as a consequence, at most 2^{2k} pairs that need to be checked. This implies that the loop for each inner node can be implemented in $O^\star(2^{2k})$ steps. Furthermore, the loop runs once for each pair of subformulas α_1, α_2 such that $\alpha_1 \circ \alpha_2$ is also a subformula of Φ. This gives a running time of $|\Phi| \cdot O^\star(2^{2k})$ for this step. Finally, in the last step a set of size k needs to be checked against a collection containing 2^k such sets, this can be done in $k \cdot O(2^k)$ steps.

We conclude that the above procedure solves the MC problem in $p(|\Phi|) \cdot O(2^{2k})$ steps for some polynomial p. The fact that we do not get a blow-up in the number of subformulas is due to the reason that the formula tree is binary. The procedure operates on a pair of subformulas in each step and the label size ($|L_\alpha|$) at the end of this step is again bounded by 2^k. $\square$

Regarding the parameter #splits, we show **paraNP**-completeness by reducing from the 3-colouring problem (3COL) and applying Proposition 3.

Theorem 18 MC *parameterised by* #splits *is* **paraNP**-*complete.*

Proof Given an instance $\langle \mathcal{G} \rangle$ where $\mathcal{G} = (V, E)$ is a graph. We map this input to an instance $\langle (T, \Phi), 2 \rangle$ where T is a team, and Φ is a $\mathcal{PDL}$-formula with 2 split-junctions. The idea of the reduction from 3COL is to construct a team as shown in Fig. 5 in combination with the formula containing two disjunctions, where each disjunct is $\bigwedge_{e_k = \{ v_i, v_j \}} \mathsf{dep}(\{y_k\}; \{x_i\})$. Intuitively, vertices of the graph correspond to assignments in the team and the three splits then map to three colours.

Let $V = \{ v_1, \ldots, v_n \}$ be the vertex set and $E = \{ e_1, \ldots e_m \}$ the given set of edges. Then we define

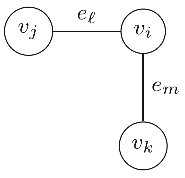
$$\mathrm{VAR}(\Phi) = \{ x_1, \ldots, x_n \} \cup \{ y_{1,1}, \ldots, y_{1,n}, \ldots, y_{m,1}, \ldots, y_{m,n} \}.$$

That is, we have (1) a variable x_i corresponding to each node v_i and (2) a variable $y_{j,k}$ corresponding to each edge e_j and each node v_k. For convenience, we will sometimes write y_j instead of $(y_{j,1} \ldots y_{j,n})$ when it is clear that we are talking about the tuple of variables corresponding to the edge e_j. Consequently, we have an n-tuple of variables y_j for each edge e_j, where $1 \leq j \leq m$. The idea of the team that we construct is that there is an assignment s_i corresponding to each node v_i that encodes the neighbourhood of v_i in the graph. The assignment s_i also encodes all the edges that v_i participates in. This is achieved by mapping each variable $y_{\ell,j}$ in tuple y_ℓ to 1 under the assignment s_j if $v_j \in e_\ell$ whereas $s_j(y_{\ell,j}) = 0$ if $v_j \notin e_\ell$ and for every $j \neq i$, $s_j(y_{\ell,j}) = 1$. Figure 5 illustrates an example to get an intuition on this construction.

Formally, we define the team as follows.

1. If $\mathcal{G}$ has an edge $e_\ell = \{ v_i, v_j \}$ then we set $s_i(x_j) = 1$ and $s_j(x_i) = 1$, and let $s_i(y_{\ell,1}) = \ldots = s_i(y_{\ell,n}) = 1$ as well as $s_j(y_{\ell,1}) = \ldots = s_j(y_{\ell,n}) = 1$
2. For the case $v_j \notin e_\ell$, we set $s_j(y_{\ell,j}) = 0$ and for the remaining indices $s_j(y_{\ell,i}) = 1$.
3. Since, we can assume w.l.o.g. the graph has no loops (self-edges) we always have $s_i(x_i) = 0$ for all $1 \leq i \leq n$.

Consequently, two assignments s_i, s_j agree on y_k if the corresponding edge e_k is the edge between v_i and v_j, and we have $s_i(y_k) = 1 = s_j(y_k)$.



	x_i	x_j	x_k	$y_{\ell,i}$	$y_{\ell,j}$	$y_{\ell,k}$	$y_{m,i}$	$y_{m,j}$	$y_{m,k}$
s_i	0	1	1	1	1	1	1	1	1
s_j	1	0	0	1	1	1	1	0	1
s_k	1	0	0	1	1	0	1	1	1

Fig. 5 A graph $\mathcal{G} : \langle \{ v_i, v_j, v_k \}, \{ e_l, e_m \} \rangle$ and a corresponding team

Now let Φ be the following formula

$$\Phi := \phi \vee \phi \vee \phi$$

where

$$\phi = \bigwedge_{e_k = \{ v_i, v_j \}} \mathsf{dep}(\{y_k\}; \{x_i\})$$

The choice of x_i or x_j to appear in the formula is irrelevant. The idea is that if there is an edge e_k between two nodes v_i, v_j and accordingly $s_i(y_k) = 1 = s_j(y_k)$ then the two assignments cannot be in the same split of the team. This is always true because in that case the assignments s_i, s_j cannot agree on any of x_i or x_j. Since, by (3.), we have $s_i(x_i) = 0$ but there is an edge to v_j and we have $s_j(x_i) = 1$. The desired result is achieved by the following claim.

Claim $\mathcal{G}$ is 3-colourable iff $\{s_1, \ldots, s_n\} \models \Phi$

Proof of Claim "$\Rightarrow$": Let V_1, V_2, V_3 be the distribution of V into three colours. Consequently, for every $v \in V$ we have an $r \leq 3$ such that $v \in V_r$. Moreover, for every $v_i, v_j \in V_r$ there is no ℓ s.t. $e_\ell = \{ v_i, v_j \}$. Let $T_r = \{ s_i \mid v_i \in V_r \}$ for each $r \leq 3$, then we show that $\bigcup_{r<3} T_r = T$ and $T_r \models \phi$. This will prove that $T \models \Phi$ because we can split T into three sub-teams such that each satisfies the disjunct.

