Complexity of Propositional Independence and Inclusion Logic

Miika Hannula\textsuperscript{1}, Juha Kontinen\textsuperscript{1}, Jonni Virtema\textsuperscript{2}, and Heribert Vollmer\textsuperscript{2}

\textsuperscript{1} University of Helsinki, Department of Mathematics and Statistics, Helsinki, Finland \{miika.hannula,juha.kontinen\}@helsinki.fi
\textsuperscript{2} Leibniz Universität Hannover, Institut für Theoretische Informatik, Fakultät für Elektrotechnik und Informatik, Germany jonni.virtema@uta.fi, vollmer@thi.uni-hannover.de

\textbf{Abstract.} We classify the computational complexity of the satisfiability, validity and model-checking problems for propositional independence and inclusion logic and their extensions by the classical negation.

\textbf{Keywords:} propositional logic, team semantics, dependence, independence, inclusion, satisfiability, validity, model-checking

\section{Introduction}

Dependence logic \cite{LS} is a new logical framework for formalising and studying various notions of dependence and independence that are important in many scientific disciplines such as experimental physics, social choice theory, computer science, and cryptography. Dependence logic extends first-order logic by dependence atoms

\begin{equation}
\text{dep}(x_1,\ldots,x_n,y)
\end{equation}

expressing that the value of the variable $y$ is functionally determined on the values of $x_1,\ldots,x_n$. Satisfaction for formulas of dependence logic is defined using sets of assignments (teams) and not in terms of single assignments as in first-order logic. Whereas dependence logic studies the notion of functional dependence, independence and inclusion logic (introduced in \cite{7} and \cite{6}, respectively) formalize the concepts of independence and inclusion. Independence logic (inclusion logic) is obtained from dependence logic by replacing dependence atoms by the so-called independence atoms $x \perp y z$ (inclusion atoms $x \subseteq y$). The intuitive meaning of the independence atom is that the variables of the tuples $x$ and $z$ are independent of each other for any fixed value of the variables in $y$, whereas the inclusion atom declares that all values of the tuple $x$ appear also as values of $y$. In database theory these atoms correspond to the so-called embedded multivalued dependencies and inclusion dependencies (see, e.g., \cite{S}). Independence atoms have also a close connection to conditional independence in statistics.

The topic of this article is propositional team semantics which has received relatively little attention so far. On the other hand, modal team semantics has
been studied actively. Since the propositional logics studied in the article are fragments of the corresponding modal logics, some upper bounds trivially transfer to the propositional setting. The study of propositional team semantics as a subject of independent interest was initiated after surprising connections between propositional team semantics and the so-called inquisitive semantics was discovered (see [20] for details). The first systematic study on the expressive power of propositional dependence logic and many of its variants is due to [20][21]. In the same works natural deduction type inference systems for these logics are also developed, whereas in [17] a complete Hilbert-style axiomatization for propositional dependence logic is presented.

The computational aspects of (first-order) dependence logic and its variants have been actively studied, and are now quite well understood. On the other hand, the complexity of the propositional versions of these logics have not been systematically studied except for [19] in which the validity problem of propositional dependence logic was shown to be NP-complete. In this article we study the complexity of satisfiability, validity and model-checking of propositional independence and inclusion logic and their extensions by the classical negation. The classical negation has turned out to be a very powerful connective in the settings of first-order and modal team semantics, see e.g., [11] and [12]. Our results (see Table 1) show that the same is true in the propositional setting. In particular, our main result shows that the satisfiability and validity problems of the extensions of propositional independence and inclusion logic by the classical negation are complete for alternating exponential time with polynomially many alternations (AEXPTIME(poly)).

2 Preliminaries

In this section we define the basic concepts and results relevant to team-based propositional logics. We assume that the reader is familiar with propositional logic.

2.1 Syntax and Semantics

Let $D$ be a finite, possibly empty, set of proposition symbols. A function $s : D \rightarrow \{0, 1\}$ is called an assignment. A set $X$ of assignments $s : D \rightarrow \{0, 1\}$ is called a team. The set $D$ is the domain of $X$. We denote by $2^D$ the set of all assignments $s : D \rightarrow \{0, 1\}$.

|   | SAT | VAL | MC |
|---|-----|-----|----|
| PL\[c\]| NP | in coNEXPTIME\[NP\] | NP |
| PL\[\subseteq\]| EXPTIME [10] | coNP | in P [9] |
| PL\[\perp, \sim\], PL\[\subseteq, \sim\]| AEXPTIME(poly) | AEXPTIME(poly) | PSPACE |

Table 1: Overview of the results (completeness results if not stated otherwise)
Let $\Phi$ be a set of proposition symbols. The syntax for propositional logic $\text{PL}(\Phi)$ is defined as follows.

$$\varphi ::= p \mid \neg p \mid (\varphi \land \varphi) \mid (\varphi \lor \varphi), \text{ where } p \in \Phi.$$  

We write $\text{Var}(\varphi)$ for the set of all proposition symbols that appear in $\varphi$. We denote by $\models_{\text{PL}}$ the ordinary satisfaction relation of propositional logic defined via assignments in the standard way. Next we give team semantics for propositional logic.

**Definition 1.** Let $\Phi$ be a set of atomic propositions and let $X$ be a team. The satisfaction relation $X \models \varphi$ is defined as follows.

- $X \models p \iff \forall s \in X: s(p) = 1$.
- $X \models \neg p \iff \forall s \in X: s(p) = 0$.
- $X \models (\varphi \land \psi) \iff X \models \varphi$ and $X \models \psi$.
- $X \models (\varphi \lor \psi) \iff \exists Y,Z \text{ such that } Y \cup Z = X$.

