Logic and Computational Complexity for Boolean Information Retrieval

Manolis Koubarakis, Spiros Skiadopoulos, and Christos Tryfonopoulos

Abstract—We study the complexity of query satisfiability and entailment for the Boolean Information Retrieval models \( WP \) and \( AWP \) using techniques from propositional logic and computational complexity. \( WP \) and \( AWP \) can be used to represent and query textual information under the Boolean model using the concept of attribute with values of type text, the concept of word, and word proximity constraints. Variations of \( WP \) and \( AWP \) are in use in most deployed digital libraries using the Boolean model, text extenders for relational database systems (e.g., Oracle 10g), search engines, and P2P systems for information retrieval and filtering.

Index Terms—Boolean information retrieval, computational complexity, data models, query languages, satisfiability, entailment, proximity.

1 INTRODUCTION

We study two well-known data models of Information Retrieval (IR) [2] and digital libraries [9], [10], [8], which we have called \( WP \) and \( AWP \) in [21], [19], [30], [29], [28], [20]. Data model \( WP \) is based on free text and its query language is based on the Boolean model for word patterns. Word patterns are formulas that enable the expression of constraints on the existence, nonexistence, or proximity of words in a text document. Data model \( AWP \) extends \( WP \) with named attributes with free text as values. The query language of \( AWP \) is also a simple extension of the query language of \( WP \) so that attributes are included.

Models such as \( WP \) that are based on word patterns were introduced in the early days of IR and have been implemented in many digital library systems in wide use today [2]. Word patterns are also used in 1) all current search engines, 2) advanced IR models such as the model of proximal nodes [22] which allows proximity operators between arbitrary structural components of a document (e.g., paragraphs or sections), and 3) recent full-text extensions to XML-based languages e.g., TeXQuery [1].

The model \( AWP \) has been used recently in our systems DIAS, P2P-DIET, DHTrie, and LibraRing [17], [19], [30], [29], [28]. DIAS [19] is a distributed alert service for digital libraries which utilizes a P2P architecture and protocols similar to that of the event dissemination system SIENA [7]. DIAS uses \( WP \) and \( AWP \) as an expressive data model and query language for textual information. P2P-DIET [17] is the ancestor of DIAS and uses \( AWP \) as a metadata model for describing and querying digital resources. An extension of model \( AWP \), called \( AWPS \), that introduces a similarity operator based on the IR vector space model, is used in the P2P systems DHTrie [29] and LibraRing [28] that are built on top of distributed hash tables [3].

In the database literature, word patterns have been studied by Chang and colleagues in the context of integrating heterogeneous digital libraries [9], [10], [8]. The model \( AWP \) is essentially the model of [8] but with a slightly different class of word patterns.

Even though many deployed systems are using \( WP \) and \( AWP \) and many papers have appeared on their variations, only [9], [10], [8], [21], [19] have studied in depth the logical foundations of these data models. As we have previously discussed in [21], we would like to develop information retrieval and filtering systems in a principled and formal way. With this motivation and the architectures of [19], [17], [30], [29], [28] in mind, we have posed the following requirements for models and languages to be used in information retrieval and filtering systems [21]:

1. Expressivity. The languages for documents and queries must be rich enough to satisfy the demands of information consumers and capabilities of information providers.
2. Formality. The syntax and semantics of the proposed models and languages must be defined formally.
3. Computational efficiency. The following problems should be defined formally and algorithms must be provided for their efficient solution (keeping in mind that there will be a trade-off with the expressivity requirement):
   a. The satisfiability problem: Deciding whether a query can be satisfied by any document at all.
   b. The satisfaction problem: Deciding whether a document satisfies a query.
   c. The filtering problem: Given a collection of queries \( Q \) and an incoming document \( d \), find all queries \( q \in Q \) that satisfy \( d \).
d. The entailment problem: Deciding whether a query is more or less “general” than another.

In previous work, we have defined formally the models \( WP \) and \( AWP \) \cite{29} and \cite{12} and presented efficient centralized and distributed algorithms for the filtering problem \cite{30}. In this paper, we continue our formal work in this area and concentrate on model-theoretic questions for the logics of \( WP \) and \( AWP \) that have been ignored in previous papers. We study the model theory of \( WP \) and \( AWP \) and especially-questions related to satisfiability and entailment. We show that the satisfiability problem for queries in \( WP \) and \( AWP \) is \( \mathcal{NP} \)-complete and the entailment problem is \( \mathcal{coNP} \)-complete. We also discuss cases where these problems can be solved in polynomial time. Our results are original and complement the studies of \cite{8}, \cite{21} where no such complexity questions were posed.

The rest of the paper is organized as follows: In the next section, we present the models \( WP \) and \( AWP \). Sections 3 and 4 presents our complexity results on satisfiability and entailment. Then, Section 5 discusses related work. The last section concludes the paper and discusses our plans for future work.

2 THE MODELS \( WP \) AND \( AWP \)

Let us start by presenting the data model \( WP \) and its query language. \( WP \) has been inspired by \cite{10}. It assumes that textual information is in the form of free text and can be queried by word patterns (hence, the acronym for the model).

We assume the existence of a finite alphabet \( \Sigma \). A word is a finite nonempty sequence of letters from \( \Sigma \). We also assume the existence of a (finite or infinite) set of words called the vocabulary and denoted by \( V \). A text value \( s \) of length \( n \) over vocabulary \( V \) is a total function \( s : \{1, 2, \ldots, n\} \rightarrow V \). In other words, a text value \( s \) is a finite sequence of words from the assumed vocabulary and \( s(i) \) gives the \( i \)-th element of \( s \). \( |s| \) will denote the length of text value \( s \) (i.e., its number of words).

