Implementation of Elements of Dependence Logic

Lewandowska, Iga Kalina[1] and Seijas Vega, José[2]

Technical University of Denmark, DK-2800 Kgs Lyngby, Denmark

Abstract. This paper explores the meaning of Dependence Logic formulas in context of specified datasets. It presents a variation of Dependence Logic aimed to be used in practical applications of analysing variable determination in given finite datasets. It introduces and defines the concepts of a mapping function and a descriptive formula, and presents their use in model checking by transforming the modified Dependence Logic formula into equivalent Propositional Logic formulas describing data subsets. Additionally, it presents a practical algorithm to verify given atomic Dependence Logic formulas in a dataset, and a method to find all such dependencies.

Keywords: Dependence logic · Team semantics · Model checking.

1 Introduction

Dependence can be defined as determination of one by another. In this form it is present in many systems we observe and describe. It is then useful to have a language describing the existence of such dependencies. *Dependence Logic* is an extension of First Order Logic (FOL) which allows expressing the condition of complete determination of one variable by a set of variables.

As dependence can only be observed by switching the value of a variable and observing the effect of that change on another variable, the logic has to be interpreted with respect to a set of assignments, in other words, within so-called *team semantics*. Given a certain set of assignments, we can talk about the FOL formula's satisfaction for each of them separately. However, to reason about determination, we must consider the whole set simultaneously. Dependence Logic introduces atomic Dependence Logic formulas, which describe the determination of a value of one variable by the values of a set of other variables. The presence of such formulas requires redefinition of truth - in Dependence Logic we talk about a formula describing a *type* of a team.

2 Preliminaries

In this paper we consider Dependence Logic as introduced by Jouko Väänänen [6]. It is used as a basis and inspiration for the proposed practical framework for exploring dependencies in data. This section presents the basic concepts and terminology connected to Dependence Logic, but for the sake of brevity some elements will be omitted. The reader is advised to consult [6].

A vocabulary L is a set of constant, relation and function symbols. Variables are denoted $x_0, x_1, \ldots$. The terms of L are denoted $t_1, t_2, \ldots$. The set $Var(t)$ is a set of variables present in t, with constant terms having $Var(t) = \emptyset$. Every constant term t has a definite value $t^{\mathcal{M}}$ in any L-structure $\mathcal{M}$, defined inductively as follows: if t is a constant symbol, $t^{\mathcal{M}}$ is defined already. Otherwise, $(ft_1 \ldots t_n)^{\mathcal{M}} = f^{\mathcal{M}}(t_1^{\mathcal{M}} \ldots t_n^{\mathcal{M}})$. An assignment s assigns a value $t^{\mathcal{M}}\langle s\rangle$ in co-domain M to any L-term t such that $Var(t) \subseteq dom(s)$ as follows: $c^{\mathcal{M}}\langle s\rangle = c^{\mathcal{M}}, x_n^{\mathcal{M}}\langle s\rangle = s(x_n)$ and $(ft_1 \ldots t_n)^{\mathcal{M}}\langle s\rangle = f^{\mathcal{M}}(t_1^{\mathcal{M}}\langle s\rangle, \ldots, t_n^{\mathcal{M}}\langle s\rangle)$. The set $Fr(\phi)$ of free variables in ϕ is defined as usual.

We will interpret our language on databases. An example of a database is presented in Table 1. The rows of the table are considered *assignments* (or *agents*) and columns are *variables (features)*. *Teams* are sets of assignments (a selection of rows). We assume that all assignments in a team share the same domain (set of variables) and all the variables have the same co-domain (set of possible values).

Dependence Logic enriches FOL by adding an atomic Dependence Logic formula: $=(x_1, \ldots, x_{n-1}, x_n)$, whose meaning is as follows: 'the value of the term x_n depends only on the values of the terms $x_1, \ldots, x_{n-1}$' [6]. For clarity, in the atomic formula $=(x_0, x_1, \ldots, x_{n-1}, x_n)$, the variables $x_0, x_1, \ldots, x_{n-1}$ will be called *dependence arguments* and x_n will be referred to as a *dependent variable*. The formula does not give any information about value of x_n, which means that any single value can satisfy the formula, if there is no assignment that has a different value of x_n for given set of values $x_1, \ldots, x_{n-1}$.

	x_0	x_1	x_2
s_0	1	3	2
s_1	1	1	2
s_2	1	1	4
s_3	2	4	1
s_4	3	1	2

Table 1: A database

Definition 1 ([6]). *Suppose L is a vocabulary and $\mathcal{M}$ is an L-structure. We define the set $\mathcal{T}$, or more precisely $\mathcal{T}_{\mathcal{M}}$, called the* fundamental predicate *of $\mathcal{M}$, of triples (ϕ, X, d), where ϕ is an L-formula of dependence logic, $Fr(\phi) \subseteq dom(X)$ and $d \in \{0, 1\}$. Set $\mathcal{T}$ satisfies:*

$(t_1 = t_2, X, 1) \in \mathcal{T}$ *iff for all* $s \in X$ *we have* $t_1^{\mathcal{M}}\langle s\rangle = t_2^{\mathcal{M}}\langle s\rangle$; (E1)