Note that in team semantics $\neg$ is not the classical negation (denoted by $\sim$ in this article) but a so-called dual negation that does not satisfy the law of excluded middle. Next proposition shows that the team semantics and the ordinary semantics for propositional logic defined via assignments coincide.

**Proposition 1 ([18]).** Let $\varphi$ be a formula of propositional logic and let $X$ be a propositional team. Then $X \models \varphi$ iff $\forall s \in X: s \models_{\text{PL}} \varphi$.

The syntax of propositional dependence logic $\text{PD}(\Phi)$ is obtained by extending the syntax of $\text{PL}(\Phi)$ by the rule

$$\varphi ::= \text{dep}(p_1, \ldots, p_n, q), \text{ where } p_1, \ldots, p_n, q \in \Phi.$$  

The semantics for the propositional dependence atoms are defined as follows:

$$X \models \text{dep}(p_1, \ldots, p_n, q) \iff \forall s, t \in X: s(p_1) = t(p_1), \ldots, s(p_n) = t(p_n) \text{ implies that } s(q) = t(q).$$

The next proposition is very useful when determining the complexity of PD, and it is proved analogously as for first-order dependence logic [18].

**Proposition 2 (Downwards closure).** Let $\varphi$ be a PD-formula and let $Y \subseteq X$ be propositional teams. Then $X \models \varphi$ implies $Y \models \varphi$.

In this article we study the variants of PD obtained by replacing dependence atoms in terms of the so-called independence or inclusion atoms: The syntax of propositional independence logic $\text{PL}[\bot_c](\Phi)$ is obtained by extending the syntax of $\text{PL}(\Phi)$ by the grammar rule

$$\varphi ::= q \bot_p r,$$
where \( p, q, \) and \( r \) are finite tuples of proposition variables (not necessarily of the same length). The syntax of *propositional inclusion logic* \( \text{PL} \subseteq \Phi \) is obtained by extending the syntax of \( \text{PL}(\Phi) \) by the grammar rule
\[
\varphi ::= p \subseteq q,
\]
where \( p \) and \( q \) are finite tuples of proposition variables with the same length. Satisfaction for these atoms is defined as follows. If \( p = (p_1, \ldots, p_n) \) and \( s \) is an assignment, we write \( s(p) \) for \((s(p_1), \ldots, s(p_n))\).

\[
X |\models q \perp p \iff \forall s, t \in X : s(p) = t(p) \quad \text{then there exists } u \in X : u(pq) = s(pq) \text{ and } u(r) = t(r).
\]

\[
X |\models p \subseteq q \iff \forall s \in X \exists t \in X : s(p) = t(q).
\]

It is easy to check that neither \( \text{PL}[\perp_c] \) nor \( \text{PL}[\subseteq] \) is a downward closed logic (cf. Proposition 2). However, analogously to first-order inclusion logic [6], the formulas of \( \text{PL}[\subseteq] \) have the following closure property.

**Proposition 3 (Closure under unions).** Let \( \varphi \in \text{PL}[\subseteq] \) and let \( X_i, \) for \( i \in I, \) be teams. Suppose that \( X_i |\models \varphi, \) for each \( i \in I. \) Then \( \bigcup_{i \in I} X_i \models \varphi. \)

We will also consider the extensions of \( \text{PL}, \text{PL}[\perp_c] \) and \( \text{PL}[\subseteq], \) by the classical negation \( \sim \) with the standard semantics:
\[
X |\models \sim \varphi \iff X \not|\models \varphi.
\]

These extensions are denoted by \( \text{PL}[\sim], \text{PL}[\perp_c, \sim] \) and \( \text{PL}[\subseteq, \sim], \) respectively.

### 2.2 Auxiliary operators

The following additional operators will be used in this paper:

\[
X |\models \varphi \odot \psi \iff X |\models \varphi \text{ or } X |\models \psi,
\]

\[
X |\models \varphi \odot \psi \iff \forall Y, Z \subseteq X : \text{if } Y \cup Z = X, \text{ then } Y |\models \varphi \text{ or } Z |\models \psi,
\]

\[
X |\models \varphi \circ \psi \iff \forall Y \subseteq X : \text{if } Y |\models \varphi, \text{ then } Y |\models \psi,
\]

\[
X |\models \max(x_1, \ldots, x_n) \iff \{(s(x_1), \ldots, s(x_n)) \mid s \in X\} = \{0, 1\}^n.
\]

If \( X |\models \max(x) \), we say that \( X \) is *maximal over* \( x. \) If tuples \( x \) and \( y \) are pairwise disjoint and \( X |\models \max(x) \wedge x \perp y, \) then we say that \( X \) is *maximal over* \( x \) *for all* \( y. \) For the proof of the following proposition, see Appendix.

**Proposition 4.** \( \text{dep}(\cdot), \odot, \circ, \sim, \) and \( \text{max}(\cdot) \) translate into \( \text{PL}[\sim] \) in polynomial time.
Table 2: Complexity of satisfiability, validity, and model checking of PL and PD. All results are completeness results.

|          | SAT | VAL | MC References |
|----------|-----|-----|---------------|
| PL       | NP  | coNP| NC            |
| PD       | NP  | NEXPTIME | NP |

2.3 Satisfiability, Validity, and Model Checking in Team Semantics

Next we define satisfiability and validity in the context of team semantics. Let $L$ be a logic with team semantics. A formula $\varphi \in L$ is **satisfiable**, if there exists a non-empty team $X$ such that $X \models \varphi$. A formula $\varphi \in L$ is **valid**, if $X \models \varphi$ holds for every non-empty team $X$ such that the proposition symbols that occur in $\varphi$ are in the domain of $X$.

Note that when the team is empty, satisfaction becomes easy to decide (see Proposition 6 in Appendix).