Since for each $s_i, s_j \in T_r$, there is no edge $e_\ell = \{ v_i, v_j \}$ this implies that for the tuple y_ℓ, we have $s_i(y_\ell) \neq s_j(y_\ell)$ and thereby making the dependence atom trivially true. Moreover, by 2-coherency it is enough to check only for pairs s_i, s_j and since the condition holds for every edge, we have $T_r \models \phi$. Since our assumption is that V can be split into three such sets, we have the split of T into three sub-teams. This gives $T \models \Phi$.

"$\Leftarrow$": Conversely, assume that T can be split into three sub-teams each satisfying ϕ. Then we show that V_1, V_2, V_3 is the partition of V into three colours. Let $V_r = \{ v_i \mid s_i \in T_r \}$ then $\bigcup_{r\leq 3} V_r = V$ and for any $v_i, v_j \in V_r$ there is no edge between v_i, v_j. Suppose to the contrary that there is an edge $e_\ell = \{ v_i, v_j \}$. Then we must have $s_i, s_j \in T_r$ such that $s_i(y_\ell) = 1 = s_j(y_\ell)$. That is, $s_i(y_{\ell,1}) = 1 = s_j(y_{\ell,1}), \ldots, s_i(y_{\ell,n}) = 1 = s_j(y_{\ell,n})$. Since we have that $s_i(x_i) = 0$ whereas $s_j(x_i) = 1$, this implies $\{ s_i, s_j \} \not\models \phi$ which is a contradiction.

This concludes the full proof. □

Theorem 19 MC *parameterised by* formula-size, formula-depth *or* #variables *is* **FPT**.

Proof Recall the relationships between various parameters (Lemma 11). The **FPT**-membership for formula-size and #variables holds due to Proposition 4 and the fact the MC when parameterised by team-size is **FPT**. Furthermore, the **FPT**-membership for formula-depth follows due to the relationship between formula-size and formula-depth. □

Finally, the case for formula-team-tw follows due to Corollary 13 in conjunction with the **FPT** result for team-size (Lemma 17).

Corollary 20 MC *parameterised by* formula-team-tw *is* **FPT**.

4 Satisfiability

In this section, we study SAT under various parameterisations, so the question of whether there exists a team T for a given formula Φ such that $T \models \Phi$. Notice first, that the question is equivalent to finding a singleton team. This is since $\mathcal{PDL}$ is downwards closed. Consequently, if there is a satisfying team, then a singleton team satisfies the formula. As a result, team semantics coincides with the usual Tarskian semantics. This facilitates, for example, determining the truth value of disjunctions in the classical way. Accordingly, simplifying the notation a bit, for SAT we now look for an assignment rather than a singleton team that satisfies the formula.

Corollary 21 *The problem* SAT *under both parameterisations of* formula-team-tw *and* formula-tw *is the same.*

The following result is obtained by classical SAT being **NP**-complete [11, 36].

Corollary 22 SAT *parameterised by* team-size*, or* dep-arity *is* **paraNP**-*complete.*

Proof The 1-slice regarding team-size (a singleton team is the same as an assignment), and the 0-slice regarding dep-arity (no dependence atoms at all) is **NP**-hard. □

Turning towards treewidth, notice that classical propositional SAT is fixed-parameter tractable when parameterised by treewidth due to Samer and Szeider [46, Thm. 1]. However, we are unable to immediately utilise their result because Samer and Szeider study CNF-formulas and we have arbitrary formulas instead. Yet, Lück et al. [38, Cor. 4.7] studying temporal logics under the parameterised approach, classified, as a byproduct, the propositional satisfiability problem with respect to arbitrary formulas to be fixed-parameter tractable.

Corollary 23 SAT *parameterised by* formula-tw *is* **FPT**.

Proof As before, we need to find a singleton team. This implies that split-junctions have the same semantics as classical disjunctions and dependence atoms are always satisfied. Consequently, replacing every occurrence of a dependence atom $\mathsf{dep}(P; Q)$ by $\top$ yields a propositional logic formula. This substitution does not increase the treewidth. Then the result follows by Lück et al. [38, Cor. 4.7]. □

Now, we turn towards the parameter #splits. We present a procedure that constructs a satisfying assignment s such that $s \models \Phi$ if there is one and otherwise it answers no. The idea is that this procedure needs to remember the positions where a modification in the assignment is possible. We show that the number of these positions is bounded by the parameter #splits.

Consider the syntax tree of Φ where, as before, multiple occurrences of subformulas are allowed. The procedure starts at the leaf level with satisfying assignment candidates (partial assignments, to be precise). Reaching the root it confirms whether it is possible to have a combined assignment or not. We assume that the leaves of the tree consist of literals, dependence atoms or negated dependence atoms. Accordingly, the internal nodes of the tree are only conjunction and disjunction nodes. The procedure sets all the dependence atoms to be trivially *true* (as we satisfy them via every singleton team). Moreover, it sets all the negated dependence atoms to be *false* because there can be no satisfying assignment for a negated dependence atom. Additionally, it sets each literal to its respective satisfying assignment. Ascending the tree, it checks the relative conditions for conjunction and disjunction by joining the assignments and thereby giving rise to conflicts. A conflict arises (at a conjunction node) when two assignments are joined with contradicting values for some variable. At this point, it sets this variable x to a conflict state c. At disjunction nodes the assignment stores that it has two options and keeps the assignments separately.

Joining a *true*-value from a dependence atom affects the assignment only at disjunction nodes. This corresponds to the intuition that a formula of the form $\mathsf{dep}(P; Q) \vee \psi$ is true under any assignment.

At a conjunction node, when an assignment s joins with a *true*, the procedure returns the assignment s.

Since at a split the procedure returns both assignments, for k splits there could be $\leq 2^k$-many assignment choices. At the root node if at least one assignment is consistent then we have a satisfying assignment. Otherwise, if all the choices contain conflicts over some variables then there is no such satisfying singleton team.