We now give the definition of word pattern. We assume the existence of a set of (distance) intervals

\[
I = \{[l, u] : l, u \in \mathbb{N}, l \geq 0 \text{ and } l \leq u\} \cup \{[l, \infty) : l \in \mathbb{N} \text{ and } l \geq 0\}.
\]

Let \( i \) be an interval in \( I \). We will denote the left-endpoint (respectively, right-endpoint) of \( i \) by \( \inf(i) \) (respectively, \( \sup(i) \)).

**Definition 1.** Let \( V \) be a vocabulary. A word pattern over vocabulary \( V \) is a formula in any of the following forms:

1. \( w \), where \( w \) is a word of \( V \).
2. \( w_1 \cdots \wedge_i \cdots \wedge_{i-1} w_n \), where \( w_1, \ldots, w_n \) are words of \( V \) and \( i_1, \ldots, i_{n-1} \) are intervals of \( I \).
3. \( \neg \psi, \phi_1 \vee \phi_2 \), or \( \phi_1 \wedge \phi_2 \), where \( \phi, \phi_1, \phi_2 \) are word patterns.

**Example 1.** The following are word patterns:

\[
\begin{align*}
\text{constraint} \wedge (\text{optimization} \vee \text{programming}) \\
\neg \text{algorithms} \wedge ((\text{complexity} \wedge \prec_{[1,5]} \text{satisfaction}) \vee (\text{complexity} \wedge \prec_{[1,8]} \text{filtering})).
\end{align*}
\]

Operator \( \prec_i \) is called a proximity operator and is a generalization of the traditional IR operators \( kW \) and \( kN \) \cite{10}. Proximity operators are used to capture the concepts of order and distance between words in a text document. They can be used to construct formulas of \( WP \) that we will call proximity word patterns (Case 2 of Definition 1). The proximity word pattern \( w_1 \prec_{[0.8]} w_2 \) stands for “word \( w_1 \) is before \( w_2 \) and is separated by \( w_2 \) by at least \( l \) and at most \( u \) words.” The interpretation of proximity word patterns with more than one operator \( \prec_i \) is similar.

Traditional IR systems have proximity operators \( kW \) and \( kN \) where \( k \) is a natural number. The proximity word pattern \( wp_1 \prec_{kW} wp_2 \) stands for “word pattern \( wp_1 \) is before \( wp_2 \) and is separated by \( wp_2 \) by at most \( k \) words.” In our work, this can be captured by \( wp_1 \prec_{[0,k]} wp_2 \). The operator \( kN \) is used to denote distance of at most \( k \) words where the order of the involved patterns does not matter. In \( WP \), the expression \( wp_1 \prec_{kW} wp_2 \) can be approximated by \( wp_1 \prec_{[0,k]} wp_2 \vee wp_2 \prec_{[0,k]} wp_1 \). Chang et al. \cite{10} gives an example (page 23) that demonstrates why these two expressions are not equivalent given the meaning of operator \( kN \). The example involves a text value and word patterns with overlapping positions in that text value hence the difference.

The development of proximity word patterns in \cite{9}, \cite{10}, \cite{8} follows closely the IR tradition, i.e., operators \( kW \) and \( kN \) (already mentioned above) are used together with the boolean operators \( AND \) and \( OR \). These operators can be intermixed in arbitrary ways (e.g., \((wp_1 AND (wp_2 (8W) wp_3))\) \((10W) wp_4\)), where \( wp_1, wp_2, wp_3, wp_4 \) are words is a legal expression), and the result of their evaluation on document databases is defined in an algebraic way. \( WP \) opts for an approach which is more in the spirit of Boolean logic, allows negation and carefully distinguishes word patterns with and without proximity operators. This leads to a simpler language because cumbersome (and not especially useful) constructions such as the above are avoided. In the spirit of Boolean logic, an atomic word pattern (i.e., a word or a proximity word pattern) allows us to distinguish between text values: those that satisfy it, and those that do not. Boolean operators are then given their standard semantics.

In addition to the above operators, \( WP \) allows the expression of simple order constraints between words using operators \( \prec_{[0,\infty]} \). Order constraints of the form \( \prec_{[0,\infty]} \) between various text structures are also present in more advanced text model proposals such as the model of proximal nodes of \cite{22}.

**Definition 2.** A word pattern will be called positive if it does not contain negation. A word pattern will be called proximity-free if it does not contain formulas of the form \( w_1 \prec_{i_1} \cdots \prec_{i_{n-1}} w_m \). A word pattern will be called conjunctive if it does not contain disjunction.

**Example 2.** The following are positive word patterns:

- satisfiability
- local \( \wedge \) search \( \wedge \) algorithms,
- information \( \wedge \) retrieval \( \wedge \) dissemination,
- logic \( \prec_{[1,1]} \) computational \( \prec_{[0,8]} \) complexity.

The first three are proximity-free word patterns. The first, second, and fourth word pattern is conjunctive.
Definition 3. Let \( \mathcal{V} \) be a vocabulary, \( s \) a text value over \( \mathcal{V} \), and \( wp \) a word pattern over \( \mathcal{V} \). The concept of \( s \) satisfying \( wp \) (denoted by \( s \models wp \)) is defined as follows:

1. If \( wp \) is a word of \( \mathcal{V} \), then \( s \models wp \) iff there exists \( p \in \{1, \ldots, |s|\} \) and \( s(p) = wp \).
2. If \( wp \) is a proximity word pattern of the form \( w_1 \prec_i \cdots \prec_{i-1} w_n \), then \( s \models wp \) iff there exist \( p_1, \ldots, p_n \in \{1, \ldots, |s|\} \) such that, for all \( j = 2, \ldots, n \) we have \( s(p_j) = w_j \) and \( p_j - p_{j-1} = 1 \in i_{j-1} \).
3. If \( wp \) is of the form \( \neg wp_1, wp_1 \land wp_2, wp_1 \lor wp_2 \) or \( (wp_1) \), then \( s \models wp \) is defined exactly as satisfaction for Boolean logic.