$(t_1 = t_2, X, 0) \in \mathcal{T}$ *iff for all* $s \in X$ *we have* $t_1^{\mathcal{M}}\langle s\rangle \neq t_2^{\mathcal{M}}\langle s\rangle$; (E2)

$(=(t_1, \ldots, t_n), X, 1) \in \mathcal{T}$ *iff for all* $s, s' \in X$ *such that*
$t_1^{\mathcal{M}}\langle s\rangle = t_1^{\mathcal{M}}\langle s'\rangle, \ldots, t_{n-1}^{\mathcal{M}}\langle s\rangle = t_{n-1}^{\mathcal{M}}\langle s'\rangle$, *we have* $t_n^{\mathcal{M}}\langle s\rangle = t_n^{\mathcal{M}}\langle s'\rangle$; (E3)

$(=(t_1, \ldots, t_n), X, 0)) \in \mathcal{T}$ *iff* $X = \emptyset$; (E4)

$(Rt_1, \ldots, t_n, X, 1) \in \mathcal{T}$ *iff for all* $s \in X$, $(t_1^{\mathcal{M}}\langle s\rangle, \ldots, t_n^{\mathcal{M}}\langle s\rangle) \in R^{\mathcal{M}}$; (E5)

$(Rt_1, \ldots, t_n, X, 0) \in \mathcal{T}$ *iff for all* $s \in X$, $(t_1^{\mathcal{M}}\langle s\rangle, \ldots, t_n^{\mathcal{M}}\langle s\rangle) \notin R^{\mathcal{M}}$; (E6)

$(\phi \vee \psi, X, 1) \in \mathcal{T}$ *iff* $X = Y \cup Z$
where $dom(Y) = dom(Z)$, $(\phi, Y, 1) \in \mathcal{T}$ *and* $(\psi, Z, 1) \in \mathcal{T}$; (E7)

$(\phi \vee \psi, X, 0) \in \mathcal{T}$ *iff* $(\phi, X, 0) \in \mathcal{T}$ *and* $(\psi, X, 0) \in \mathcal{T}$; (E8)

$(\neg\phi, X, 0) \in \mathcal{T}$ *iff* $(\phi, X, 1) \in \mathcal{T}$; (E9)

$(\neg\phi, X, 1) \in \mathcal{T}$ *iff* $(\phi, X, 0) \in \mathcal{T}$; (E10)

$(\exists x_n\phi, X, 1) \in \mathcal{T}$ *iff* $(\phi, X(F/x_n), 1) \in \mathcal{T}$ *for some* $F : X \to M$; (E11)

$(\exists x_n\phi, X, 0) \in \mathcal{T}$ *iff* $(\phi, X(M/x_n), 0) \in \mathcal{T}$. (E12)

Subsequently, we will say that X is *of type* ϕ in $\mathcal{M}$, denoted $\mathcal{M} \models_X \phi$, if $(\phi, X, 1) \in \mathcal{T}$; ϕ is *true* in $\mathcal{M}$, denoted $\mathcal{M} \models \phi$, if $\models_{\{\emptyset\}}\phi$; ϕ is *valid*, denoted $\models \phi$, if $\mathcal{M} \models \phi$ for all $\mathcal{M}$.

The key concepts needed to understand points E11 and E12 of the definition, and hence the use of quantifiers in Dependence Logic, are duplicate and supplement.

Definition 2 ([6]). *If M is a set, X is a team with M as its codomain and $F : X \to M$, we let $X(F/x_n)$ denote the supplement team $\{s(F(s)/x_n) : s \in X\}$.*

Definition 3 ([6]). *If M is a set and X is a team of M we use $X(M/x_n)$ to denote the duplicate team $\{s(a/x_n) : a \in M, s \in X\}$*

It is also relevant to point out that an empty team has the type of any Dependence Logic formula ϕ (and thus also $\neg\phi$ at the same time). This leads to a conclusion that there exist no formulas ϕ and ψ of Dependence Logic such that if $\mathcal{M} \models_X \phi$, then $\mathcal{M} \not\models_X \psi$.

Furthermore, the types in Dependence Logic are subject to downward closure. As dependence means that there are no counterexamples for the variable value determination, in case of a team of certain type, removing assignments from the set preserves the type of team: if $X \models \phi$ and $Y \subseteq X$ then $Y \models \phi$.

The truth in Dependence Logic is not subject to the law of excluded middle. Given a team X, one of the following will hold:

- $(\phi, X, 1)$ - X is of type ϕ;
- $(\phi, X, 0)$ (equivalent to $(\neg\phi, X, 1)$) - either for each assignment in X ϕ evaluates to *False* or the team X is empty;
- and finally neither of the previous two - X's type is not expressed neither by ϕ nor $\neg\phi$.

There is an interesting consequence of E7. The definition requires to divide the set of assignments into subsets in order to check if a formula holds. For a disjunction, to decide if a team X is of type $\phi = \phi_1 \vee \phi_2$, we need to find two subsets of X, X_1 and X_2, such that $X_1 \cup X_2 = X$, X_1 is of type ϕ_1 and X_2 is of type ϕ_2 (E7). This property allows us to describe different dependence types in one set by splitting it into subsets described by different formulas. For example, the team presented in Table 1 is not of type $=(x_0, x_2)$ as $s_1(x_0) = s_2(x_0)$ and $s_1(x_2) \neq s_2(x_2)$. However, this team can be described by $=(x_0, x_2) \vee =(x_0, x_2)$ when it is considered as $\{s_0, s_1, s_3, s_4\} \cup \{s_2\}$.