The satisfiability problem $\text{SAT}(L)$ and the validity problem $\text{VAL}(L)$ are then defined in the obvious manner: Given a formula $\varphi \in L$, decide whether the formula is satisfiable (valid, respectively). The variant of the model checking problem that we are concerned in this article is the following: Given a formula $\varphi \in L$ and a team $X$, decide whether $X \models \varphi$. See Table 2 for known complexity results on PL and PD.

3 Complexity of Satisfiability and Validity

In this section we consider the complexity of the satisfiability and validity problems for propositional independence logic and inclusion logic, and their extensions by the classical negation $\neg$.

3.1 The Logics $\text{PL}[\bot_c]$ and $\text{PL}[]$

We consider first the complexity of $\text{SAT}([\bot_c])$. The following simple lemma turns out to be very useful.

**Lemma 1.** Let $\varphi \in \text{PL}[\bot_c]$ and $X$ a team such that $X \models \varphi$. Then $\{s\} \models \varphi$, for all $s \in X$.  

**Proof.** The claim is proved using induction on the construction of $\varphi$. It is easy to check that a singleton team satisfies all independence atoms, and the cases corresponding to disjunction and conjunction are straightforward.

**Theorem 1.** $\text{SAT}([\bot_c])$ is complete for NP.

---

3 It is easy to show that all of the logics considered in this article have the so-called locality property, i.e., satisfaction of a formula depends only on the values of the proposition symbols that occur in the formula $\mathcal{G}$.
Proof. Note first that since SAT(PL) is NP-complete, it follows by Proposition 1 that SAT(PL[⊥c]) is NP-hard. For containment in NP, note that by Lemma 1 a formula ϕ ∈ PL[⊥c] is satisfiable iff it is satisfied by some singleton team {s}. It is immediate that for any s, {s} |= ϕ iff {s} |= ϕT, where ϕT ∈ PL is acquired from ϕ by replacing all independence atoms by (p ∨ ¬p). Thus it follows that ϕ is satisfiable iff ϕT is satisfiable. Therefore, the claim follows. ⊓⊔

Next we consider the complexity of VAL(PL[⊥c]).

Theorem 2. VAL(PL[⊥c]) is hard for NEXPTIME and is in coNEXPTIMENP.

Proof. Since the dependence atom dep(x, y) is equivalent to the independence atom y ⊥ x y and VAL(PD) is NEXPTIME-complete [19], hardness for NEXPTIME follows. We will show in Theorem 9 on p. 10 that the model checking problem for PL[⊥c] is complete for NP. It then follows that the complement of the problem VAL(PL[⊥c]) is in NEXPTIMENP: the question whether ϕ is in the complement of VAL(PL[⊥c]) can be decided by guessing a subset X of 2D, where D contains the set of proposition variables appearing in ϕ, and checking whether X ̸|= ϕ. Therefore VAL(PL[⊥c]) ∈ coNEXPTIMENP. ⊓⊔

Next we turn to propositional inclusion logic.

Theorem 3 ([10]). SAT(PL[⊆]) is complete for EXPTime.

We end this section by determining the complexity of VAL(PL[⊆]).

Theorem 4. VAL(PL[⊆]) is complete for coNP.

Proof. Recall that PL is a sub-logic of PL[⊆], and hence VAL(PL[⊆]) is hard for coNP. Therefore, it suffices to show VAL(PL[⊆]) ∈ coNP. It is easy to check that, by Proposition 3 a formula ϕ ∈ PL[⊆] is valid iff it is satisfied by all singleton teams {s}. Note also that, over a singleton team {s}, an inclusion atom (p1,...,pn) ⊆ (q1,...,qn) is equivalent to the PL-formula

$$\bigwedge_{1\leq i\leq n} p_i \leftrightarrow q_i.$$ 

Denote by ϕ∗ the PL-formula acquired by replacing all inclusion atoms in ϕ by their PL-translations. By the above, ϕ is valid iff ϕ∗ is valid. Since VAL(PL) is in coNP the claim follows. ⊓⊔

3.2 The Logics PL[⊥c, ∼] and PL[⊆, ∼]

Next we incorporate classical negation in our logics. The main result of this section is that the satisfiability and validity problems for PL[⊥c, ∼] and PL[⊆, ∼] are complete for AEXPTIME(poly). The upper bound follows by an exponential-time alternating algorithm where alternation is bounded by formula depth. For the lower bound we first relate AEXPTIME(poly) to polynomial-time alternating Turing machines that query to oracles obtained from a quantifier prefix of
polynomial length. We then show how to capture this characterization with our logics.

First we observe that the classical negation gives rise to polynomial-time reductions between the validity and the satisfiability problems. Hence, we restrict our attention to satisfiability hereafter.

**Proposition 5.** Let \( \varphi \in \text{PL}[\mathcal{C}, \sim] \) where \( \mathcal{C} \subseteq \{ \text{dep}(\cdot), \perp, \subseteq \} \). Then one can construct in polynomial time formulae \( \psi, \theta \in \text{PL}[\mathcal{C}, \sim] \) such that

1. \( \varphi \) is satisfiable iff \( \psi \) is valid, and
2. \( \varphi \) is valid iff \( \theta \) is satisfiable.

**Proof.** We define

\[
\psi := \max(\mathbf{x}) \to \left( (p \lor \neg p) \lor (\varphi \land \sim (p \land \neg p)) \right),
\]
\[
\theta := \max(\mathbf{x}) \land (\sim (p \land \neg p) \to \varphi),
\]

where \( \mathbf{x} \) lists \( \text{Var}(\varphi) \). Note that \( X \models (p \land \neg p) \) iff \( X \) is non-empty. It is straightforward to show that (i) and (ii) hold. Also by Proposition 4, \( \psi \) and \( \theta \) can be constructed in polynomial time from \( \varphi \). \( \square \)

First we show the upper bound for the satisfiability problem of propositional logic with the classical negation, and the independence and inclusion atoms.