A word pattern \( wp \) is called satisfiable if there is a text value \( s \) that satisfies it. Otherwise, it is called unsatisfiable.

Example 3. The word patterns of Examples 1 and 2 are satisfiable. Word patterns

\[-\text{programming} \land (\text{constraint} \prec_{[0,0]} \text{programming}),
\]

\[(\text{constraint} \prec_{[0,0]} \text{programming}) \land (\neg (\text{constraint} \prec_{[0,2]} \text{programming}))
\]

are unsatisfiable.

Definition 4. Let \( wp_1 \) and \( wp_2 \) be word patterns. We will say that \( wp_1 \) entails \( wp_2 \) (denoted by \( wp_1 \models wp_2 \)) iff for every text value \( s \) such that \( s \models wp_1 \), we have \( s \models wp_2 \). If \( wp_1 \models wp_2 \) and \( wp_2 \models wp_1 \), then \( wp_1 \) and \( wp_2 \) are called equivalent (denoted by \( wp_1 \equiv wp_2 \)).

Example 4. Word pattern \( \text{constraint} \land \text{programming} \) entails word pattern \( \text{constraint} \). Word pattern

\[\text{optimization} \land (\text{constraint} \prec_{[0,0]} \text{programming})
\]

entails \( \text{constraint} \prec_{[0,10]} \text{programming} \).

Finally, word patterns

\[\text{constraint} \prec_{[0,4]} \text{programming},
\]

\[\text{constraint} \land (\text{constraint} \prec_{[0,4]} \text{programming})
\]

are equivalent.

Proposition 1. Let \( wp_1 \) and \( wp_2 \) be two word patterns. \( wp_1 \models wp_2 \iff wp_1 \land \neg wp_2 \) is unsatisfiable.

Let us close this section by pointing out that proximity word patterns have been considered as atomic formulas of \( WP \) (Definition 1) because, in general, negation cannot be moved inside a proximity word pattern as in the case of Boolean operators. The interested reader can be persuaded by trying to do this for the following formula:

\[-(\text{luxurious} \prec_{[0,3]} \text{hotel} \prec_{[0,3]} \text{beach})
\]

If we restrict our attention to proximity formulas with a single proximity operator, this restriction can easily be lifted. For example, the word pattern

\[-(\text{luxurious} \prec_{[0,3]} \text{hotel})
\]

is equivalent to the following:

\[-\text{luxurious} \lor \neg \text{hotel} \lor \text{hotel} \prec_{[0,\infty]} \text{luxurious}\lor \text{luxurious} \prec_{[4,\infty]} \text{hotel}.
\]

Let us now use the machinery of \( WP \) to define data model \( AWP \). The new concept of \( AWP \) is the concept of attribute with value free text (in the acronym \( AWP \), the letter \( A \) stands for “attribute”).

We assume the existence of a countably infinite set of attributes \( \mathcal{U} \) called the attribute universe. A document schema \( D \) is a pair \((\mathcal{A}, \mathcal{V})\), where \( \mathcal{A} \) is a subset of the attribute universe \( \mathcal{U} \) and \( \mathcal{V} \) is a vocabulary. A document \( d \) over schema \((\mathcal{A}, \mathcal{V})\) is a set of attribute-value pairs \((A, s)\) where \( A \in \mathcal{A}, s \) is a text value over \( \mathcal{V} \), and there is at most one pair \((A, s)\) for each attribute \( A \in \mathcal{A} \).

Example 5. The following is a document over schema \((\{\text{AUTHOR}, \text{TITLE}, \text{ABSTRACT}\}, \mathcal{V})\):

\[(\{\text{AUTHOR},"John Brown\"\},
\]

\[\{\text{AUTHOR,"Local search and constraint programming"},
\]

\[\{\text{ABSTRACT,"In this paper we show ..."}\}\}.
\]

The syntax of the query language of \( AWP \) is given by the following recursive definition.

Definition 5. A query over schema \((\mathcal{A}, \mathcal{V})\) is a formula in any of the following forms:

1. \( A \models wp \), where \( A \in \mathcal{A} \) and \( wp \) is a word pattern over \( \mathcal{V} \) (this is read as “\( A \) contains word pattern \( wp \)”).
2. \( A = s \), where \( A \in \mathcal{A} \) and \( s \) is a text value over \( \mathcal{V} \).
3. \( \neg \phi, \phi_1 \lor \phi_2, \phi_1 \land \phi_2 \), where \( \phi, \phi_1 \), and \( \phi_2 \) are queries.

Example 6. The following is a query over the schema shown in Example 5:

\[\text{AUTHOR} \equiv \text{Brown} \land \\
\text{TITLE} \equiv \text{search} \land (\text{constraint} \prec_{[0,0]} \text{programming}).
\]

Definition 6. Let \( D \) be a document schema, \( d \) a document over \( D \), and \( \phi \) a query over \( D \). The concept of document \( d \) satisfying query \( \phi \) (denoted by \( d \models \phi \)) is defined as follows:

1. If \( \phi \) is of the form \( A \models wp \), then \( d \models \phi \iff \exists s\ (d \models wp) \land s \models wp \).
2. If \( \phi \) is of the form \( A = s \), then \( d \models \phi \iff \exists s\ (d \models s) \land s \models \).
3. If \( \phi \) is of the form \( \neg \phi_1 \), then \( d \models \phi \iff \neg \phi_1 \).