Note that any Dependence Logic formula can be transformed into negation normal form (NNF), i.e. a form where negation is only applied to atoms and the only allowed Boolean operators are conjunction and disjunction, using the following rewriting rules for FOL [5]:

$$\phi \Rightarrow \psi \mapsto \neg\phi \vee \psi$$

$$\neg(\phi \vee \psi) \mapsto \neg\phi \wedge \neg\psi$$
$$\neg\neg\phi \mapsto \phi$$
$$\neg\exists x_j \phi \mapsto \forall x_j \neg\phi$$
$$\neg\forall x_j \phi \mapsto \exists x_j \neg\phi$$

3 Discussion and Modifications of the quantifiers of Dependence Logic

The meaning of existential and universal quantifiers in Dependence Logic is derived from the supplement and duplicate respectively (see definitions 2 and 3).

To decide if a team X is of type $\phi := \exists x_n \phi_1$, we need to verify whether there exists a supplement team $X' = X(F/x_n)$ for some F such that $(\phi, X(F/x_n), 1)$ (E11). In other words, the starting point for this consideration is a team without the variable x_n. A function F is then applied to find such values of x_n, to create a certain X' which satisfies the formula.

When it comes to the universal quantifier, we have that $(\forall x_n \phi, X, 1) \in \mathcal{T}$ iff $(\phi, X(M/x_n), 1) \in \mathcal{T}$ [6]. Again, the creation of team $X' = X(M/x_n)$ can be intuitively interpreted as starting from a team without the variable x_n and duplicating it enough times to assign every value from the domain to x_n in one of the duplicates. Then it comes down to verifying whether the formula ϕ describes the type of X' The meaning of the falsity of the formula with universal quantifier is $(\forall x_n \phi, X, 0) \in \mathcal{T}$ iff $(\phi, X(F/x_n), 0) \in \mathcal{T}$ for some $F : X \to M$ is intuitively analogical to $\neg\exists x \neg\phi(x)$ in FOL.

As a consequence of the above, when considering a formula without free variables, one must start from an empty set an successively build an X' team by supplementing and duplicating X according to the quantifiers in the formula. This approach has a limitation connected to its application to real world data - even when we consider a different domain for every variable, it may be the case that some combinations of the values of different variables will never coexist in one assignment, yet they have to be considered when evaluating formulas in Dependence Logic. As a consequence, the formula can evaluate to $False$ in a team generated in the above described way, even though it describe the type of the originally considered team. While this property is vital when using Dependence Logic to describe properties of abstract objects (see the Examples section in [6]), it becomes inconvenient when working with datasets, which aim to describe observable phenomena.

To adapt the Dependence Logic to the purpose of verifying the formulas in datasets, we propose a different meaning of the quantifiers. The purpose of the change is to replace the abstract operations of supplementing and duplicating with a more practical approach of considering subsets of assignments from a team.

$(\exists x_n \phi, X, 1) \in \mathcal{T}$ iff for some team $X' \in X$ and a value $a \in dom(x_n)$ such that for every $s \in X'$: $s(x_n) = a$ we have $(\phi, X', 1) \in \mathcal{T}$ (F11)

$(\forall x_n \phi, X, 1) \in \mathcal{T}$ iff for all values $a_1, \ldots, a_k \in dom(x_n)$ and teams $(X_1, \ldots, X_k) \subset X$ such that for all $s \in X_i$: $s(x_n) = a_i$, we have $(\phi, X_i, 1) \in \mathcal{T}$ (F12)

This approach allows us to remain in the domain of *existing* assignments and restricts the creation of new (and not necessarily plausible) ones. It allows for staying in the realm of reality and analysing dependencies within the limited data.

As a side note, the same idea is expressed by the *generalized assignment model* [1] - this allows for restricting the model to contain only some available variable assignments, thus expressing the possible coexistence of variable values.

To conclude, what is changed between the original Dependence Logic definition and the practical approach described in this paper is the method of generating the team. Further in this paper the above defined meaning of quantifiers is used. The definition of truth is modified to substitute them for E11 and E12.

It is possible to use the model-checking technique explained later according to the original Dependence Logic definition, using the concepts of supplement and duplicate, if the domain of the variables and the team size are finite. Then the dataset used must be constructed according to the quantifiers in the formula.

Given the original team X and a quantified variable x_n, for both universal and existential quantifiers, the final dataset X' will be: $X' = X(M/x_n)$.

In the case of the universal quantifier it is easy to see the correspondence to E11.

For the existential quantifier, according to the definition 2 it is necessary to find a function F which maps an x_n value to each assignment. This function however is unknown. This problem is handled by creating a duplicate team, so one containing all possible x_n assignments. This is possible as a resulting team is later divided into subsets in the process of defining the $\mathcal{D}$-Descriptive formula (see A2) and the truth valuation is preserved.

4 Model-checking a Dependence Logic formula

As previously described, in contrast to the First Order Logic formulas, the Dependence Logic formulas ($\mathcal{D}$-formulas) can only be evaluated on sets of assignments. This makes the model-checking process more complex. The first issue is how to evaluate an atomic $\mathcal{D}$-formula in a set of assignments. The second issue is how to find the subsets of assignments of X where the atomic $\mathcal{D}$-formula needs to be evaluated, because it depends on the structure of $\mathcal{D}$-formula.