Theorem 24 SAT *parameterised by* #splits *is* **FPT**. *Moreover, there is an algorithm that solves the problem in* $O(2^{\#\mathsf{splits}(\Phi)} \cdot |\Phi|^{O(1)})$ *for any* $\Phi \in \mathcal{PDL}$.

Proof We consider partial mappings of the kind $t\colon \mathrm{VAR} \to \{0, 1, c\}$. Intuitively, these mappings are used to find a satisfying assignment in the process of the presented algorithm.

If t, t' are two (partial) mappings then $t \circledcirc t'$ is the assignment such that

$$(t \circledcirc t')(x) := \begin{cases} \text{undefined} & \text{, if both } t(x) \text{ and } t'(x) \text{ are undefined,} \\ c & \text{, if both are defined and } t(x) \neq t'(x), \\ t(x) & \text{, if only } t(x) \text{ is defined,} \\ t'(x) & \text{, if only } t'(x) \text{ is defined.} \end{cases}$$

We prove the following claim.

Claim The formula Φ is satisfiable if and only if Algorithm 2 returns a consistent (partial) assignment s that can be extended to a satisfying assignment for Φ over $\mathrm{VAR}(\Phi)$.

Proof of Claim We prove using induction on the structure of Φ.

Base case Start with a variable, $\Phi = x$. Then Φ is satisfiable and $s \models \Phi$ such that $s(x) = 1$. Moreover, such an assignment is returned by the procedure as depicted by line 3 of the algorithm. Similarly, the case $\Phi = \neg x$ follows by line 4. The case $\Phi = \mathsf{dep}(P; Q)$ or $\Phi = \top$ is a special case of a $\mathcal{PDL}$-formula since this is true under any assignment. Line 6 in our procedure returns such an assignment that can be extended to any consistent assignment. Finally, for $\Phi = \bot$ or $\Phi = \neg\mathsf{dep}(P; Q)$, the assignment contains a conflict and can not be extended to a consistent assignment as the algorithm returns "Φ is not satisfiable".

Induction Step. Notice first that if either of the two operands is $\top$ then this is a special case and triggers lines 9–11 of the algorithm thereby giving the satisfying assignment.

Suppose now that $\Phi = \psi_0 \wedge \psi_1$ and that the claim is true for ψ_0 and ψ_1. As a result, both ψ_0 and ψ_1 are satisfiable if and only if the algorithm returns a satisfying assignment for each. Let S_i for $i = 0, 1$ be such that some consistent $t_i' \in S_i$ can be extended to a satisfying assignment t_i for ψ_i. Our claim is that S_Φ returned by the procedure (line 13) is non-empty and contains a consistent assignment for Φ if and only if Φ is satisfiable. First note that, by induction hypothesis, S_i contains all the possible partial assignments that satisfy ψ_i for $i = 0, 1$. Consequently, S_Φ contains all the possible ©-joins of such assignments that can satisfy Φ. Let ψ_0 be satisfied by t'_0 and ψ_1 be satisfied by t'_1. Moreover, let $s' \in S_\Phi$ be an assignment such that $s' = t'_0 © t'_1$. If s' is consistent then s' can be extended to a satisfying assignment s for Φ since $s' \models \psi_i$ for $i = 0, 1$. On the other hand if every $s'_0 © s'_1$ is conflicting (for $s'_i \in S_i$) then there is no assignment over $\text{VAR}(\Phi) = \text{VAR}(\psi_1) \cup \text{VAR}(\psi_2)$ that satisfies Φ. Accordingly, Φ is not satisfiable.

The case for split-junction is simpler. Suppose that $\Phi = \psi_0 \vee \psi_1$ and that the claim is true for ψ_1 and ψ_2. Then Φ is satisfiable if and only if either ψ_0 or ψ_1 is satisfiable. Since the label S_Φ for Φ is the union of all the labels from ψ_0 and ψ_1, it is enough to check that either the label of ψ_0 (that is, S_0) or the label of ψ_1 (S_1) contains a consistent partial assignment. By induction hypothesis, this is equivalent to checking whether ψ_0 or ψ_1 is satisfiable. This completes the case for split-junction and the proof to our claim.

Finally, notice that the label size adds at the occurrence of a split-junction. That is, we keep all the assignment candidates separate and each such candidate is present in the label for split-junction node. In contrast, at conjunction nodes, we 'join' the assignments and for this reason, the label size is the product of the two labels. Notice that we do not get a blow-up in the number of conjunction. This is because, initially the label size for each node is 1 and only at a split-junction, the size increases. This implies that the maximum size for any label is bounded by $2^{\#\text{splits}}$. As a consequence, the above algorithm runs in polynomial time in the input and exponential in the parameter. □

Figure 6 presents an example of using the above algorithm. To simplify the notation, we consider the assignment labels of the form $\{x_i, \neg x_j\}$ rather than $\{x_i \mapsto 1, x_j \mapsto 0\}$.

The remaining cases for the parameters #variables, formula-size, or formula-depth follow easily from the previously shown results.

Theorem 25 SAT *parameterised by* #variables, formula-size, *or* formula-depth*is* **FPT**.