Similarly, for \( \land \) and \( \lor \).

Example 7. The query of Example 6 is satisfied by the document of Example 5.

Proposition 2. Let \( A \) be an attribute and \( wp_1, wp_2 \) be word patterns. Then, the following equivalences hold:

1. \( \neg A \equiv wp \equiv A \equiv \neg wp \).
2. \( A \equiv (wp_1 \land wp_2) \equiv (A \equiv wp_1) \land (A \equiv wp_2) \).
3. \( A \equiv (wp_1 \lor wp_2) \equiv (A \equiv wp_1) \lor (A \equiv wp_2) \).
4. \( \neg (A \equiv (wp_1 \land wp_2)) \equiv (\neg A \equiv wp_1) \lor (\neg A \equiv wp_2) \).
5. \( (A \equiv (wp_1 \lor wp_2)) \equiv (\neg A \equiv wp_1) \land (\neg A \equiv wp_2) \).
Definition 7. A query is called atomic if it is of the form \( A = t \) where \( t \) is a text value, or \( A \supseteq wp \) where \( wp \) is a word or a proximity word pattern. A query is called conjunctive if it does not contain disjunction.

Example 8. The following queries are atomic:

\[ \text{AUTHOR = "James Brown,"} \]
\[ \text{TITLE \supseteq search,} \]
\[ \text{ABSTRACT \supseteq constraint \lt_{[0,0]} \text{programming}.} \]

Proposition 3. Every query is equivalent to a Boolean combination of atomic queries.

Proof. Use the first three equivalences of Proposition 2 repeatedly.

\[ \square \]

3 Satisfiability and Entailment in WP

An instance of the satisfiability problem for proximity-free word patterns can be considered as an instance of the satisfiability problem for Boolean logic (SAT) and vice versa (by interchanging the roles of words and Boolean variables). Thus, we have to consider any complications that might arise due to proximity word patterns only.

In what follows, we will need the binary operation of concatenation of two text values.

Definition 8. Let \( s_1 \) and \( s_2 \) be text values over vocabulary \( V \).

Then, the concatenation of \( s_1 \) and \( s_2 \) is a new text value denoted by \( s_1s_2 \) and defined by the following:

1. \( |s_1s_2| = |s_1| + |s_2| \)
2. \( s_1s_2(x) = \begin{cases} s_1(x) & \text{for all } x \in \{1, \ldots, |s_1|\} \\ s_2(x - |s_1|) & \text{for all } x \in \{|s_1| + 1, \ldots, |s_2| + |s_1|\}. \end{cases} \)

We will also need the concept of the empty text value which is denoted by \( \epsilon \) and has the property \( |\epsilon| = 0 \). The following properties of concatenation are easily seen:

1. \( (s_1s_2)s_3 = s_1(s_2s_3) \), for all text values \( s_1, s_2, \) and \( s_3 \).
2. \( se = es = s \) for every text value \( s \).

The associativity of concatenation allows us to write concatenations of more than two text values without using parentheses.

The following variant of the concept of satisfaction captures the notion of a set of positions in a text value containing exactly the words that contribute to the satisfaction of a positive proximity-free word pattern. This variant is used in Lemma 1 and in Proposition 4.

Definition 9. Let \( V \) be a vocabulary, \( s \) a text value over \( V \), \( wp \) a positive proximity-free word pattern over \( V \), and \( P \) a subset of \( \{1, \ldots, |s|\} \). The concept of \( s \) satisfying \( wp \) with set of positions \( P \) (denoted by \( s \models_P wp \)) is defined as follows:

1. If \( wp \) is a word of \( V \), then \( s \models_P wp \) iff there exists \( x \in \{1, \ldots, |s|\} \) such that \( P = \{x\} \) and \( s(x) = wp \).
2. If \( wp \) is of the form \( wp_1 \land wp_2 \), then \( s \models_P wp \) iff there exist sets of positions \( P_1, P_2 \subseteq \{1, \ldots, |s|\} \) such that \( s \models_{P_1} wp_1, s \models_{P_2} wp_2 \) and \( P = P_1 \cup P_2 \).
3. If \( wp \) is of the form \( wp_1 \lor wp_2 \), then \( s \models_P wp \) iff \( s \models_P wp_1 \) or \( s \models_P wp_2 \).
4. If \( wp \) is of the form \( (wp_1) \), then \( s \models_P wp \) iff \( s \models_P wp_1 \).

We also need the following notation: Let \( P \) be a subset of the set of natural numbers \( \mathbb{N} \), and \( x \in \mathbb{N} \). We will use the notation \( P + x \) to denote the set of natural numbers \( \{p + x : p \in P\} \).

Lemma 1. Let \( s \) and \( s' \) be text values, \( wp \) be a positive proximity-free word pattern, and \( P \subseteq \{1, \ldots, |s|\} \). If \( s \models_P wp \), then \( ss' \models_P wp \) and \( s's \models_{P+[|s|]} wp \).

Proposition 4. If \( wp \) is a positive proximity-free word pattern, then \( wp \) is satisfiable. In fact, there exists a text value \( s_0 \) such that

1. \( |s_0| \leq |wp| \cdot \text{ops}(wp) \), where \( \text{ops}(wp) \) is the number of operators of \( wp \) (or 1 if \( wp \) has no operators).
2. Every word of \( s_0 \) is a word of \( wp \).
3. \( s_0 \models_{\{1, \ldots, |s_0|\}} wp \).