To address the first problem, let us return to the definition of dependence. An atomic Dependence Logic formula says that a dependent variable is determined by the value of the dependence arguments. In other words, a set of assignments will be of the type of a given Dependence Logic formula, if for all the assignments with the same values for the dependence arguments, there is a single value for the dependent variable. An algorithm for checking this is proposed in Section 5.

For a better understanding of the second problem, recall the interpretation of a disjunction as finding subsets satisfying disjuncts of the $\mathcal{D}$-formula in the dataset.

Let us look at an example. Given the formula $\phi \coloneqq =(x_1, x_3) \vee =(x_2, x_3)$, each assignment needs be described by at least one of the two types in ϕ, thus if we split the team X in two subsets, one with the assignments that satisfy the first formula, X_1, and the other with the assignments that satisfy the second, X_2, we have that $(\phi, X, 1) \in \mathcal{T}$ iff $(=(x_1, x_3), X_1, 1) \in \mathcal{T}$ and $(=(x_2, x_3), X_2, 1) \in \mathcal{T}$, where $X_1, X_2 \subseteq X$ and $X_1 \cup X_2 = X$ (E7).

However, it is not possible to know how to find the subsets X_1 and X_2 *a priori*. A possible solution would be to check the formula against all the possible combinations of subsets, but this is impossible in practice, as with an increasing complexity of the formula, the amount of possible combinations will grow too fast. To overcome this challenge we propose a method of storing and aggregating the information from each assignment in a *descriptive formula* and encoding the meaning of atomic Dependence Logic formulas with the use of *mapping functions*.

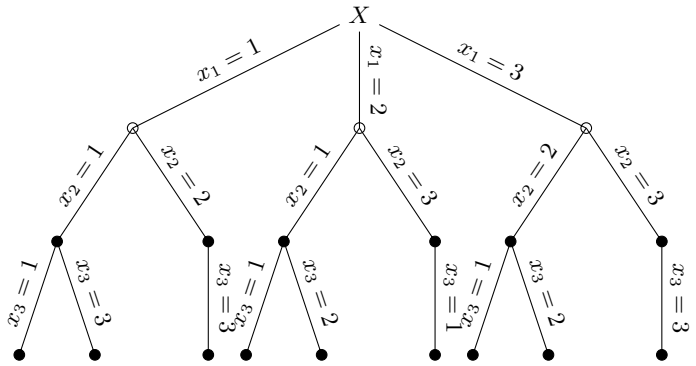
Consider a Dependence Logic formula $\forall x_1 \forall x_2 \exists x_3 (=(x_2, x_3))$ and a team presented in Table 2. Firstly, the team is divided into subsets according to the values of the variables under subsequent quantifiers, creating a tree. At a leaf node, the values (assigned throughout the path from the root to the leaf node) of dependence arguments and the dependent variable is saved, defining the mapping function, which captures the information about values of the dependent variables for sets of dependence arguments. The mapping function describes the relation expressed by the atomic Dependence Logic formula in a form $f(x_2) = x_3$. At this stage, the function is derived locally in the leaves. Note that the subsets of X at leaf nodes can have one or multiple assignments - the tree branches according to quantified variables, so their values at leaves are unique, while the values of free variables remain undetermined.

Then, the individual mappings from the leaf nodes are aggregated according to the quantifier which created the split ($\wedge$ for $\forall$ and $\vee$ for $\exists$). In this way, the atomic Dependence Logic formula is interpreted in a team in a form of an equivalent propositional logic formula, later called $\mathcal{D}-$descriptive formula.

The tree interpretation is presented in Figure 1. Skipping ahead, for the purpose of finishing this example, we can conclude that it results in the descriptive formula: $((f(1) = 1 \vee f(1) = 3) \wedge f(2) = 3) \wedge ((f(1) = 1 \vee f(1) = 2) \wedge f(3) = 1) \wedge ((f(2) = 1 \vee f(2) = 2) \wedge f(3) = 3)$. The resolution of the formula results in $False$, because of the definition of the mapping function (see Section 4.1).

The analysis of this example shows the complex nature of the Dependence Logic formulas' evaluation and illustrates its expressive power. Simply backtrack-

Fig. 1: A tree interpretation of the formula ϕ in team X



x_1	x_2	x_3
1	1	1
1	1	3
1	2	3
2	1	1
2	1	2
2	3	1
3	2	1
3	2	2
3	3	3

Table 2: Team X

ing the valuations of the leaf nodes is not sufficient to assess the whole formula - existence of the consistent mapping function across the whole team must be verified instead.

