# Weak Containment for Partial Words is coNP-complete

Paweł Parys

*University of Warsaw, Warsaw, Poland*

## Abstract

A partial word is a string that may contain a number of "hole" symbols, matching any letter. Such a partial word $u$ defines a language $L_A(u)$ of all words over an alphabet $A$ to which it matches as a subword. We prove that for every alphabet $A$ of size at least two it is coNP-complete to decide given two partial words $u$, $v$ whether $L_A(u) \subseteq L_A(v)$.

*Keywords:* computational complexity, formal languages, NP-completeness, partial words, language containment

## 1. Introduction

A partial word is a string over a finite alphabet that may contain a number of "do not know" symbols, called *holes* and denoted by $\diamond$. Such a hole matches any single letter of the alphabet (possibly different letters for different holes). Partial words appear in natural ways in several areas of current interest such as molecular biology, data communication, XML querying, and DNA computing [1].

Given a partial word $u$, its *(weak) language* $L_A(u)$ contains all words over an alphabet $A$ to which $u$ matches as a subword (infix). This paper investigates the following very natural problem of weak containment: given two partial words $u$, $v$ decide whether $L_A(u) \subseteq L_A(v)$ (for some fixed alphabet $A$). The same can be stated in a more direct way: is it the case that $v$ matches as a subword to every full word $w$ over the alphabet $A$ to which $u$ matches as a subword. We prove that this problem is coNP-complete for every alphabet $A$ of size at least two, as well as when $A$ is considered as a part of the input.

Since it is more natural to reason about "existential problems", we immediately switch to the complement of the weak containment problem, about which we prove that is NP-complete. It asks whether $L_A(u) \setminus L_A(v) \neq \emptyset$, that is whether there exists a full word $w$ (a *counterexample*) over the alphabet $A$, such that $u$ matches to a subword of $w$, but $v$ does not match to any subword of $w$.

Let us start by an easy observation: if such a counterexample $w$ exists, and $u$ matches to its proper subword $w'$, then $w'$ is itself a counterexample—if $v$ does not match to any subword of $w$, then it cannot match to any subword of $w'$ as well. Thus, equivalently, we may look only for counterexamples of the same length as $u$. This immediately shows that the the problem is in NP: we can just guess a full word of length $|u|$, and check whether $u$ matches to it, and $v$ does not match to any of its subwords. It works when the alphabet $A$ is fixed, as well as when it is a part of the input.

Notice also that when the alphabet $A$ contains only one letter, the problem can be trivially solved in polynomial time, since there is only one full word of length $|u|$. Thus we may expect NP-completeness only when $|A| \geq 2$.

*Related Work.* One may consider *strong*[1] languages of partial words, and the strong containment problem, where the partial words $u$, $v$ are required to be aligned at the beginning of the matching full word, not as subwords. This problem can be easily solved in polynomial time: we have a fixed correspondence between positions of the two partial words, so in the counterexample we should try to put in some hole of $u$ a letter that does not appear at the corresponding position of $v$.

In the (complement of the) problem of tree pattern query (TPQ) containment in the presence of a DTD, we are given two tree pattern queries $p$, $q$ (which are like partial words, but in general may be branching, and may contain descendant edges saying that the next letter is matched not in a child but in any descendant)

*Email address:* parys@mimuw.edu.pl (Paweł Parys)

[1]This *weak* vs. *strong* terminology comes from the area of tree pattern queries.

and a DTD $S$ (Document Type Definition—a formalism for specifying a set of trees, weaker than finite tree automata), and we are asked whether there exists a tree satisfying $S$ to which $p$ matches but $q$ does not match. By TPC(DTD, $/, *$) let us denote the variant of this problem where $p$ and $q$ consists of a single branch without descendant edges (so, basically, these are partial words, matching to some branch of a tree). This problem is a generalization of our problem: a DTD $S$ can restrict considerations to trees consisting of a single branch (that is words) labeled by letters from alphabet $A$,[2] and partial words $u$, $v$ can be interpreted as tree pattern queries $p$, $q$. The TPC(DTD, $/, *$) problem was first incorrectly claimed to be in PTIME ([2], Theorem 7), even in presence of descendant edges, but later the authors admitted a mistake ([3], page 14). Recently in [4] (Theorem 6.6) it was shown that TPC(DTD, $/, *$) is EXPTIME-complete. This hardness proof cannot be transferred back to our partial words setting, since it is using a nontrivial DTD.

In the problem of avoidability of sets of partial words, one is given a set of partial words $X$, and has to decide whether there exists an infinite full word to which none of the partial words from $X$ matches. This problem is shown to be NP-hard [5]. It looks much harder, and it remains open whether it can be solved in NP, it is only known that it is in PSPACE [6].

A finite counterpart of the avoidability problem was considered in [7]. There one looks for a full word of given finite length (instead of an infinite word) to which none of given partial words matches. Different variants of this problem are shown to be NP-complete or #P-complete. In fact, in our proof we base on one of the results from [7].