**Theorem 5.** \( \text{SAT}(\text{PL}[[\perp, \subseteq, \sim]]) \in \text{AEXPTIME}(\text{poly}) \).

**Proof.** Let \( \varphi \in \text{PL}[[\perp, \subseteq, \sim]] \). First existentially guess a possibly exponential-size team \( T \) with domain \( \text{Var}(\varphi) \). Then implement Algorithm 1 (see Appendix) on \( \text{mc}(T, \varphi, 1) \). The result follows since this algorithm runs in polynomial time and its alternation is bounded by the size of \( \varphi \). \( \square \)

Let us then turn to the lower bound. We show that the satisfiability problems of \( \text{PL}[[\perp, \subseteq, \sim]] \) and \( \text{PL}[[\perp, \sim]] \) are both hard for \( \text{AEXPTIME}(\text{poly}) \). For this, we first relate \( \text{AEXPTIME}(\text{poly}) \) to polynomial-time oracle Turing machines. This approach is originally due to Orponen in [16], where the classes \( \Sigma_k^{\text{EXP}} \) and \( \Pi_k^{\text{EXP}} \) of the exponential-time hierarchy were characterized by polynomial-time constant-alternation oracle Turing machines that query to \( k \) oracles. Recall that the exponential-time hierarchy corresponds to the class of problems that can be recognized by an exponential-time alternating Turing machine with constantly many alternations. In the next theorem we generalize Orponen’s characterization to exponential-time alternating Turing machines with polynomially many alternations (i.e. the class \( \text{AEXPTIME}(\text{poly}) \)) by allowing queries to polynomially many oracles.

By \( (A_1, \ldots, A_k) \) we denote an efficient disjoint union of sets \( A_1, \ldots, A_k \), e.g. \( (A_1, \ldots, A_k) = \{(i, x) : x \in A_i, 1 \leq i \leq k\} \).

**Theorem 6.** A set \( A \) belongs to the class \( \text{AEXPTIME}(\text{poly}) \) iff there exist a polynomial \( f \) and a polynomial-time alternating oracle Turing machine \( M \) such that, for all \( x \),

\[
x \in A \text{ iff } Q_1 A_1 \ldots Q_f(x) A_f(x) (M \text{ accepts } x \text{ with oracles } (A_1, \ldots, A_f(x))),
\]
where \( n \) is the length of \( x \) and \( Q_1, \ldots, Q_{f(n)} \) alternate between \( \exists \) and \( \forall \), i.e. \( Q_{i+1} \in \{\forall, \exists\} \setminus \{Q_i\} \).

**Proof.** The proof is a straightforward generalization of the proof of Theorem 5.2 in [10] (see Appendix).

Using this theorem we now prove Theorem 7. For the quantification over oracles \( A_i \), we use repetitively \( \lor \) and \( \sim \). For simulating the computation of an alternating polynomial-time oracle Turing machine, we first quantify over polynomially many Boolean sequences of polynomial length and then simulate the computation of a deterministic polynomial-time Turing machine which queries to the quantified oracles.

**Theorem 7.** \( \text{SAT}(\text{PL}[\perp, \sim]) \) and \( \text{SAT}(\text{PL}[\subseteq, \sim]) \) are hard for \( \text{AEXPTIME}(\text{poly}) \).

**Proof.** Let \( A \in \text{AEXPTIME}(\text{poly}) \). From Theorem 6 we obtain a polynomial \( f \) and an alternating oracle Turing machine \( M \) with running time bounded by \( g \). By [3], the alternating machine can be replaced by a sequence of word quantifiers over a deterministic Turing machine. (Strictly speaking, [3] speaks only about a bounded number of alternations, but the generalization to the unbounded case is straightforward.) W.l.o.g. we may assume that each configuration of \( M \) has at most two configurations reachable in one step. It then follows by Theorem 6 that one can construct a polynomial-time deterministic oracle Turing machine \( M^* \) such that \( x \in A \) iff

\[
Q_1 A_1 \ldots Q_{f(n)} A_{f(n)} Q'_1 y_1 \ldots Q'_{g(n)} y_{g(n)} \quad (M^* \text{ accepts } (x, y_1, \ldots, y_{g(n)}))
\]

where \( Q_1, \ldots, Q_{f(n)} \) and \( Q'_1, \ldots, Q'_{g(n)} \) are alternating sequences of quantifiers \( \exists \) and \( \forall \), and each \( y_i \) is a \( g(n) \)-ary sequence of propositional variables where \( n \) is the length of \( x \). Note that \( M^* \) runs in polynomial time also with respect to \( n \). Using this characterization we now show how to reduce in polynomial time any \( x \) to a formula \( \varphi \) in \( \text{PL}[\perp, \sim] \) (or in \( \text{PL}[\subseteq, \sim] \)) such that \( x \in A \) iff \( \varphi \) is satisfiable. We construct \( \varphi \) inductively. As a first step, we let

\[
\varphi := \max(qr y) \land p_t \land \neg p_f \land \varphi_1
\]

where

- \( qr \) and \( r \) list propositional variables that are used for encoding oracles;
- \( y \) lists propositional variables that occur in \( y_1, \ldots, y_{g(n)} \) and in \( z_i \) that are used to simulate configurations of \( M^* \) (see phase (3) below);
- \( p_t \) and \( p_f \) are propositional variables that do not occur in \( qr y \).