Proof Recall that the question $\mathcal{PDL}$-SAT boils down to $\mathcal{PL}$-SAT of finding an assignment for a given propositional logic formula. The latter problem, when parameterised by the

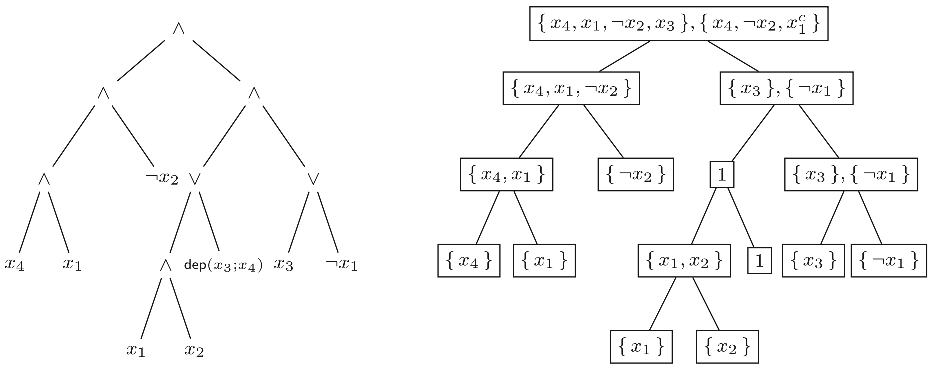


Fig. 6 (left) syntax tree of example formula, and (right) computation of Algorithm 2. Notation: $x/\neg x/x^c$ means a variable is set to true/false/conflict. Clearly, $\{x_4, x_1, \neg x_2, x_3\}$ satisfies the formula

Algorithm 2 SAT-algorithm for #splits which tries to find a satisfying singleton team.

```
Input : $\mathcal{PDL}$-formula $\Phi$ represented by a syntax tree with atomic/negated atomic
        subformulas as leaves
Output: An assignment $s$ such that $s \models \Phi$ or "$\Phi$ is not satisfiable"
1  begin
2      foreach Leaf $\ell$ of the syntax tree do          //atomic or negated-atomic subformula
3          if $\ell = x$ is a variable then $S_\ell \leftarrow \{\{x \mapsto 1\}\}$;
4          else if $\ell = \neg x$ is a negated variable then $S_\ell \leftarrow \{\{x \mapsto 0\}\}$;
5          else if $\ell = \bot$ or $\ell = \neg\mathsf{dep}(\cdot;\cdot)$ then pick $x \in \mathrm{VAR}$ and
           $S_\ell \leftarrow \{\{x \mapsto c\}\}$;
6          else $S_\ell \leftarrow \{1\}$;                  // case $\top$ or $\mathsf{dep}(\cdot;\cdot)$
7      foreach Inner node $\ell$ of the syntax tree in bottom-up order do
8          Let $\ell_0, \ell_1$ be the children of $\ell$ with $S_0, S_1$ the resp. sets
           partial assignments;
9          if $1 \in S_i$ then
10             if $\ell$ is a conjunction then $S_\ell \leftarrow S_{1-i}$;
11             else $S_\ell \leftarrow \{1\}$;      // empty split for a split-junction
12         else if $\ell$ is a conjunction then
13             foreach $s_0 \in S_0$ and $s_1 \in S_1$ do $S_\ell \leftarrow S_\ell \cup \{s_0 ^{\copyright}\ s_1\}$ ;
14         else                                     // $\ell$ is a split-junction
15             foreach $s_0 \in S_0$ and $s_1 \in S_1$ do $S_\ell \leftarrow S_\ell \cup \{s_0, s_1\}$ ;
16     if there exists a non-conflicting assignment $s \in S_\Phi$ then return $s$;
17     else return ''$\Phi$ is not satisfiable'';
```

number of variables in the input formula, is **FPT** which implies that the former problem is also **FPT**.

Note that formula-size $= |\Phi|$ and any parameterised problem Π is **FPT** for the parametrisation input-length. Consequently, SAT parameterised by formula-size is **FPT**.

If a formula Φ has formula-depth $= d$ then there are $\leq 2^d$ leaves and $\leq 2^d$ internal nodes accordingly we have formula-size $\leq 2^{2d}$ which shows **FPT** membership, when parameterised by formula-depth. □

4.1 A satisfiability variant

The shown results suggest that it might be interesting to study the following variant of SAT, in which we impose an additional input 1^m (unary encoding) with $m \geq 2$ and ask for a satisfying team of size at least m. Let us call the problem m-SAT. Notice that since $\mathcal{PDL}$-formulas are downwards closed, m-SAT is equivalent to finding a team of size exactly m. We wish to emphasise that m-SAT is not the same as SAT parameterised by team-size. This is because, SAT does not ask for a team of a particular size. We also wish to mention here that the problem of finding a team of a given size (m-SAT) has not been studied before in the classical setting. A related yet different problem in the context of database is the existence of an Armstrong relation [3, 19]. Given a set Γ of functional dependencies, the question is whether there exists a database D that satisfies a functional dependency $\mathsf{dep}(P; Q)$ if and only if $\mathsf{dep}(P; Q)$ is implied by Γ. In our setting, the problem m-SAT considers general $\mathcal{PDL}$ formulas (so not only functional dependencies). Moreover, m-SAT questions whether it is possible to generate a database of a particular size (m in our case), whereas the restriction that it should not satisfy any other functional dependency, is dropped. We begin by classifying the problem unparameterised, followed by proving results in parameterised setting.

The motivation for this computational task stems from the fact that, often one has a set of constraints and wishes to populate the database with a certain number of entries (m in our case). Consider a person R making a teaching-schedule at a university department. There are quite many lectures to arrange weekly and R has to follow certain constraints such as 'the teacher and the room determine the course', 'the teacher and the time determine room and the course' and many more. Obviously, R would like to have an algorithm such that the input to this algorithm is this collection of constraints as well as the solution size (the number of weekly lectures) and the algorithm makes a consistent teaching schedule for him/her.

Notice also that we require m to be given in unary. This is because, R might be looking for an arbitrary large solution. Consequently, it makes sense to ask whether we can fill the database (even with arbitrary large size) in polynomial time with respect to the input. This corresponds to the intuition that we are given an empty database with allocated memory of size m (that is, with empty rows) and we wish to populate it with the data such that it satisfies the constraints.

If we lift the restriction of m not being given in unary then the problem immediately becomes **NEXP**-complete. The hardness follows due to a reduction from the validity problem for $\mathcal{PDL}$ [49]. In polynomial time, one can count the number of proposition symbols n in a formula Φ and ask whether there is a satisfying team for Φ of size m where $m = 2^n$. It is easy to observe that Φ is valid if and only if Φ has a satisfying team of size m.