4.1 Mapping function and $\mathcal{D}$-Descriptive Formula ϕ^{des}

Let X be a team and ϕ a dependence formula $=(x_1, x_2, \ldots, x_{n-1}, x_n)$. To be able to conclude that $(\phi, X, 1) \in \mathcal{T}$, we have to demonstrate that for all $s, s' \in X$ such that $t_1^M\langle s\rangle = t_1^M\langle s'\rangle, \ldots, t_{n-1}^M\langle s\rangle = t_{n-1}^M\langle s'\rangle$, we have $t_n^M\langle s\rangle = t_n^M\langle s'\rangle$ (E3). To achieve this we will show that it is possible to construct a global mapping function, such that $f : \{x_1, \ldots, x_{n-1}\} \to x_n$ for every atomic $\mathcal{D}$-formula in ϕ. For convenience, *a pair* is defined as a tuple of the mapping function arguments and a value: $((x_1, \ldots, x_{n-1}), x_n)$, where $x_n = f(x_1, \ldots, x_{n-1})$. The function is understood as a list of such pairs. One mapping function is created for every atomic $\mathcal{D}$-formula in ϕ. The functions are evaluated independently of each other.

The $\mathcal{D}$-descriptive formula, as previously mentioned, has the purpose of aggregating information about possible mapping function assignments (values of the dependent variable) that would satisfy a given dependence logic formula in each subset of the team considered separately. Then, according to the structure of the quantifiers, all the partial definitions of the mapping function will be connected with the logical operators $\wedge$ and $\vee$ following rules described later.

Thus, the $\mathcal{D}$-descriptive formula is an interpretation of the $\mathcal{D}$-formula on team X, where X is the team considered in a given node. It is worth highlighting that any change in the team will entail a change in the $\mathcal{D}$-descriptive formula.

Below a technique of defining the $\mathcal{D}$-descriptive formula ϕ^{des} is introduced.

Let ϕ be a dependence logic formula in a negation normal form and X a non-empty finite team. Then ϕ^{des} is defined inductively as follows:

$$\text{If } \phi \coloneqq \forall x_i \psi, \qquad \text{then } \phi_X^{des} \equiv \psi^{des}_{\{X_1, x_i = a_1\}} \wedge \cdots \wedge \psi^{des}_{\{X_n, x_i = a_n\}} \quad \text{(A1)}$$

$$\text{If } \phi \coloneqq \exists x_i \psi, \qquad \text{then } \phi_X^{des} \equiv \psi^{des}_{\{X_1, x_i = a_1\}} \vee \cdots \vee \psi^{des}_{\{X_n, x_i = a_n\}} \quad \text{(A2)}$$

$$\text{If } \phi \coloneqq \psi \wedge \xi, \qquad \text{then } \phi_X^{des} \equiv \psi_X^{des} \wedge \xi_X^{des} \tag{A3}$$

$$\text{If } \phi \coloneqq \psi \wedge \theta, \qquad \text{then } \phi_X^{des} \equiv \psi_X^{des} \wedge \theta_X \tag{A4}$$

$$\text{If } \phi \coloneqq =(x_1, \ldots, x_n), \qquad \text{then}$$

$$\phi_X^{des} \equiv f(a_{1_1}, \ldots, a_{n-1_1}) = a_1 \wedge \cdots \wedge f(a_{1_m}, \ldots, a_{n-1_m}) = a_m \tag{A5}$$

$$\text{If } \phi \coloneqq \neg =(x_1, \ldots, x_n), \qquad \text{then } \phi_X^{des} \equiv (\top or \bot) \text{ - see explanation below} \tag{A6}$$

Let X be a set of assignments, then by $\{X_i, x_j = a_k\}$ we denote a non-empty subset of X in which $x_j = a_k$. We assume that $X_1 \cup X_2 \cup \ldots \cup X_n = X$ and $a_1, \ldots, a_n$ is a list of all possible values of x_j. ψ and ξ are dependence logic formulas and θ is a first order logic formula. In A5 m is the number of assignments in the team X.

To expand on the meaning of A5, let us describe the process of interpreting a $\mathcal{D}$-formula. First, lists of mapping function pairs are created locally on the team subsets. Subsequently, the lists are combined together, which can result in a consistent global function or in a contradiction. It is important to note that this aggregation can only be done on the level of the whole team, after the lists of pairs are defined for all subsets of X. The valuation of a conjunction of functions with the same arguments and different values results in $\bot$, as such function cannot exist.

Let us further explain the implications of A6. According to the definition of truth and satisfaction $(\neg =(x_1, \ldots, x_n), X, 1) \equiv (=(x_1, \ldots, x_n), X, 0)$, and due to E4, $(=(x_1, \ldots, x_n), X, 0) \in \mathcal{T}$ iff $X = \emptyset$. Thus there are two possibilities:

$$\text{If } \phi \coloneqq \neg =(x_1, \ldots, x_n) \text{ and } X = \emptyset, \qquad \text{then } \phi_X^{des} \equiv \top \tag{A6a}$$

$$\text{If } \phi \coloneqq \neg =(x_1, \ldots, x_n) \text{ and } X \neq \emptyset, \qquad \text{then } \phi_X^{des} \equiv \bot \tag{A6b}$$

As described in [6], the sentences in Dependence Logic are equivalent to sentences in Second Order Logic. The $\mathcal{D}$-descriptive formula, expressed in propositional logic, can only be equivalent to the Dependence Logic formula in a limited scope of a given finite dataset. Then $(\phi, X, 1) \in \mathcal{T}$ if and only if $\mathcal{M} \models \phi_X^{des}$.

4.2 Evaluation of the $\mathcal{D}$-Descriptive Formula ϕ^{des}

When a dependence logic formula has subformulas, θ, without dependence atoms, these subformulas can be evaluated in the assignments one by one. For that reason, to find ϕ^{des}, it is necessary to evaluate all the first order logic subformulas.