1. **Quantification over oracles.** Next we show how to simulate quantification over oracles. W.l.o.g. we may assume that \( M^* \) queries binary strings that are of length \( h(n) \) for some polynomial \( h \). Let \( q \) be a sequence of length \( h(n) \) and \( r \) a sequence of length \( \log(f(n)) + 1 \). For \( i \in \mathbb{N} \), we let \( \text{bin}(i) \) denote a
binary representation of \( i \) (with an appropriate number of insignificant zeros).

For a string of bits \( b = b_1 \ldots b_k \) and a sequence \( s = (s_1, \ldots, s_k) \) of proposition symbols, we write \( s = b \) for \( \bigwedge_{i=1}^k s_i = b_i \) where \( s_i = 0 \) and \( s_i = 1 \) denote \( \neg s_i \) and \( s_i \), respectively. The idea is that, given a team \( X \) over \( qr \), an oracle \( A_i \), and a binary string \( a = a_1 \ldots a_{h(n)} \), the membership of \( a \) in \( A_i \) is expressed by \( X \models \neg(q = a \wedge r = \text{bin}(i)) \). Note that the latter indicates that there exists \( s \in X \) mapping \( q \mapsto a \) and \( r \mapsto \text{bin}(i) \). Following this idea we next show how to simulate quantification over oracles \( A_i \). We define \( \varphi_i \), for \( 1 \leq i \leq f(n) \), inductively from root to leaves. Depending on whether \( A_i \) is existentially or universally quantified, we let

\[
\exists: \varphi_i := r = \text{bin}(i) \lor (\alpha \land \varphi_{i+1}), \\
\forall: \varphi_i := \neg r = \text{bin}(i) \otimes (\neg \alpha \land \varphi_{i+1}),
\]

where \( \alpha \) is defined in \( \text{SAT}(\text{PL}[\bot, \neg]) \) and in \( \text{SAT}(\text{PL}[\subseteq, \neg]) \) respectively as follows:

\[
\alpha := \max(\mathbf{y}) \land \mathbf{y} \perp qr, \\
\alpha := \bigwedge_{i=1}^{\left|\mathbf{y}\right|} qr y_1 \ldots y_{i-1} p f \subseteq qr y_1 \ldots y_i \land \bigwedge_{i=1}^{\left|\mathbf{y}\right|} qr y_1 \ldots y_{i-1} p i \subseteq qr y_1 \ldots y_i.
\]

Let us explain the idea behind the definitions of \( \varphi_i \), first in the case of existential quantification. Assume that we consider a formula \( \varphi \) and a team \( X \) where

\[
X \models \alpha \land \varphi_i,
\]

and \( \{ s \in X \mid s(r) = \text{bin}(i) \} \) is maximal over \( qy \). Then by (2) we may choose two subsets \( Y, Z \subseteq X \), \( Y \cup Z = X \), where \( Y \models r = \text{bin}(i) \) and \( Z \models \alpha \land \varphi_{i+1} \). The idea is that \( Z \) must include all assignments \( s \in X \) where \( s(r) \neq \text{bin}(i) \), and it may exclude an arbitrary number of assignments \( s \in X \) where \( s(r) = \text{bin}(i) \). Hence since \( \{ s \in X \mid s(r) = \text{bin}(i) \} \) is maximal over \( qy \), the set \( \{ s(q) \mid s \in Z, s(r) = \text{bin}(i) \} \) can be chosen to be an arbitrary subset of \( \{0, 1\}^{\left|q\right|} \). The only restriction for this choice is that it must be uniform with respect to values of \( \mathbf{y} \), meaning that \( Z \) must remain maximal over \( \mathbf{y} \) for all \( qr \). This is ensured by requiring that \( Z \models \alpha \).

Universal quantification is simulated analogously. This time we range over all subsets \( Y, Z \subseteq X \) where \( Y \cup Z = X \). By (2) for all such \( Y \) and \( Z \), we have that \( Z \models \neg \alpha \land \varphi_{i+1} \) if \( Y \models \neg r = \text{bin}(i) \). Hence for all subsets \( Z \) having only \( s \in X \) with \( s(r) = \text{bin}(i) \) removed from it, \( Z \models \neg \alpha \land \varphi_{i+1} \). This means that such subsets \( Z \) satisfy \( \varphi_{i+1} \) whenever they are formed uniformly with respect to values of \( \mathbf{y} \). Analogously to the existential case, we now observe that this corresponds to universal quantification of \( A_i \).

(2) **Quantification over propositional variables.** Next we show how to simulate the quantifier block \( Q_i^1 y_1 \ldots Q_i^{h(n)} y_{g(n)} \exists \mathbf{z} \) where \( \mathbf{z} \) lists all propositional variables that occur in \( \mathbf{y} \) but not in any \( y_i \) (i.e. the remaining variables that
Theorem 9. Let $M$ be a deterministic Turing machine. Assume that the running time of $M$ is in $\text{AEXPTIME}(\text{poly})$. If $X$ is a set of states of $M$ such that $X$ is maximal over $y_i$ for all $qry_i...y_i-1$, then $X$ is in accepting state.

4 Complexity of Model Checking

In this section we consider the related model checking problems. We first focus on logics without classical negation.

Theorem 9. $\text{MC}(\text{PL}[\perp,\sim])$ is complete for $\text{NP}$. 
Proof. The upper bound follows since the model checking problem for modal independence logic is \(NP\)-complete \([11]\). Since dependence atoms can be expressed efficiently by independence atoms (see the proof of Theorem \([2]\), the lower bound follows from the \(NP\)-completeness of \(MC(PD)\) (see Table \([3]\)).

The following unpublished result was shown by Hella.