## 2. Reduction

In our hardness proof we reduce from Problem 2 from [7], which we call finite avoidability. In the *finite avoidability* problem we are given a set $X$ of partial words, all of the same length $n$, and we are asked whether there exists a full word of length $n$ (over a fixed alphabet $A$ of size at least 2) to which none of the partial words from $X$ matches. Theorem 1 in [7] says that this problem is NP-complete; the hardness proof amounts to a direct reduction from the CNF-SAT problem [8].

[2]Equally well we could allow all trees over the alphabet $A$: if there exists a branching counterexample, then we can look at its branch to which $p$ (that is $u$) matches, and it will be still a good counterexample. Nevertheless a DTD is needed to restrict the alphabet to $A$.

As already mentioned, we reduce from the finite avoidability problem to the complement of the weak containment problem. Consider an instance of the finite avoidability problem: a set $X = \{v_1, \dots, v_s\}$ of partial words, all of the same length $n$. Let $A = \{\mathsf{a}, \mathsf{b}\} \uplus C$ be the fixed alphabet (of any size not smaller than two) over which this instance is considered. We will create two partial words $u$, $v$ such that there exists a full word of length $n$ to which none of the partial words $v_1, \dots, v_s$ matches if and only if there exists a full word to which $u$ matches but $v$ does not match to any of its subwords.

The overall idea is as follows. The partial word $u$ will contain $n$ holes, surrounded by some prefix and suffix, and $v$ will contain all $v_1, \dots, v_s$ surrounded and separated by appropriate partial words. Then the choice of a full word $w$ to which $u$ matches boils down to the choice of an $n$-letter word put in the place of the $n$ holes in $u$. The construction is maintained in such a way that:

- if $v$ is aligned so that none of $v_i$ is placed precisely over the $n$ holes, then surely $v$ will not match, and
- if $v$ is aligned so that some $v_i$ is placed precisely over the $n$ holes, then the rest of $v$ (except $v_i$) surely matches, so $v$ aligned in this way matches to $w$ if and only if $v_i$ matches to the word replacing the $n$ holes.

Now we give the details of the construction. Both partial words will consist of several blocks, each of length $2n + 5$. We use the following kinds of blocks:

- the *empty block* $\mathsf{b}^n\mathsf{bab}^{n+2}\mathsf{a}$,
- the *query block* $\diamond^n\mathsf{aab}^{n+2}\mathsf{a}$,
- the *easy-fit block* $\diamond^n\diamond\mathsf{ab}^{n+2}\mathsf{a}$,
- for each $i \in \{1, \dots, s\}$ the *i-th negative block* $v_i\mathsf{aab}^{n+2}\mathsf{a}$.

The partial word $v$ consists of

- 1-st negative block,
- $s^2 + 1$ easy-fit blocks,
- 2-nd negative block,
- $s^2 + 2$ easy-fit blocks,
- $\dots$,
- $i$-th negative block,
- $s^2 + i$ easy-fit blocks,
- $\dots$,

- $(s-1)$-th negative block,
- $s^2 + s - 1$ easy-fit blocks,
- $s$-th negative block.

Observe that the distance (in number of blocks) between the $i$-th and the $j$-th negative block in $v$ is different for each pair $i, j$, where $i < j$. Indeed, this distance is strictly between $s^2(j-i)$ and $s^2(j-i+1)$, so it determines the difference $j-i$; when the difference is fixed, the number of easy-fit blocks is clearly different between each pair.

Let $m$ be the number of blocks in $v$. The partial word $u$ consists of $2m-1$ blocks. Exactly in the middle (at block $m$) we put the query block. When the distance in $v$ between the $i$-th and the $j$-th negative block is $k$ for some $i, j$ where $i < j$, at block $m-k$ in $u$ we put the $i$-th negative block, and at block $m+k$ we put the $j$-th negative block. At the remaining positions we put empty blocks.

Below we present the two partial words for $s = 2$, where "1" and "2" denote the first and the second negative block, "e" denotes the empty block, "q" denotes the query block, and "f" denotes the easy-fit block.

$$u = \begin{array}{|c|c|c|c|c|c|c|c|c|c|c|c|c|}\hline 1 & e & e & e & e & e & q & e & e & e & e & e & 2 \\ \hline \end{array}$$

$$v = \begin{array}{|c|c|c|c|c|c|c|}\hline 1 & f & f & f & f & f & 2 \\ \hline \end{array}$$

Now, assume that there is a word $w$ to which $u$ matches, but such that $v$ does not match to any of its subwords. Let $w'$ be the fragment of $w$ substituted for the $n$ holes in the query block in $u$. We will prove that none of $v_1, \ldots, v_s$ matches to $w'$. In fact, this is straightforward: for each $i$, consider the alignment of $v$ in which the $i$-th negative block is aligned along the query block. Then the $j$-th negative block in $v$ (where $j \neq i$) is aligned to $w$ in the same place as an $j$-th negative block in $u$, so it necessarily matches. Also each easy-fit block in $v$ necessarily matches to $w$, as it is aligned in the same place as some block of $u$. We have assumed that $v$ does not match to $w$ in any place, so necessarily the $i$-th negative block of $v$ does not match to the query block in $w$. This simply means that $v_i$ does not match to $w'$.