Evaluation of the $\mathcal{D}$-Descriptive formula starts with evaluating the First Order Logic elements of it, since this can be done in each of the team subsets independently. With this procedure, the $\mathcal{D}$-descriptive formula can be represented as follows:

$$\text{If } \phi = \psi \wedge \theta \text{ and } (\theta, X, 1) \in \mathcal{T}, \qquad \text{then } \phi_X^{des} \equiv \psi_X^{des} \wedge \top \equiv \psi_X^{des} \tag{A4a}$$

$$\text{If } \phi = \psi \wedge \theta \text{ and } (\theta, X, 1) \notin \mathcal{T}, \qquad \text{then } \phi_X^{des} \equiv \psi_X^{des} \wedge \bot \equiv \bot \tag{A4b}$$

$$\text{If } \phi = \psi \vee \theta \text{ and } (\theta, X, 1) \in \mathcal{T}, \qquad \text{then } \phi_X^{des} \equiv \psi_X^{des} \vee \top \equiv \top \tag{A4c}$$

$$\text{If } \phi = \psi \vee \theta \text{ and } (\theta, X, 1) \notin \mathcal{T}, \qquad \text{then } \phi_X^{des} \equiv \psi_X^{des} \vee \bot \equiv \psi_X^{des} \tag{A4d}$$

$$\text{If } \phi = \psi \vee \theta, (\theta, Z, 1), (\theta, Y, 0) \in \mathcal{T} \text{ and } Y \cup Z = X, \qquad \text{then } \phi_X^{des} \equiv \psi_Y^{des} \tag{A4e}$$

The two first expressions, A4a and A4b, because they are conjunctions, need all the elements to be true, so in case $\mathcal{M} \not\models \theta$ it is not necessary to evaluate ψ as we can claim directly that $(\phi, X, 1) \notin \mathcal{T}$. In the other way, when $\mathcal{M} \models \theta$, we still need to evaluate $(\psi, X, 1) \in \mathcal{T}$.

The following three expressions are disjunctions, so according to E7, $(\phi, X, 1) \in \mathcal{T}$ iff $(\psi, Y, 1) \in \mathcal{T}$ and $(\theta, Z, 1) \in \mathcal{T}$ where $X = Y \cup Z$. Because θ is a First Order Logic formula, we can find the Z as the maximal subset of assignments that makes θ true. Thus if $Z = X$, $(\phi, X, 1) \in \mathcal{T}$, A4c, if $Z = \emptyset$, ψ has to be evaluated in X, A4d, and finally, if $Z \subset X$, then ψ has to be evaluated in Y, A4e.

After these evaluations, the $\mathcal{D}$-descriptive formula results in series of conjunctions and disjunctions of pairs of the mapping function. All the First Order Logic elements are evaluated. Subsequently, the formula is transformed to disjunctive normal form. Every disjunct is a conjunction of pairs of the mapping function.

$$\begin{aligned} \phi^{des} &= \phi_{conj_1} \vee \cdots \vee \phi_{conj_k} \text{ where} \\ \phi_{conj_i} &= ((a_{1_1}, \ldots, a_{n-1_1}), a_1) \wedge \cdots \wedge ((a_{1_m}, \ldots, a_{n-1_m}), a_m) \end{aligned} \tag{1}$$

Let us consider any conjunction, ϕ_{conj_i}. Analyzing all the pairs of the mapping function in the conjunction, we establish if there is a contradiction, i.e., if $x_1\langle s\rangle = x_1\langle s'\rangle$, $\ldots$, $x_{n-1}\langle s\rangle = x_{n-1}\langle s'\rangle$ and $x_n\langle s\rangle \neq x_n\langle s'\rangle$. In this case indeed $\phi_{conj_i} \equiv \bot$. There exists a global mapping function f (and thus $(\phi, X, 1) \in \mathcal{T}$) if and only if there exists at least one conjunction ϕ_{conj_i} with no contradiction between its elements.

Otherwise, when all the conjunctions ϕ_{conj_i} contain contradictions then $\phi^{des} = \bot \vee \cdots \vee \bot = \bot$. Thus, mapping function f does not exist and $(\phi, X, 1) \notin \mathcal{T}$.

Example 1 (Checking a $\mathcal{D}$-formula).

Taking as an example the formula $\phi := \forall x_0 \exists x_1 \forall x_2 \exists x_3 {=}(x_2, x_3)$ together with the team X shown in Table 3 it is possible to find ϕ^{des} applying iteratively A1 and A2 according to section 4.1:

$$\begin{aligned} \phi^{des} = \{[(\psi^{des}_{x_0=1,x_1=3,x_2=1,x_3=1} \vee \psi^{des}_{x_0=1,x_1=3,x_2=1,x_3=2}) \wedge (\psi^{des}_{x_0=1,x_1=3,x_2=2,x_3=2} \vee \\ \psi^{des}_{x_0=1,x_1=3,x_2=2,x_3=3}) \wedge \psi^{des}_{x_0=1,x_1=3,x_2=3,x_3=3}] \vee [\psi^{des}_{x_0=1,x_1=2,x_2=1,x_3=2} \wedge \\ \psi^{des}_{x_0=1,x_1=2,x_2=2,x_3=1}]\} \wedge \{[(\psi^{des}_{x_0=2,x_1=2,x_2=1,x_3=1} \vee \psi^{des}_{x_0=2,x_1=2,x_2=1,x_3=2} \wedge \\ \psi^{des}_{x_0=2,x_1=2,x_2=3,x_3=1}] \vee \left[\psi^{des}_{x_0=2,x_1=3,x_2=2,x_3=1} \wedge \psi^{des}_{x_0=2,x_1=3,x_2=3,x_3=3}\right]\} \end{aligned}$$