**Theorem 10** (\([9]\)). \(MC(PL[\subseteq])\) is in \(P\).

**Theorem 11.** \(MC(PL[\sim])\) is complete for \(PSPACE\).

Proof. For the upper bound note that Algorithm \([\text{1}]\) decides the problem in \(APTIME\) which is exactly \(PSPACE\) \([3]\). For the lower bound, we reduce from \(TQBF\) which is known to be \(PSPACE\)-complete. Let \(Q_1x_1 \ldots Q_nx_n\theta\) be a quantified boolean formula. Let \(r\) be a sequence of propositional variables such that its length is \(\log(n) + 1\), and let \(T := \{s_1, \ldots, s_n\}\) where \(s_i(r)\) writes \(i\) in binary. We define inductively top-dow a \(\varphi \in PL[\sim]\) such that

\[
Q_1x_1 \ldots Q_nx_n\theta \text{ is true iff } T |= \varphi. \tag{4}
\]

Let \(\varphi := \varphi_1\), and for \(1 \leq i \leq n\), depending on whether \(x_i\) is existentially or universally quantified we let

\[
\exists: \varphi_i := r = \text{bin}(i) \lor \varphi_{i+1},
\]

\[
\forall: \varphi_i := \sim r = \text{bin}(i) \otimes \varphi_{i+1}.
\]

Finally, we let \(\varphi_{n+1} := \theta(\psi_i/x_i)\) where \(\psi_i := \sim r = \text{bin}(i)\), the meaning of which is that \(s_i\) exists in the team. Since the above simulation of universal and existential quantification reminds that of oracles in the proof of Theorem \([\text{7}]\) we notice that \(\varphi\) holds. Also \(T\) and \(\varphi\) can be constructed in polynomial time, and hence we obtain the result. \(\square\)

Since Algorithm \([\text{1}]\) can also be applied to independence and inclusion atoms, we obtain the following corollary.

**Corollary 1.** \(MC(PL[\perp, \sim])\) and \(MC(PL[\subseteq, \sim])\) are complete for \(PSPACE\).

5 Conclusion

As it is apparent from the summary of our new contributions depicted in Table \([\text{1}]\) a couple open questions still remain:

- Is \(VAL(PL[\perp])\) complete for \(coNEXPTIME^{NP}\)?
- Is \(MC(PL[\subseteq])\) complete for \(P\)?
- Are \(SAT(PL[\sim])\) and \(VAL(PL[\sim])\) complete for \(AEXPTIME(poly)\)? For this, it would suffice to find a polynomial-time translation of independence or inclusion atom in \(PL[\sim]\).
Acknowledgement

The first two authors were supported by grants 264917 and 275241 of the Academy of Finland. The third author was supported by the grant 266260 of the Academy of Finland and by the Jenny and Antti Wihuri Foundation.

References

1. Samson Abramsky and Jouko Väänänen. From IF to BI. Synthese, 167:207–230, 2009. 10.1007/s11229-008-9415-6.
2. Sam Buss. The boolean formula value problem is in ALOGLTIME. In Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing, STOC ’87, pages 123–131, New York, NY, USA, 1987. ACM.
3. Ashok K. Chandra, Dexter C. Kozen, and Larry J. Stockmeyer. Alternation. J. ACM, 28(1):114–133, January 1981.
4. Stephen A. Cook. The complexity of theorem-proving procedures. In Proceedings of the third annual ACM symposium on Theory of computing, STOC ’71, pages 151–158, New York, NY, USA, 1971. ACM.
5. Johannes Ebbing and Peter Lohmann. Complexity of model checking for modal dependence logic. In Mária Bieliková, Gerhard Friedrich, Georg Gottlob, Stefan Katzenbeisser, and György Turán, editors, SOFSEM 2012: Theory and Practice of Computer Science, volume 7147 of Lecture Notes in Computer Science, pages 226–237. Springer Berlin / Heidelberg, 2012.
6. Pietro Galliani. Inclusion and exclusion dependencies in team semantics: On some logics of imperfect information. Annals of Pure and Applied Logic, 163(1):68 – 84, 2012.
7. Erich Grädel and Jouko Väänänen. Dependence and independence. Studia Logica, 101(2):399–410, 2013.
8. Miika Hannula and Juha Kontinen. A finite axiomatization of conditional independence and inclusion dependencies. In Christoph Beierle and Carlo Meghini, editors, Foundations of Information and Knowledge Systems - 8th International Symposium, FoIKS 2014, Bordeaux, France, March 3-7, 2014. Proceedings, volume 8367 of Lecture Notes in Computer Science, pages 211–229. Springer, 2014.
9. Lauri Hella. Private communication.
10. Lauri Hella, Antti Kuusisto, Arne Meier, and Heribert Vollmer. Modal inclusion logic: Being lax is simpler than being strict. In preparation.
11. Juha Kontinen, Julian-Steffen Müller, Henning Schnoor, and Heribert Vollmer. A van Benthem theorem for modal team semantics. CoRR, abs/1410.6648, 2014.
12. Juha Kontinen and Ville Nurmi. Team logic and second-order logic. Fundam. Inform., 106(2-4):259–272, 2011.
13. Leonid A. Levin. Universal search problems. Problems of Information Transmission, 9(3), 1973.
14. Peter Lohmann and Heribert Vollmer. Complexity results for modal dependence logic. Studia Logica, 101(2):343–366, 2013.
15. Julian-Steffen Müller. Satisfiability and model checking in team based logics. PhD Thesis, Leibniz Universität Hannover, Cuvillier Verlag Göttingen, 2014.
16. Pekka Orponen. Complexity classes of alternating machines with oracles. In Automata, Languages and Programming, 10th Colloquium, Barcelona, Spain, July 18-22, 1983, Proceedings, pages 573–584, 1983.
17. Katsuhiko Sano and Jonni Virtema. Axiomatizing propositional dependence logics. 
   *CoRR*, abs/1410.5038, 2014.
18. Jouko Väänänen. *Dependence Logic*. Cambridge University Press, 2007.
19. Jonni Virtema. Complexity of validity for propositional dependence logics. In Adriano Peron and Carla Piazza, editors, *Proceedings Fifth International Symposium on Games, Automata, Logics and Formal Verification, GandALF 2014, Verona, Italy, September 10-12, 2014.*, volume 161 of *EPTCS*, pages 18–31, 2014.
20. Fan Yang. *On Extensions and Variants of Dependence Logic*. PhD thesis, University of Helsinki, 2014.
21. Fan Yang and Jouko Väänänen. Propositional logics of dependence and independence, part I. *CoRR*, abs/1412.7998, 2014.
Appendix