Proof. The proof is by induction on the structure of \( wp \).

Base case: Let \( wp \) be a word \( w \in V \). In this case, \( wp \) is satisfiable because we can form a text value \( s_0 \) such that \( s_0 \models_{\{1\}} w \), where \( |s_0| = 1 \) and \( s_0(1) = w \). The conclusion of the lemma is now obviously satisfied.

Inductive step: Let \( wp \) be a positive proximity-free word pattern of the form \( wp_1 \land wp_2 \), and assume that the inductive hypothesis holds for \( wp_1 \) and \( wp_2 \). Then, we can form text values \( s_0^1 \) and \( s_0^2 \) such that \( s_0^1 \models_{\{1, \ldots, |s_0^1|\}} wp_1 \) and \( s_0^2 \models_{\{1, \ldots, |s_0^2|\}} wp_2 \). Then, from Lemma 1, we have

\[ s_0^1s_0^2 \models_{\{1, \ldots, |s_0^1|+|s_0^2|\}} wp_1 \land wp_2 \]

and

\[ s_0^1s_0^2 \models_{\{1, \ldots, |s_0^1|+|s_0^2|\}} wp_2. \]

Finally, from Definition 9, we have

\[ s_0^1s_0^2 \models_{\{1, \ldots, |s_0^1|+|s_0^2|\}} wp_1 \land wp_2 \]

as required. It is also easy to see that

\[ |s_0^1s_0^2| = |s_0^1| + |s_0^2| \leq |wp_1| \cdot \text{ops}(wp_1) + |wp_2| \cdot \text{ops}(wp_2) < |\text{ops}(wp_1) + \text{ops}(wp_2)| \cdot |wp| < \text{ops}(wp_1) \cdot |wp|. \]

The \( \lor \) case is done similarly. \( \square \)

Obviously, proximity word patterns are also satisfiable.

Proposition 5. Let \( wp \) be a proximity word pattern of the form \( w_1 \preceq_{i_1} \cdots \preceq_{i_l} w_n \). Then, \( wp \) is satisfied by the text value \( s = w_1z_1 \cdots z_{n-1}w_n \), where \( z_l, l = 1, \ldots, n-1 \) are text values of the following form. If \( \inf(i) > 0 \) then \( z_l \) is formed by \( \inf(i) \) successive occurrences of the special word \( \# \) which is
not contained in \( wp \). Otherwise, if \( \text{inf}(i_1) \), then \( z_1 \) is the empty text value \( \epsilon \).

Moreover, any text value satisfying a proximity word pattern is of a very special form.

**Proposition 6.** Let \( wp \) be a proximity word pattern of the form \( w_1 \prec i_1 \cdots \prec i_{\ell-1} \prec w_n \). If \( s \models wp \), then \( s \) is of the form:

\[
s = \underbrace{w_1 \prec \cdots \prec w_1}_{i_0 \text{ times}} \cdots \underbrace{w_{\ell-1} \prec \cdots \prec w_{\ell-1}}_{i_{\ell-1} \text{ times}} \underbrace{w_n \prec \cdots \prec w_n}_{i_n \text{ times}},
\]

where \( 0 \leq i_0, i_1, \ldots, i_{\ell-1} \in i_{\ell-1} \), \( 0 \leq i_n \), and each occurrence of the symbol \( ? \) represents an arbitrary (and not necessarily the same) word.

**Example 9.** Let us consider the proximity word pattern

\[ wp = \text{constraint} \prec \big{(}\text{constraint}\text{programming}\big{)} \text{methods}. \]

It is easy to verify that text value “many applications use constraint programming algorithms and methods to solve interesting problems” 1) is of the form set by Proposition 6 and 2) satisfies word pattern \( wp \).

Finally, we show that any positive word pattern is satisfiable.

**Proposition 7.** If \( wp \) is a positive word pattern, then \( wp \) is satisfiable.

**Proof.** We will construct a text value \( t \) such that \( t \models wp \). If \( wp \) contains \( m \) proximity word patterns \( \phi_1, \ldots, \phi_m \), text value \( t \) is of the form \( s_0s_1 \cdots s_m \) where:

- \( s_0 \) is a sequence formed by the juxtaposition of all words appearing in \( wp \) in any order, and
- for every \( j = 1, \ldots, m \), \( s_j \) is a text value, formed as in Proposition 5, such that \( s_j \models \phi_j \).

**Lemma 2.** Let \( wp_1 \) and \( wp_2 \) be proximity word patterns of the following form:

\[
wp_1 = a_1 \prec i_1 \cdots \prec i_{\ell-1} \prec a_n \quad \text{and} \quad wp_2 = b_1 \prec j_1 \cdots \prec j_{\ell-1} \prec b_m.
\]

Word pattern \( wp_1 \) entails \( wp_2 \) iff the following conditions hold:

- **Condition 1.** Word pattern \( wp_2 \) is equal to

\[
ap_1 \prec j_1 \cdots \prec j_{\ell-1} \prec p_m,
\]

where \( 1 \leq p_1 < \cdots < p_m \leq n \).

- **Condition 2.** For every \( v = 1, \ldots, m - 1 \), we have:

\[
\text{inf}(j_v) \leq \text{inf}(i_p) + \cdots + \text{inf}(i_{p_{v+1}}) + p_{v+1} - p_v - 1
\]

\[
\sup(j_v) \begin{cases} 
\geq \sup(i_p) + \cdots + \sup(i_{p_{v+1}}) & \text{if all } \sup(i_p), \ldots,
\sup(i_{p_{v+1}}) \text{ are different than } \infty \\
\infty & \text{otherwise.}
\end{cases}
\]

**Proof.** The “if” case is obvious. For the “only if” part, let us assume that \( wp_1 \models wp_2 \) holds. We will prove that \( wp_2 \) is of the form set by the lemma. The proof is in three steps.