We prove in Theorem 26 that restricting m to be unary drops the complexity to **NP**-completeness.

Theorem 26 *The problem m-*SAT *is* **NP**-*complete when m is given in unary.*

Proof Notice first that finding a team of size m is computationally harder than finding a team of size 1. Consequently, the hardness follows because SAT for $\mathcal{PDL}$ is **NP**-hard.

We prove **NP**-membership by presenting a non-trivial nondeterministic algorithm running in polynomial time that constructs a team of size m if it exists. Intuitively, the algorithm tries to construct a satisfying team of size m for a given formula Φ in bottom-up fashion. Starting at the level of atomic or negated atomic subformulas of Φ, the algorithm iteratively builds a team T_i for $i \in \mathrm{SF}(\Phi)$.

Given the input Φ, the algorithm labels each node of the tree with a satisfying team of size at most m. For each atomic/negated atomic subformula, this team is guessed nondeterministically, whereas, for each conjunction (resp., split-junction), the team is the intersection (union) of the two subteams from the successor nodes. Specifically, for each literal ℓ, the label T_ℓ is a team of size at most m and $T_\ell = \{\, s_i \mid s_i(\ell) = 1 \,\}$. The question of how are the other variables mapped by each s_i is answered nondeterministically. For each dependence atom $\alpha = \mathsf{dep}(X; Y)$, the label T_α is a team such that $|T_\alpha| = \min\{m, 2^{|X|}\}$. If $|T_\alpha| = m$, the assignments over $\mathrm{VAR}(\Phi)$ are selected nondeterministically in such a way that $T_\alpha \models \alpha$. However, if $|T_\alpha| = 2^{|X|}$, then each assignment over X is selected and these assignment are extended to variables in $\mathrm{VAR}(\phi)\setminus X$ nondeterministically. At conjunctions, the label is the intersection of the two labels from the successor nodes and at split-junctions, the label is

Algorithm 3 Algorithm for finding a satisfying team of size m.

```
Input  : $\mathcal{PDL}$-formula $\Phi$ represented by a syntax tree with atomic/negated atomic
         subformulas as leaves
Output: A team $T$ of size $m$ such that $T \models \Phi$ or "$\Phi$ does not have a satisfying team
         of size $m$"
1 begin
2     foreach Leaf $\ell$ of the syntax tree do
3         nondeterministically guess a team $T_\ell$ such that $|T_\ell| \leq m$ and $T_\ell \models \ell$
4     foreach Inner node $\ell$ of the syntax tree in bottom-up order do
5         Let $\ell_0, \ell_1$ be the children of $\ell$ with $T_0, T_1$ the resp. team labels;
6         if $\ell$ is a conjunction then $T_\ell = T_0 \cap T_1$ ;
7         else $T_\ell = T_0 \cup T_1$;
8     if $|T_\Phi| \geq m$ then return $T_\Phi$;
9     else return "$\Phi$ does not have a satisfying team of size $m$";
```

the union. At the root level, if T_Φ has size at least m then Algorithm 3 accepts, otherwise it rejects. The result follows from the downward closure property of $\mathcal{PDL}$-formulas (see page 6). Moreover, the maximum label size for any node can be at most $|\Phi| \cdot m$ because the size only increases (potentially doubles) at a split-junction. This is unproblematic since m is given in unary. □

Claim Φ has a satisfying team of size m if and only if Algorithm 3 outputs such a team when given the input Φ. Moreover, Algorithm 3 runs in nondeterministic polynomial time.

Proof of Claim The result is justified by Algorithm 3 constructing the truth function for Φ in a bottom-up fashion. The idea of a truth function was introduced by Yang [51].

"$\Rightarrow$": Let T be a satisfying team for Φ of size m. Consider the syntax tree of Φ. Each node in the tree can be labelled with a satisfying team for this node. Moreover, the label for each node is a subteam of T, consequently, the size is bounded by m. Algorithm 3 simply selects those teams in the guessing phase that are labelled at the leaf nodes of Φ by the truth function. These teams correctly 'add-up' to T at the root node and $T \models \Phi$ due to our assumption.

"$\Leftarrow$": Suppose that Algorithm 3 outputs a team of size at least m. Let $T_1, \ldots T_n$ be the teams labelled by the algorithm at the leaves of Φ, where n is the number of atomic or negated atomic subformulas (leaves in the syntax tree) of Φ. First note that $T_i \models \ell_i$ where ℓ_i a leaf of Φ and $i \leq n$. Moreover, for each split-junction, $\beta = \alpha_i \vee \alpha_j$ the team label T_β for the node β is the union $T_{\alpha_i} \cup T_{\alpha_j}$, where T_{α_s} is the team label for α_s and $s \in \{i, j\}$. Similarly at the conjunction nodes, the team label for the parent node is the intersection of the teams from each conjunct. Finally, at the root level, the label has size m. Now, working in the backward direction, one can obtain the truth function for $\langle T, \Phi \rangle$ proving that $T \models \Phi$ where T is a team of size at least m. □

This completes the proof to the theorem. Now, we move on to the parameterised complexity of m-SAT.

Theorem 27 *m-SAT parameterised by* #variables*,* formula-size*, or* formula-depth *is* **FPT***, with time even linear in the input length.*

Proof In the case of #variables, the maximal satisfying team has size $2^{\#\text{variables}}$. Moreover, there are a total of $2^{2^{\#\text{variables}}}$ many teams. Consequently, we can find all satisfying teams of size m (if any) in **FPT**-time with respect to the parameter #variables.

For formula-size, notice first that #variables $\leq$ formula-size, that is, bounding formula-size also bounds #variables (Lem. 11). As a result, we have that m-SAT parameterised by formula-size is **FPT**.

For formula-depth notice that formula-size $\leq 2^{2\cdot\text{formula-depth}}$ and thereby the problem is **FPT** under this parametrisation.

Finally, consider the brute-force (bottom-up) algorithm in each case. This algorithm evaluates each suformula against the candidate subteam for this subformula. As a consequence, this implies that the running time with respect to the input is linear in the number of subformulas. That is, the algorithm runs in time $f(k) \cdot |\Phi|$ for a computable function f. □

Neither the arity of the dependence atoms nor the team-size alone are fruitful parameters which follows from Corollary 22.