Proof (Theorem 6). If-part. Let $f$ be a polynomial and $M$ a polynomial-time alternating oracle Turing machine. We describe the behaviour of an alternating Turing machine $M'$ such that for all $x$,

$$M' \text{ accepts } x \iff Q_1 A_1 \ldots Q_{f(n)} A_{f(n)}(M \text{ accepts } x \text{ with oracle } (A_1, \ldots, A_{f(n)})).$$

First $M'$ simulates quantification over $Q_1 A_1 \ldots Q_{f(n)} A_{f(n)}$ in steps 1, \ldots, $f(n)$ as follows: at step $k$, if $Q_k = \exists$ (or $Q_k = \forall$), then existentially (universally) guess $A_k$ that consists of $i$ with $|i| \leq f(n)$. Then $M'$ evaluates the computation tree $M$ associates with $x$ and $(A_1, \ldots, A_{f(n)})$, replacing queries to $A_k$ with investigations of the corresponding guess. We notice that $M'$ constructed in this way satisfies [5], alternates $f(n)$ many times, and runs in time $2^{O(n)}$ for some polynomial $h$.

Only-if part. Let $M$ be an alternating exponential-time Turing machine with polynomially many alternations. W.l.o.g. there are polynomials $f$ and $g$ such that $M$ runs in time at least $n$ and at most $2^{O(n)}$ and has $g(n)$ many alternations.

Let $\#$ be a symbol not in the alphabet and denote $2^{|f(n)|} - 1$ by $m$. Each configuration of $M$ can be represented as a string

$$\alpha = uqv\# \ldots \# |\alpha| = m,$$

meaning that $M$ is in state $q$, has string $uv$ on its tape, and reads the first symbol of string $v$. The symbol $\#$ is only used to pad configurations to the same length. A computation of $M$ over $x$ may be represented as a sequence of configurations $s_0, s_1, \ldots, s_m$ with $s_0 = q_0 x\# \ldots \#$, $s_m = uqv\# \ldots \#$ for some final state $q$, and for each $i \leq m - 1$, either $s_i$ can be reached from $s_{i+1}$ with one step or $s_{i+1}$ is the unique configuration such that

$$\alpha = uqv\# \ldots \# |\alpha| = m.$$ 

Each oracle $A_k$ can encode a computation sequence $\alpha_0^k, \alpha_1^k, \ldots, \alpha^k_m$ with triples $(i,j,k)$ where $|i|, |j| \leq f(n)$ and $\alpha^k_{i,j}$ is the $j$th symbol of configuration $\alpha^k_i$. Determining whether $k, i, j$ generate a unique $\alpha^k_{i,j}$ can be done with a bounded number of queries since there are only finitely many alphabet and state symbols in $M$.

Using this encoding we next describe the behaviour of a polynomial-time alternating oracle Turing machine $M'$ over input $x$ of length $n$. W.l.o.g. we may assume that $M$ starts in existential state. Hence let $Q_1, \ldots, Q_{g(n)}$ be an alternating quantifier sequence that starts with $\exists$ (i.e. $Q_1 = \exists$ and $Q_{i+1} \in \{\forall, \exists\} \setminus \{Q_i\}$). We describe $M'$ in steps 1, \ldots, $g(n)$. Before each step $k$ assume that some $v$ with $|v| \leq f(n)$ (i.e. the last alternation point) is given. Assume also first that $Q_k = \exists$. Then $M$ proceeds at step $k$, for $k > 1$, according to the following instructions (step $k$, for $k = 1$, is obtained by setting $v \mapsto 0$ and $\alpha^{k-1}_{v,j} \mapsto \text{pr}_j(0)$ for $\alpha_0 = q_0 x\# \ldots \#$).

1. Universally check that for all $i,j$ with $|i|, |j| \leq f(n)$, $v \leq i$, it holds that $\alpha^{k-1}_{v,j} = \alpha_{v,j}^k$, and $\alpha^k_{i,j-1}, \alpha^k_{i,j}, \alpha^k_{i,j+1}, \alpha^k_{i,j+2}$ correctly determine $\alpha^k_{i,j+1,j}$.

2. Existentially choose $u$ with $|u| \leq f(n)$, $v < u$, and then universally check that for all $i,j$ with $|i|, |j| \leq f(n)$, $v < i < u$, it holds that $\alpha^k_{i,j}$ is not a universal state.
Algorithm 1 APTIME algorithm for MC(PL[⊥, ⊑, ¬])