**Step 1 (Condition 1).** We will first prove that the words of \( wp_2 \) are a subset of the words in \( wp_1 \), i.e.,

\[
\{b_1, \ldots, b_m\} \subseteq \{a_1, \ldots, a_n\}.
\]

By contradiction, let us assume that there exists a word \( b_v, 1 \leq v \leq m \), of \( wp_2 \) such that \( b_v \notin \{a_1, \ldots, a_n\} \). Let us now consider text value \( t \) defined as:

\[
\tau = a_1 \# \cdots \# a_{n-1} \# \# a_n,
\]

where \# is a special word which is not contained in \( wp_1 \) and \( wp_2 \) and \( 1 \leq i_1, \ldots, i_n \in i_{n-1} \). It is easy to verify that \( \tau \) satisfies \( wp_1 \) but, since \( \tau \) does not include word \( b_v \), it does not satisfy \( wp_2 \). Thus, we have \( wp_1 \not\models wp_2 \) which contradicts our initial assumption.

**Step 2 (Condition 1).** We will now prove that the words of \( wp_1 \) that appear in \( wp_2 \) actually appear in the same order as they do in \( wp_1 \), i.e., word pattern \( wp_2 = a_n \prec j_1 \cdots \prec j_{\ell-1} \prec p_m \), where \( 1 \leq p_1 < \cdots < p_m \leq n \).

By contradiction, let us assume that there exist two distinct words \( b_i = a_p \) and \( b_i = a_{p+1} \), \( 1 \leq v' \leq m \), of \( wp_2 \) such that \( p_v \geq p_v' \).

In other words,

\[
wp_1 = a_1 \prec i_1 \cdots \prec i_{\ell-1}
\]

\[
ap_1 \prec i_{p_1} \cdots \prec i_{p_{v+1}}
\]

\[
ap_v \prec i_{p_v} \cdots \prec i_{p_{m}}
\]

\[
wp_2 = a_{p_1} \prec j_1 \cdots \prec j_{\ell-1}
\]

\[
ap_v \prec j_{p_v} \cdots \prec j_{p_{m}}
\]

Word pattern \( wp_2 \) is equal to

\[
ap_1 \prec j_1 \cdots \prec j_{\ell-1} \prec p_m.
\]

It is easy to verify that text value \( \tau \) (defined in (1)) satisfies \( wp_1 \) but it does not satisfies \( wp_2 \) a contradiction.

**Step 3 (Condition 2).** Finally, we will prove that for every \( v = 1, \ldots, m - 1 \), we have:

\[
\text{inf}(j_v) \leq \text{inf}(i_p) + \cdots + \text{inf}(i_{p_{v+1}}) + p_{v+1} - p_v - 1
\]

\[
\sup(j_v) \begin{cases} 
\geq \sup(i_p) + \cdots + \sup(i_{p_{v+1}}) & \text{if all } \sup(i_p), \ldots,
\sup(i_{p_{v+1}}) \text{ are different than } \infty \\
\infty & \text{otherwise.}
\end{cases}
\]

By contradiction, let us assume that there exists a subformula \( a_p \prec j_1 \prec a_{p+1} \) of \( wp_2 \) such that

\[
\text{inf}(j_v) > \text{inf}(i_p) + \cdots + \text{inf}(i_{p_{v+1}}) + p_{v+1} - p_v - 1.
\]

From Step 2, word patterns \( wp_1 \) and \( wp_2 \) are of the following form:

\[
w_1 = a_1 \prec i_1 \cdots \prec i_{p_{v+1}}
\]

\[
ap_v \prec i_{p_v} \cdots \prec i_{p_{m}}
\]

\[
ap_v \prec i_{p_v} \cdots \prec i_{p_{m}}
\]

\[
w_2 = a_{p_1} \prec j_1 \cdots \prec j_{p_m}
\]

\[
ap_v \prec j_{p_v} \cdots \prec j_{p_{m}}
\]

\[
ap_v \prec j_{p_v} \cdots \prec j_{p_{m}}
\]
Let us now construct a text value \( \tau' \) defined as:

\[
\tau' = a_1 \# \cdots \# a_2 \cdots
\]

\[
\vdots
\]

\[
\vdots
\]

\[
\vdots
\]

(3)

where \# is a special word which is not contained in \( wp_1 \) and \( wp_2 \), and for every \( s \), \( 1 \leq s \leq n - 1 \), \( i_s = \inf(i_s) \) holds. It is easy to verify that \( \tau' \) satisfies \( wp_1 \). Notice that between words \( a_p \) and \( a_{p+1} \) in \( \tau' \) there are exactly \( \inf(i_p) + \cdots + \inf(i_{p+1}) + p_{v+1} - p_v - 1 \) words. Therefore, since (2) holds, \( \tau' \) does not satisfy the subformula \( a_p \supset_{\tau} a_{p+1} \) of \( wp_2 \) and, thus, it does not satisfy \( wp_2 \). Thus, we have \( wp_1 \not\models wp_2 \) which contradicts our initial assumption.