Corollary 28 *m-SAT parameterised by* team-size *or* dep-arity *is* **paraNP***-complete.*

5 Conclusion

In this paper, we started a systematic study of the parameterised complexity of model checking and satisfiability in propositional dependence logic. For both problems, we exhibited a complexity dichotomy (see Table 2): depending on the parameter, the problem is either **FPT** or **paraNP**-complete. Interestingly, there exist parameters for which MC is easy, but SAT is hard (team-size) and *vice versa* (#splits).

Towards the end, we introduced a satisfiability question which also asks for a team of a given size (m-SAT). To the best of the authors' knowledge, this problem has not been studied at all in the setting of team logics. We show that in the classical setting the problem is **NP**-complete. The parameterised complexity of this problem currently behaves similarly to SAT though the parameters #splits and formula-tw are open for further research. Another related question would be to study the relation between the natural number $m \in \mathbb{N}$ and the properties expressible in the downwards closed team based logic. In other words, what type of $\mathcal{PDL}$-formulas have a satisfying team of size m?

As future work, we want to study combinations of the studied parameters, e.g., #splits + dep-arity. This parameter is quite interesting, as dep-arity alone is always hard for all three problems, whereas #splits allows for SAT to reach **FPT**. It is also interesting to observe that in both of our reductions for proving hardness of MC under the parametrisation #splits and dep-arity, if dep-arity is fixed then #splits is unbounded and vice versa. The authors believe that the combination (#splits + dep-arity) may not yield **FPT**-membership although it might lower the complexity from **paraNP**-hardness.

The concept of backdoor sets [50] has turned out to be a significant parameter to study. Intuitively, it utilises the distance of a given problem instance to an efficient sub-case. When it was firstly introduced, the concept was defined with respect to CNF formulas of propositional logic. Yet, it has been transferred to different kind of logics and other areas as well: abduction [42], answer set programming [21, 22], argumentation [17], default logic [20], temporal logic [40], planning [35], and constraint satisfaction [25]. For dependence logic, such a formalism does not exist so far. We discussed in Section 4, that an instance Φ of $\mathcal{PDL}$-SAT can be reduced to Ψ of $\mathcal{PL}$-SAT such that Φ and Ψ are equi-satisfiable. This is achieved by simply replacing every dependence atom $\mathsf{dep}(P; Q)$ by $\top$. This implies that the backdoor approach of classical propositional logic for SAT also applies to propositional dependence logic. It would be interesting to study backdoors in the context of MC as well, that is, given a team T and a $\mathcal{PDL}$-formula Φ, whether $T \models \Phi$? Some approaches into the direction of a good notion of backdoor sets are promising others not. Taking the backdoor set to be a collection of variables (as in the propositional setting) is problematic, because we wish to talk about total assignments which implies that this set should contain all variables. If the backdoor set is a collection of sub-teams, then an approach via sub-solvers makes sense, but which sub-team corresponds to which sub-formula should also be specified by the backdoor set (otherwise the sub-solver has to guess it nondeterministically). Mimicking the idea of resolvents (as in the CNF-setting) is not immediate, as removing dependence atoms does not allow for removing the assignments appearing in the backdoor: e.g., for $x \wedge \mathsf{dep}(P; Q)$ the literal x can be problematic.

Also, further operators such as independence and inclusion atoms will be interesting to consider. Closing, another important question for future research is to consider the parameterised version of validity and implication problem for $\mathcal{PDL}$.

Acknowledgements The authors thank the anonymous referees for their comments in improving the paper. We thank Phokion Kolaitis, Juha Kontinen, and Sebastian Link for discussions on m-SAT. We are grateful to Jonni Virtema for his thoughts on the #splits parameter in the database setting. This work was supported by the German Research Foundation (DFG), project ME 4279/1-2. The authors would like to thank the participants and organisers of the Dagstuhl Seminar 19031 "Logics for Dependence and Independence" [27] for valuable comments and suggestions after the presentation of partial results of this article.

Funding Open Access funding enabled and organized by Projekt DEAL.

References

1. Abramsky, S., Kontinen, J., Väänänen, J., Vollmer, H.: Dependence Logic, Theory and Applications. Springer, Berlin (2016). https://doi.org/10.1007/978-3-319-31803-5
2. Arnborg, S., Corneil, D.G., Proskurowski, A.: Complexity of finding embeddings in a k-tree. SIAM J. Algebraic Discrete Methods **8**(2), 277–284 (1987). https://doi.org/10.1137/0608024
3. Beeri, C., Dowd, M., Fagin, R., Statman, R.: On the structure of armstrong relations for functional dependencies. J. ACM **31**(1), 30–46 (1984). https://doi.org/10.1145/2422.322414
4. Bläsius, T., Friedrich, T., Schirneck, M.: The Parameterized Complexity of Dependency Detection in Relational Databases. In: Guo, J., Hermelin, D. (eds.) 11th International Symposium on Parameterized and Exact Computation (IPEC 2016), Leibniz International Proceedings in Informatics (LIPIcs), vol. 63, pp. 6:1–6:13. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl (2017). https://doi.org/10.4230/LIPIcs.IPEC.2016.6. http://drops.dagstuhl.de/opus/volltexte/2017/6920