where $\psi := {=}(x_2, x_3)$, thus, $\psi_X^{des} \equiv f(x_2 = a_1) = (x_3 = b_1) \wedge \cdots \wedge f(x_2 = a_n) = (x_3 = b_n)$.[1]

[1] In this example ϕ does not have a first order logic subformula to evaluate so the material from section 4.2 is not applied.

	x_0	x_1	x_2	x_3	ϕ
s_1	1	3	1	1	$f(1) = 1$
s_2	1	3	1	2	$f(1) = 2$
s_3	1	3	2	2	$f(2) = 2$
s_4	1	3	2	3	$f(2) = 3$
s_5	1	3	3	3	$f(3) = 3$
s_6	1	2	1	2	$f(1) = 2$
s_7	1	2	2	1	$f(2) = 1$
s_8	2	2	1	1	$f(1) = 1$
s_9	2	2	1	2	$f(1) = 2$
s_{10}	2	2	3	1	$f(3) = 1$
s_{11}	2	3	2	1	$f(2) = 1$
s_{12}	2	3	3	3	$f(3) = 3$

Table 3: Team X and ψ^{des} for each subset

The last step is transforming the formula to the disjunctive normal form and evaluating it, this step is shown in detail in 6. In DNF it is easy to see that the formula holds for one of the disjunctions, namely $f(x_2 = 1) = 2 \wedge f(x_2 = 2) = 1 \wedge f(x_2 = 3) = 1$ (assignments s_4, s_5, s_7 and s_8) or $f(x_2 = 1) = 2 \wedge f(x_2 = 2) = 1 \wedge f(x_2 = 3) = 3$ (assignments s_4, s_5, s_9 and s_{10}).

5 Algorithm for checking atomic formula in a dataset

In this section we present a simple algorithm for checking whether a formula ϕ describes the type of a team X and exploring possible dependencies in the dataset [4].

The atomic Dependence Logic formula $=(x_1, \ldots, x_{n-1}, x_n)$ states that the value of a variable x_n in any assignment in the team X of type $=(x_1, \ldots, x_{n-1}, x_n)$ can be determined solely with the values of variables $x_1, x_2, \ldots, x_{n-1}$. Algorithm 1 checks for such relations. Further, all dependencies in the dataset can be explored. To achieve this, first one column is chosen to be the dependant variable. Then, all possible sets of possible dependence arguments of a desired cardinality are found. This results in a list of atomic formulas, which are subsequently checked with Algorithm 1.

6 Conclusions and further work

Dependence logic is a fascinating framework expanding First Order Logic to express deterministic relation between variables. Because of the evaluation on teams instead of single assignments, it is capable of conveying information about the structure of the set of assignments. However, for the same reason valuation of the formula is a greater algorithmic challenge. The complexity of interpretation of the $\mathcal{D}$-formula grows both with the size of domains of variables and the number of quantifiers in the formula.

Algorithm 1 Check if the atomic formula describes the type of team X

```
1: function CHECK DEPENDENCE(data, (x_1, ..., x_{n-1}), x_n)
2:     Find all possible combinations of values of x_1, ..., x_{n-1} (C)
3:     for each c in C do
4:         Find the list of unique values of x_n in the assignments
5:         if The length of the generated lists > 1 then
6:             return False
7:             break
8:     return True
```

This paper has investigated how the satisfaction of a complex $\mathcal{D}$-formula in a team X can be shown and it has established some steps to prove it. Then, the next step would be to program an engine capable of checking the satisfaction of these formulas given a dataset and explore possibilities of optimization, based on the construction of the descriptive formula. Another interesting direction is to analyze other frameworks describing dependence, such as *branching quantifier logic* [2] or *independence friendly logic* [3] and establish whether our results can be extended to these frameworks.

Appendix 1

ϕ^{des} in disjunctive normal form

$$\{[(f(x_2=1)=1 \vee f(x_2=1)=2) \wedge (f(x_2=2)=2 \vee f(x_2=2)=3) \wedge f(x_2=3)=3] \vee [f(x_2=1)=2 \wedge f(x_2=2)=1]\} \wedge \{[(f(x_2=1)=1 \vee f(x_2=1)=2) \wedge f(x_2=3)=1] \vee [f(x_2=2)=1 \wedge f(x_2=3)=3]\}$$