The proof involving \( \sup(j_x) \) is similar. It differs only in the way we construct text value \( \tau' \) (3) and specifically in the values of \( i_1, \ldots, i_{n-1} \). We now require that \( i_1 \in i_1, \ldots, i_{n-1} \in i_{n-1} \) and for every \( s \), \( p_v \leq s \leq p_{v+1} \), we define:

\[
i_s = \begin{cases} 
\sup(i_s) & \text{if } \sup(i_s) \text{ is different} \\
\sup(j_s) + 1 & \text{otherwise}
\end{cases}
\]

Proposition 8. Let \( wp_1 \) and \( wp_2 \) be proximity word patterns with \( n \) and \( m \) words, respectively. Deciding whether \( wp_1 \models wp_2 \) can be done in \( O(n + m) \) time.

\[
\text{Proposition 9. } SAT \text{ is polynomially reducible to } SAT(WP).
\]

\[
\text{Proof. } \text{Trivial by considering propositional variables to be words.}
\]

Proposition 10. \( SAT(WP) \) is polynomially reducible to \( SAT \).

\[
\text{Proof. } \text{Let } \phi \text{ be a formula of } WP. \text{ We transform } \phi \text{ into an instance } \phi' \text{ of } SAT \text{ as follows: We start with } \phi' \text{ being } \phi \text{ (words of } \phi \text{ play the role of propositional variables in } \phi') \text{. Then, we substitute each proximity word pattern } wp \text{ of } \phi' \text{ by a brand new propositional variable } v_{wp}. \text{ Finally, we conjoin to } \phi' \text{ the following formulas:}
\]

- \( v_{wp} \Rightarrow u \) for each proximity word pattern \( wp \) and word \( w \) of \( wp \).
- \( v_{wp} \Rightarrow v_{wp} \) for each pair of proximity word patterns \( wp_1, wp_2 \) such that \( wp_1 \models wp_2 \).

The above steps can be done in polynomial time because entailment of proximity word patterns can be done in polynomial time (Proposition 8). It is also easy to see that \( \phi \) is a satisfiable formula of \( WP \) iff \( \phi' \) is a satisfiable formula of Boolean logic. Then, the result holds.

Propositions 9 and 10 have the following corollary.

Corollary 1. \( SAT \) is polynomially reducible to \( SAT(WP) \).

\[
\text{Proof. } \text{Let } \phi \text{ be a query of } WP. \text{ Using Proposition 2, } \phi \text{ can easily be transformed into a formula } \theta \text{ which is a Boolean combination of atomic queries. This transformation can be done in time linear in the size of the formula. The next step is to substitute in } \theta \text{ atomic formulas } A = s \text{ and } A \supset wp \text{ (where } wp \text{ is a word or a proximity word pattern) by propositional variables } p_{A_{s=0}} \text{ and } p_{A_{wp}}, \text{ respectively, to obtain formula } \theta'. \text{ Finally, the following formulas are conjoined to } \theta' \text{ to obtain } \psi:
\]

1. If \( A = s_1 \) and \( A = s_2 \) are conjuncts of \( \theta' \) and \( s_1 \neq s_2 \), then conjoin \( p_{A_{s=s_1}} \equiv \neg p_{A_{s=s_2}} \).
2. If \( A = s \) and \( A \supset wp \) are conjuncts of \( \theta' \) and \( s \models wp \), then conjoin \( p_{A_{s=0}} \Rightarrow p_{A_{wp}} \).
3. If \( A = s \) and \( A \supset wp \) are conjuncts of \( \theta' \) and \( s \not\models wp \), then conjoin \( p_{A_{s=0}} \Rightarrow \neg p_{A_{wp}} \).
4. If \( A \supset wp_1 \) and \( A \supset wp_2 \) are conjuncts of \( \theta' \) and \( wp_1 \models wp_2 \), then conjoin \( p_{A_{wp_1}} \Rightarrow p_{A_{wp_2}} \).

The above step can be done in polynomial time because satisfaction and entailment of word patterns in \( \theta \) can be done in polynomial time. The result for satisfaction is obvious and the result for entailment is from Proposition 8. It is also easy to see that \( \phi \) is a satisfiable query if \( \psi \) is a satisfiable formula of Boolean logic. Then, the result holds.

Propositions 12 and 13 have the following corollary.

Corollary 2. \( SAT(WP) \) is polynomially reducible to \( SAT \).

\[
\text{Proof. } \text{Let } \phi \text{ be a query of } WP. \text{ Using Proposition 2, } \phi \text{ can easily be transformed into a formula } \theta \text{ which is a Boolean combination of atomic queries. This transformation can be done in time linear in the size of the formula. The next step is to substitute in } \theta \text{ atomic formulas } A = s \text{ and } A \supset wp \text{ (where } wp \text{ is a word or a proximity word pattern) by propositional variables } p_{A_{s=0}} \text{ and } p_{A_{wp}}, \text{ respectively, to obtain formula } \theta'. \text{ Finally, the following formulas are conjoined to } \theta' \text{ to obtain } \psi:
\]

1. If \( A = s_1 \) and \( A = s_2 \) are conjuncts of \( \theta' \) and \( s_1 \neq s_2 \), then conjoin \( p_{A_{s=s_1}} \equiv \neg p_{A_{s=s_2}} \).
2. If \( A = s \) and \( A \supset wp \) are conjuncts of \( \theta' \) and \( s \models wp \), then conjoin \( p_{A_{s=0}} \Rightarrow p_{A_{wp}} \).
3. If \( A = s \) and \( A \supset wp \) are conjuncts of \( \theta' \) and \( s \not\models wp \), then conjoin \( p_{A_{s=0}} \Rightarrow \neg p_{A_{wp}} \).
4. If \( A \supset wp_1 \) and \( A \supset wp_2 \) are conjuncts of \( \theta' \) and \( wp_1 \models wp_2 \), then conjoin \( p_{A_{wp_1}} \Rightarrow p_{A_{wp_2}} \).