5. Bodlaender, H.L.: A tourist guide through treewidth. Acta Cybern. **11**(1-2), 1–21 (1993). https://cyber.bibl.u-szeged.hu/index.php/actcybern/article/view/3417
6. Chen, H., Mengel, S.: A trichotomy in the complexity of counting answers to conjunctive queries. In: Arenas, M., Ugarte, M. (eds.) 18th International Conference on Database Theory, ICDT 2015, March 23-27, 2015, Brussels, Belgium, *LIPIcs*, vol. 31, pp. 110–126. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik (2015). https://doi.org/10.4230/LIPIcs.ICDT.2015.110
7. Chen, H., Mengel, S.: Counting answers to existential positive queries: A complexity classification. In: Milo, T., Tan, W. (eds.) Proceedings of the 35th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2016, pp. 315–326. ACM, San Francisco (2016). https://doi.org/10.1145/2902251.2902279
8. Chen, H., Mengel, S.: The logic of counting query answers. In: 32nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2017, pp. 1–12. IEEE Computer Society, Reykjavik (2017). https://doi.org/10.1109/LICS.2017.8005085
9. Chen, H., Müller, M.: The fine classification of conjunctive queries and parameterized logarithmic space. TOCT **7**(2), 7:1–7:27 (2015). https://doi.org/10.1145/2751316
10. Ciardelli, I., Groenendijk, J., Roelofsen, F.: Towards a logic of information exchange - an inquisitive witness semantics. In: Logic, Language, and Computation - 9th International Tbilisi Symposium on Logic, Language, and Computation, TbiLLC 2011, Kutaisi, Revised Selected Papers, pp. 51–72 (2011). https://doi.org/10.1007/978-3-642-36976-6_6
11. Cook, S.A.: The complexity of theorem-proving procedures. In: Harrison, M.A., Banerji, R.B., Ullman, J.D. (eds.) Proceedings of the 3rd Annual ACM Symposium on Theory of Computing, pp. 151–158. ACM, Shaker Heights (1971). https://doi.org/10.1145/800157.805047
12. Cygan, M., Fomin, F.V., Kowalik, L., Lokshtanov, D., Marx, D., varPilipczuk, M., varPilipczuk, M., Saurabh, S.: Parameterized Algorithms. Springer (2015). https://doi.org/10.1007/978-3-319-21275-3
13. Downey, R.G., Fellows, M.R.: Fundamentals of Parameterized Complexity. Texts in Computer Science. Springer (2013). https://doi.org/10.1007/978-1-4471-5559-1
14. Durand, A., Hannula, M., Kontinen, J., Meier, A., Virtema, J.: Approximation and dependence via multiteam semantics. Ann. Math. Artif. Intell. **83**(3-4), 297–320 (2018). https://doi.org/10.1007/s10472-017-9568-4
15. Durand, A., Hannula, M., Kontinen, J., Meier, A., Virtema, J.: Probabilistic team semantics. In: Ferrarotti, F., Woltran, S. (eds.) Foundations of Information and Knowledge Systems - 10th International Symposium, FoIKS 2018, Proceedings, *Lecture Notes in Computer Science*, vol. 10833, pp. 186–206. Springer, Budapest (2018). https://doi.org/10.1007/978-3-319-90050-6_11
16. Durand, A., Mengel, S.: Structural tractability of counting of solutions to conjunctive queries. Theory Comput. Syst. **57**(4), 1202–1249 (2015). https://doi.org/10.1007/s00224-014-9543-y
17. Dvořák, W., Ordyniak, S., Szeider, S.: Augmenting tractable fragments of abstract argumentation. **186**, 157–173 (2012). https://doi.org/10.1016/j.artint.2012.03.002
18. Ebbing, J., Lohmann, P.: Complexity of model checking for modal dependence logic. In: Bieliková, M., Friedrich, G., Gottlob, G., Katzenbeisser, S., Turán, G. (eds.) SOFSEM 2012: Theory and Practice of Computer Science - 38th Conference on Current Trends in Theory and Practice of Computer Science, Proceedings, *Lecture Notes in Computer Science*, vol. 7147, pp. 226–237. Springer, Špindlerův Mlýn (2012). https://doi.org/10.1007/978-3-642-27660-6_19
19. Fagin, R., Vardi, M.Y.: Armstrong databases for functional and inclusion dependencies. Inf. Process. Lett. **16**(1), 13–19 (1983). https://doi.org/10.1016/0020-0190(83)90005-4
20. Fichte, J.K., Meier, A., Schindler, I.: Strong backdoors for default logic. In: Theory and Applications of Satisfiability Testing - SAT 2016 - 19th International Conference, Proceedings, pp. 45–59, Bordeaux (2016). https://doi.org/10.1007/978-3-319-40970-2_4
21. Fichte, J.K., Szeider, S.: Backdoors to normality for disjunctive logic programs. ACM Trans. Comput. Log. **17**(1), 7:1–7:23 (2015). https://doi.org/10.1145/2818646
22. Fichte, J.K., Szeider, S.: Backdoors to tractable answer set programming. Artif. Intell. **220**, 64–103 (2015). https://doi.org/10.1016/j.artint.2014.12.001
23. Flum, J., Grohe, M.: Parameterized Complexity Theory. Texts in Theoretical Computer Science. An EATCS Series. Springer (2006). https://doi.org/10.1007/3-540-29953-X
24. Galliani, P.: Inclusion and exclusion dependencies in team semantics - on some logics of imperfect information. Ann. Pure Appl. Logic **163**(1), 68–84 (2012). https://doi.org/10.1016/j.apal.2011.08.005
25. Gaspers, S., Misra, N., Ordyniak, S., Szeider, S., Zivny, S.: Backdoors into heterogeneous classes of SAT and CSP. J. Comput. Syst. Sci. **85**, 38–56 (2017). https://doi.org/10.1016/j.jcss.2016.10.007
26. Grädel, E.: Model-checking games for logics of imperfect information. Theor. Comput. Sci. **493**, 2–14 (2013). https://doi.org/10.1016/j.tcs.2012.10.033