1: function mc(T, ϕ, I)
2:     if ϕ = ψ1 ∧ ψ2 then
3:         if I = 1 then
4:             universally choose i ∈ {1, 2}
5:             return mc(T, ψi, I)
6:         else if I = 0 then
7:             existentially choose i ∈ {1, 2}
8:             return mc(T, ψi, I)
9:     else if ϕ = ψ1 ∨ ψ2 then
10:        if I = 1 then
11:            existentially choose T1 ∪ T2 = T
12:            universally choose i ∈ {1, 2}
13:            return mc(Ti, ψi, I)
14:        else if I = 0 then
15:            universally choose T1 ∪ T2 = T
16:            existentially choose i ∈ {1, 2}
17:            return mc(Ti, ψi, I)
18:     else if ϕ = ¬ψ then
19:        if I = 1 then
20:            return mc(T, ψ, 0)
21:        else if I = 0 then
22:            return mc(T, ψ, 1)
23:     else if ϕ = p (ϕ = ¬p) then
24:         1 ← x
25:         for s ∈ T do
26:             if s(p) = 0 (s(p) = 1) then
27:                 0 ← x
28:             if x = I = 1 or x = I = 0 then
29:                 return true
30:         else
31:             return false
32:     else if ϕ = p ⊑ q then
33:         1 ← x
34:         for s ∈ T do
35:             0 ← y
36:             for s' ∈ T do
37:                 if s(p) = s'(q) then
38:                     1 ← y
39:                 if y = 0 then
40:                     0 ← x
41:                 if x = I = 1 or x = I = 0 then
42:                     return true
43:             else
44:                 return false
45:     else if ϕ = q ⊥ r then
46:         1 ← x
47:         for s, s' ∈ T with s(p) = s'(p) do
48:             0 ← y
49:             for s'' ∈ T do
50:                 if s(p) = s''(p), s(q) = s''(q), s'(r) = s''(r) then
51:                     1 ← y
52:                 if y = 0 then
53:                     0 ← x
54:                 if x = I = 1 or x = I = 0 then
55:                     return true
56:             else
57:                 return false
It is easy to check that \( \emptyset \mid \).

**Proof.** Define a function \( \text{dep} \) as the depth. This concludes the proof. \( \odot \odot \)

It is straightforward to check that all the translations can be done in polynomial time and that

\[ M \text{ accepts } x \iff Q_1A_1 \ldots Q_g(n)A_g(n)(M' \text{ accepts } x \text{ with oracle } (A_1, \ldots, A_g(n))). \]

\( \square \)

**Proposition 6.** Checking whether \( \emptyset \models \varphi \), for \( \varphi \in \text{PL}[\bot, \subseteq, \sim] \), can be done in \( \mathbb{P} \). Furthermore, \( \emptyset \models \varphi \) for all \( \varphi \in \text{PL}[\bot, \subseteq] \).

**Proof.** Define a function \( \pi : \text{PL}[\bot, \subseteq, \sim] \to \{0, 1\} \) recursively as follows. Note that addition is mod 2.

- If \( \varphi \in \{p, \lnot p, q \bot p, r, p \subseteq q\} \), then \( \pi(\varphi) = 1 \).
- If \( \varphi = \psi_0 \land \psi_1 \), then \( \pi(\varphi) = \pi(\psi_0) \cdot \pi(\psi_1) \).
- If \( \varphi = \psi_0 \lor \psi_1 \), then \( \pi(\varphi) = \pi(\psi_0) \cdot \pi(\psi_1) + 1 \).
- If \( \varphi = \lnot \psi \), then \( \pi(\varphi) = \pi(\psi) + 1 \).

It is easy to check that \( \emptyset \models \varphi \) iff \( \pi(\varphi) = 1 \). Since \( \pi(\varphi) \) can be computed in \( \mathbb{P} \), the claim follows.

**Proof (Proposition 6).**

We present the following translations of which item 3 is due to [15] and item 4 uses the idea of [11].

1. The connective \( \otimes \) is actually the dual of \( \lor \), and hence \( \varphi \otimes \psi \) can be written as \( \lnot (\lnot \varphi \lor \lnot \psi) \).
2. Intuitionistic disjunction \( \varphi \otimes \psi \) can be written as \( \lnot (\lnot \varphi \land \lnot \psi) \).
3. Intuitionistic implication \( \varphi \rightarrow \psi \) can be expressed as \( (\lnot \varphi \lor \psi) \otimes (p \lor \lnot p) \).
4. First note that \( \text{dep}(x) \) can be written as \( x \otimes \lnot x \). Using this we can write \( \text{dep}(x_1, \ldots, x_n, y) = \bigwedge_{i=1}^n \text{dep}(x_i) \rightarrow \text{dep}(y) \).
5. We show that \( \max(x_1, \ldots, x_n) \) is equivalent to \( \lnot \bigvee_{i=1}^n \text{dep}(x_i) \). Assume first that \( X \models \bigvee_{i=1}^n \text{dep}(x_i) \), we show that \( X \not\models \max(x_1, \ldots, x_n) \). By the assumption, we find \( Y_1, \ldots, Y_n \subseteq X, \bigcup_{i=1}^n Y_i = X \), such that \( Y_i \models \lnot (x_i) \). Now for all \( i \) there exists a \( b_i \in \{0, 1\} \) such that if \( Y_i \not\models \emptyset \), then for all \( s \in Y_i, s(x_i) = b_i \). Since the assignment \( x_i \rightarrow b_i \) is not in \( X \), we obtain that \( X \not\models \max(x_1, \ldots, x_n) \).

Assume then that \( X \not\models \max(x_1, \ldots, x_n) \), we show that \( X \models \bigvee_{i=1}^n \text{dep}(x_i) \). By the assumption there exists a boolean sequence \( (b_1, \ldots, b_n) \) such that for no \( s \in X \) we have \( s(x_i) = b_i \) for all \( i = 1, \ldots, n \). Let \( Y_i := \{s \in X : s(x_i) \neq b_i\} \). Since then \( X = \bigcup_{i=1}^n Y_i \) and \( Y_i \models \lnot (x_i) \), we obtain that \( X \models \bigvee_{i=1}^n \text{dep}(x_i) \).

It is straightforward to check that all the translations can be done in polynomial time. This concludes the proof. \( \square \)