The above step can be done in polynomial time because satisfaction and entailment of word patterns in \( \theta \) can be done in polynomial time. The result for satisfaction is obvious and the result for entailment is from Proposition 8. It is also easy to see that \( \phi \) is a satisfiable query if \( \psi \) is a satisfiable formula of Boolean logic. Then, the result holds.

Propositions 12 and 13 have the following corollary.
The following proposition shows that, as in the case of \( WP \), satisfiability and entailment of conjunctive queries in \( AWP \) can be done in PTIME. This is good news given that conjunctive \( AWP \) queries are typically utilized in implementations such as [19], [17], [28].

**Proposition 14.** The satisfiability and entailment problems for conjunctive \( AWP \) queries can be solved in polynomial time.

To obtain a more accurate picture of the tractable versus intractable classes of queries in \( AWP \) one can profitably utilize such results from the propositional satisfiability literature. For example, it is easy to see now that each tractable class \( C \) of \( SAT \) formulas has a corresponding class \( C' \) of tractable formulas of \( WP \) or \( AWP \) if the 2-variable propositional formulas used in the proofs of Propositions 10 and 13 belong to \( C \) (e.g., this holds for \( C \) being the class of propositional formulas with at most two variables using the tractability of 2-SAT).

### 5 RELATED WORK

In this section, we discuss related research. Since formal analysis based on logic and complexity as done in this paper is not common in Information Retrieval research, this section briefly surveys other data models (and systems) related to the ones studied in this paper.

#### 5.1 WP

To the best of our knowledge, the papers by Chang and colleagues [9], [10], [8] and the present paper are the only comprehensive formal treatments of proximity word patterns in the literature.

Search engines use models similar to \( WP \) and \( AWP \). The most common support for word patterns in search engines includes the ability to combine words using the Boolean operators \( \land, \lor, \) and \( \neg \). However, search engines support a version of negation in the form of binary operator \( AND-NOT \) which is essentially set difference, and therefore \( safe \) in the database sense of the term [26]. For example, a search engine query \( wp_1 \land AND-NOT \land wp_2 \) will return the set of documents that satisfy \( wp_1 \land \neg \) these that satisfy \( wp_2 \). Note also that the previous work of [10] has not considered negation in its word pattern language but has considered negation in the query language which supports attributes (the one that corresponds to our model \( AWP \)).

Proximity operators are a useful extension of the concept of “phrase search” used in current search engines. Limited forms of proximity operators have been offered in the past by various search engines of the pre-Google era (e.g., Altavista had an operator \( NEAR \) which meant word-distance 10, Lycos had an operator \( NEAR \) which meant word-distance 25, and Infoseek used to have a more sophisticated facility). Google supports proximity by the use of operator “*” which, when used between two keywords, specifies a minimum distance of one word between them (multiple occurrences of * can also be used to specify a larger minimum distance). The search engine Exalead\(^1\) has an operator \( NEAR \) which returns documents that contain given keywords in a vicinity of a fixed number of words, but no ordering of words is supported.

The need to change their index structures and the high computational cost of proximity search, is probably the reason why current search engines limit proximity support to less general operators compared to those used in models \( WP \) and \( AWP \).

Proximity operators have also been implemented in other systems such as freeWAIS [23] and INQUERY [5]. There are also advanced IR models such as the model of proximal nodes [22] with proximity operators between arbitrary structural components of a document (e.g., paragraphs or sections). Data models and query languages for full-text extensions to XML, e.g., TeXQuery [1] is the most recent area of research where proximity operators have been used.

Proximity word patterns can also be viewed as a particular kind of \emph{order constraints} in the sense of constraint networks [14] and databases [25]. There are many papers that discuss algorithms and complexity of various kinds of order constraints, e.g., gap-order constraints [24] or temporal constraints [18], [18]. The algorithms and complexity results regarding \( WP \) can also be viewed as a contribution to this research area.

#### 5.2 AWP

The data model \( AWP \) discussed in Section 2 complements recent proposals for representing and querying textual information in publish/subscribe systems [7], [6] by using linguistically motivated concepts such as \textit{word} and traditional IR operators (instead of strings and operators such as string containment [7], [6]). The methodology and techniques of this paper can be used to study the complexity of satisfiability and entailment for the subscription query language of [6] and we expect the complexity results to be similar.

In [21], [19], we have extended the model \( AWP \) by introducing a “similarity” operator based on the IR vector space model [2]. The similarity concept of this model, called \( AWPS \) (where \( S \) stands for similarity), has in the past been used in database systems with IR influences (e.g., WHIRL [13]) and, more recently, in XML-based query languages, e.g., ELIXIR [12], XIRQL [16], and XXL [27].

### 6 OUTLOOK

We have studied the model theory of \( WP \) and \( AWP \) and especially questions related to satisfiability and entailment. We showed that the satisfiability problem for queries in \( WP \) and \( AWP \) is \( NP \)-complete and the entailment problem is \( co-NP \)-complete. We also discussed cases where these problems can be solved in polynomial time.

We would like to use the lessons learned in this paper to study the complexity of query evaluation in RDBMS with text functionalities, combinations of RDBMS and IR systems [11], and proposals for full-text extensions to XML [1]. This recent paper [4] is a good example of such a study where the authors consider the concept of strings in various query languages.

---

1. Exalead (http://www.exalead.com/) is a search engine developed in France. We mention it here because Exalead is involved in the Quaero project launched in Europe in the summer of 2005 as the European response to Google.
ACKNOWLEDGMENTS

This work was performed while Manolis Koubarakis was with the Technical University of Crete.

REFERENCES