Next, we prove the opposite direction. Consider any full word $w'$ of length $n$ to which none of $v_1, \ldots, v_s$ matches. We construct $w$ by substituting $w'$ for the $n$ holes in the query block $u$, and any letter for other holes in $u$ (that may be present in negative blocks). Of course $u$ matches to $w$. We should prove that $v$ does not match to any subword of $w$. For that, we have to consider all possible alignments of $v$. When the beginning of $v$ is not aligned at the beginning of a block in $w$, then it surely does not match: near the end of $v$ we have $n+2$ consecutive letters b, which cannot appear on any other place of a block. Notice also that a negative block in $v$ does not fit to the empty block in $w$ (because of the a on the $(n+1)$-th position of the negative block, which is b in the empty block). The query block in $u$ is surrounded by $s^2+1$ empty blocks in each direction (this is the minimal distance between negative blocks in $v$). Thus when no negative block of $v$ is aligned above the query block of $w$, then some negative block is aligned above an empty block, so it does not match (there is some negative block to the left, and some to the right, and the maximal distance between consecutive negative blocks in $v$ is smaller than $2s^2$). The remaining alignments are such that, for some $i \in \{1, \ldots, s\}$, the $i$-th negative block in $v$ is over the query block in $w$. Then $v$ does not match to $w$ since $v_i$ was not matching to $w'$. This finishes the proof.

We remark that our reduction is motivated by Lemma 3 in [9], although the details of the proof are completely different.

[1] F. Blanchet-Sadri, Algorithmic Combinatorics on Partial Words, Discrete mathematics and its applications, CRC Press, 2008.
URL `http://www.crcpress.com/product/isbn/9781420060928`

[2] F. Neven, T. Schwentick, XPath containment in the presence of disjunction, DTDs, and variables, in: D. Calvanese, M. Lenzerini, R. Motwani (Eds.), Database Theory - ICDT 2003, 9th International Conference, Siena, Italy, January 8-10, 2003, Proceedings, Vol. 2572 of Lecture Notes in Computer Science, Springer, 2003, pp. 312–326. doi:10.1007/3-540-36285-1_21.
URL `http://dx.doi.org/10.1007/3-540-36285-1_21`

[3] F. Neven, T. Schwentick, On the complexity of XPath containment in the presence of disjunction, DTDs, and variables, Logical Methods in Computer Science 2 (3). doi:10.2168/LMCS-2(3:1)2006.
URL `http://dx.doi.org/10.2168/LMCS-2(3:1)2006`

[4] W. Czerwiński, W. Martens, P. Parys, M. Przybyłko, The (almost) complete guide to tree pattern containment, in: T. Milo, D. Calvanese (Eds.), Proceedings of the 34th ACM Symposium on Principles of Database Systems, PODS 2015, Melbourne, Victoria, Australia, May 31 - June 4, 2015, ACM, 2015, pp. 117–130. doi:10.1145/2745754.2745766.
URL `http://doi.acm.org/10.1145/2745754.2745766`

[5] F. Blanchet-Sadri, R. M. Jungers, J. Palumbo, Testing avoidability on sets of partial words is hard, Theor. Comput. Sci. 410 (8-10) (2009) 968–972. doi:10.1016/j.tcs.2008.11.011.
URL `http://dx.doi.org/10.1016/j.tcs.2008.11.011`

[6] B. Blakeley, F. Blanchet-Sadri, J. Gunter, N. Rampersad, On the complexity of deciding avoidability of sets of partial words, Theor. Comput. Sci. 411 (49) (2010) 4263–4271. doi:10.1016/j.tcs.2010.09.006.
URL `http://dx.doi.org/10.1016/j.tcs.2010.09.006`

[7] F. Manea, C. Tiseanu, The hardness of counting full words compatible with partial words, J. Comput. Syst. Sci. 79 (1) (2013) 7–22. doi:10.1016/j.jcss.2012.04.001.
URL `http://dx.doi.org/10.1016/j.jcss.2012.04.001`

[8] M. R. Garey, D. S. Johnson, Computers and intractability: a guide to NP-completeness, Vol. 52, WH Freeman New York, 1979.

[9] G. Miklau, D. Suciu, Containment and equivalence for

a fragment of XPath, J. ACM 51 (1) (2004) 2–45. doi:10.1145/962446.962448.
URL `http://doi.acm.org/10.1145/962446.962448`