27. Grädel, E., Kontinen, J., Väänänen, J., Vollmer, H.: Logics for Dependence and Independence (Dagstuhl Seminar 15261). Dagstuhl Reports **5**(6), 70–85 (2016). https://doi.org/10.4230/DagRep.5.6.70. http://drops.dagstuhl.de/opus/volltexte/2016/5508
28. Grohe, M., Schwentick, T., Segoufin, L.: When is the evaluation of conjunctive queries tractable?. In: Vitter, J.S., Spirakis, P.G., Yannakakis, M. (eds.) Proceedings on 33rd Annual ACM Symposium on Theory of Computing, pp. 657–666. ACM, Heraklion (2001). https://doi.org/10.1145/380752.380867
29. Hannula, M., Kontinen, J., Virtema, J.: Polyteam semantics. In: Artëmov, S.N., Nerode, A. (eds.) Logical Foundations of Computer Science - International Symposium, LFCS 2018, Proceedings, *Lecture Notes in Computer Science*, vol. 10703, pp. 190–210. Springer, Deerfield Beach (2018). https://doi.org/10.1007/978-3-319-72056-2_12
30. Hannula, M., Kontinen, J., Virtema, J., Vollmer, H.: Complexity of propositional logics in team semantic. ACM Trans. Comput. Log. **19**(1), 2:1–2:14 (2018). https://doi.org/10.1145/3157054
31. Hella, L., Kuusisto, A., Meier, A., Virtema, J.: Model checking and validity in propositional and modal inclusion logics. J. Log. Comput (2019). https://doi.org/10.1093/logcom/exz008
32. Kontinen, J., Müller, J., Schnoor, H., Vollmer, H.: Modal independence logic. J. Log. Comput. **27**(5), 1333–1352 (2017). https://doi.org/10.1093/logcom/exw019
33. Krebs, A., Meier, A., Virtema, J.: A team based variant of CTL. In: Grandi, F., Lange, M., Lomuscio, A. (eds.) 22nd International Symposium on Temporal Representation and Reasoning, TIME 2015, pp. 140–149. IEEE Computer Society, Kassel (2015). https://doi.org/10.1109/TIME.2015.11
34. Krebs, A., Meier, A., Virtema, J., Zimmermann, M.: Team semantics for the specification and verification of hyperproperties. In: Potapov, I., Spirakis, P.G., Worrell, J. (eds.) 43rd International Symposium on Mathematical Foundations of Computer Science, MFCS 2018, *LIPIcs*, vol. 117, pp. 10:1–10:16. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Liverpool (2018). https://doi.org/10.4230/LIPIcs.MFCS.2018.10
35. Kronegger, M., Ordyniak, S., Pfandler, A.: Variable-deletion backdoors to planning. In: Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, pp. 3305–3312, Austin (2015). http://www.aaai.org/ocs/index.php/AAAI/AAAI15/paper/view/9885
36. Levin, L.: Universal search problems. Problems of Information Transmission **9**(3), 115–116 (1973)
37. Lohmann, P., Vollmer, H.: Complexity results for modal dependence logic. Studia Logica **101**(2), 343–366 (2013). https://doi.org/10.1007/s11225-013-9483-6
38. Lück, M., Meier, A., Schindler, I.: Parameterised complexity of satisfiability in temporal logic. ACM Trans. Comput. Log. **18**(1), 1:1–1:32 (2017). https://doi.org/10.1145/3001835
39. Mahmood, Y., Meier, A.: Parameterised complexity of model checking and satisfiability in propositional dependence logic. In: Herzig, A., Kontinen, J. (eds.) Foundations of Information and Knowledge Systems - 11th International Symposium, FoIKS 2020, Proceedings, *Lecture Notes in Computer Science*, vol. 12012, pp. 157–174. Springer, Dortmund (2020). https://doi.org/10.1007/978-3-030-39951-1_10
40. Meier, A., Ordyniak, S., Sridharan, R., Schindler, I.: Backdoors for linear temporal logic. In: 11th International Symposium on Parameterized and Exact Computation, IPEC 2016, pp. 23:1–23:17, Aarhus (2016). https://doi.org/10.4230/LIPIcs.IPEC.2016.23
41. Meier, A., Reinbold, C.: Enumeration complexity of poor man's propositional dependence logic. In: Ferrarotti, F., Woltran, S. (eds.) Foundations of Information and Knowledge Systems - 10th International Symposium, FoIKS 2018, Proceedings, *Lecture Notes in Computer Science*, vol. 10833, pp. 303–321. Springer, Budapest (2018). https://doi.org/10.1007/978-3-319-90050-6_17
42. Pfandler, A., Rümmele, S., Szeider, S.: Backdoors to abduction. In: Rossi, F. (ed.) Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI'13), pp. 1046–1052, Beijing (2013)
43. varPippenger, N.: Theories of computability. Cambridge University Press (1997)
44. Robertson, N., Seymour, P.D.: Graph minors. v. excluding a planar graph. J. Comb. Theory, Ser. B **41**(1), 92–114 (1986). https://doi.org/10.1016/0095-8956(86)90030-4
45. Samer, M., Szeider, S.: Fixed-parameter tractability. In: Biere, A., Heule, M., van Maaren, H., Walsh, T. (eds.) Handbook of Satisfiability, *Frontiers in Artificial Intelligence and Applications*, vol. 185, pp. 425–454. IOS Press (2009). https://doi.org/10.3233/978-1-58603-929-5-425
46. Samer, M., Szeider, S.: Algorithms for propositional model counting. J. Discrete Algorithm. **8**(1), 50–64 (2010)
47. Väänänen, J.A.: Dependence Logic - A New Approach to Independence Friendly Logic, *London Mathematical Society student texts*, vol. 70. Cambridge University Press (2007). http://www.cambridge.org/de/knowledge/isbn/item1164246/?site%_locale=de_DE
48. Väänänen, J.A.: Modal dependence logic. In: Apt, K., van Rooij, R. (eds.) New Perspectives on Games and Interaction. Amsterdam University Press (2008)
49. Virtema, J.: Complexity of validity for propositional dependence logics. Inf. Comput. **253**, 224–236 (2017). https://doi.org/10.1016/j.ic.2016.07.008

50. Williams, R., Gomes, C.P., Selman, B.: Backdoors to typical case complexity. In: IJCAI-03, Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence, Acapulco, pp. 1173–1178 (2003)
51. Yang, F., Väänänen, J.: Propositional logics of dependence. Ann. Pure Appl. Logic **167**(7), 557–589 (2016). https://doi.org/10.1016/j.apal.2016.03.003