Redistributing the first conjunction:

$$\{(f(x_2=1)=1 \vee f(x_2=1)=2) \wedge (f(x_2=2)=2 \vee f(x_2=2)=3) \wedge f(x_2=3)=3 \wedge (f(x_2=1)=1 \vee f(x_2=1)=2) \wedge f(x_2=3)=1\} \vee \{(f(x_2=1)=1 \vee f(x_2=1)=2) \wedge (f(x_2=2)=2 \vee f(x_2=2)=3) \wedge f(x_2=3)=3 \wedge f(x_2=2)=1 \wedge f(x_2=3)=3\} \vee \{f(x_2=1)=2 \wedge f(x_2=2)=1 \wedge (f(x_2=1)=1 \vee f(x_2=1)=2) \wedge f(x_2=3)=1\} \vee \{f(x_2=1)=2 \wedge f(x_2=2)=1 \wedge f(x_2=2)=1 \wedge f(x_2=3)=3\}$$

Redistributing the interior disjunctions the formula is on DNF:

$$
\begin{aligned}
&(f(x_2 = 1) = 1 \wedge f(x_2 = 2) = 2 \wedge f(x_2 = 3) = 3 \wedge f(x_2 = 1) = 1 \wedge f(x_2 = 3) = 1) \vee \\
&(f(x_2 = 1) = 2 \wedge f(x_2 = 2) = 2 \wedge f(x_2 = 3) = 3 \wedge f(x_2 = 1) = 1 \wedge f(x_2 = 3) = 1) \vee \\
&(f(x_2 = 1) = 1 \wedge f(x_2 = 2) = 3 \wedge f(x_2 = 3) = 3 \wedge f(x_2 = 1) = 1 \wedge f(x_2 = 3) = 1) \vee \\
&(f(x_2 = 1) = 2 \wedge f(x_2 = 2) = 3 \wedge f(x_2 = 3) = 3 \wedge f(x_2 = 1) = 1 \wedge f(x_2 = 3) = 1) \vee \\
&(f(x_2 = 1) = 1 \wedge f(x_2 = 2) = 2 \wedge f(x_2 = 3) = 3 \wedge f(x_2 = 1) = 2 \wedge f(x_2 = 3) = 1) \vee \\
&(f(x_2 = 1) = 2 \wedge f(x_2 = 2) = 2 \wedge f(x_2 = 3) = 3 \wedge f(x_2 = 1) = 2 \wedge f(x_2 = 3) = 1) \vee \\
&(f(x_2 = 1) = 1 \wedge f(x_2 = 2) = 3 \wedge f(x_2 = 3) = 3 \wedge f(x_2 = 1) = 2 \wedge f(x_2 = 3) = 1) \vee \\
&(f(x_2 = 1) = 2 \wedge f(x_2 = 2) = 3 \wedge f(x_2 = 3) = 3 \wedge f(x_2 = 1) = 2 \wedge f(x_2 = 3) = 1) \vee \\
&(f(x_2 = 1) = 1 \wedge f(x_2 = 2) = 2 \wedge f(x_2 = 3) = 3 \wedge f(x_2 = 2) = 1 \wedge f(x_2 = 3) = 3) \vee \\
&(f(x_2 = 1) = 2 \wedge f(x_2 = 2) = 2 \wedge f(x_2 = 3) = 3 \wedge f(x_2 = 2) = 1 \wedge f(x_2 = 3) = 3) \vee \\
&(f(x_2 = 1) = 1 \wedge f(x_2 = 2) = 3 \wedge f(x_2 = 3) = 3 \wedge f(x_2 = 2) = 1 \wedge f(x_2 = 3) = 3) \vee \\
&(f(x_2 = 1) = 2 \wedge f(x_2 = 2) = 3 \wedge f(x_2 = 3) = 3 \wedge f(x_2 = 2) = 1 \wedge f(x_2 = 3) = 3) \vee \\
&(f(x_2 = 1) = 2 \wedge f(x_2 = 2) = 1 \wedge f(x_2 = 1) = 1 \wedge f(x_2 = 3) = 1) \vee \\
&(f(x_2 = 1) = 2 \wedge f(x_2 = 2) = 1 \wedge f(x_2 = 1) = 2 \wedge f(x_2 = 3) = 1) \vee \\
&(f(x_2 = 1) = 2 \wedge f(x_2 = 2) = 1 \wedge f(x_2 = 2) = 1 \wedge f(x_2 = 3) = 3)
\end{aligned}
$$

References

1. Andréka, H., Németi, I., van Benthem, J.: Modal languages and bounded fragments of predicate logic. Journal of Philosophical Logic **27**(3), 217–274 (1998). https://doi.org/10.1023/a:1004275029985
2. Henkin, L.: Some remarks on infinitely long formulas. Journal of Symbolic Logic **30**(1), 167–183 (1961)
3. Hintikka, J., Sandu, G.: Informational independence as a semantical phenomenon. In: Fenstad, J.E., Frolov, I.T., Hilpinen, R. (eds.) Logic, Methodology and Philosophy of Science VIII, Studies in Logic and the Foundations of Mathematics, vol. 126, pp. 571–589. Elsevier (1989)
4. Lewandowska, Iga Kalina; Seijas Vega, J.: Checking dependence logic atomic formula in a dataset. https://github.com/s212445/Checking-Dependence-Logic-Atomic-Formula-in-a-Dataset (2022)
5. Robinson, J., Voronkov, A.: Handbook of Automated Reasoning: Volume 1. MIT Press, Cambridge, MA, USA (2001)
6. Väänänen, J.: Dependence Logic: A New Approach to Independence Friendly Logic. London Mathematical Society Student Texts, Cambridge University Press (2007)