Implementing Matching in $\mathcal{ALN}$

Sebastian Brandt* and Hongkai Liu
Theoretical Computer Science,
TU Dresden, Germany
brandt@tcs.inf.tu-dresden.de

Abstract

Although matching in Description Logics (DLs) is theoretically well-investigated, an implementation of a matching algorithm exists only for the DL $\mathcal{ALE}$. The present paper presents an implementation of an existing polynomial time matching algorithm for the DL $\mathcal{ALN}$. Benchmarks using randomly generated matching problems indicate a relatively good performance even on large matching problems. Nevertheless, striking differences are revealed by direct comparison of the $\mathcal{ALN}$- and the $\mathcal{ALE}$-algorithm w.r.t. $\mathcal{FL}_\neg$-matching problems.

1 Motivation

Matching in Description Logics (DLs) has been first introduced by Borgida and McGuinness in the context of the Classic system [5] as a means to filter out irrelevant aspects of large concept descriptions. A matching problem (modulo equivalence) consists of a concept description C and a concept *pattern* D, i.e., a concept description with variables. Matching D against C means finding a substitution of the variables in D by concept descriptions such that C is equivalent to the instantiated concept pattern D.

To some extent, matching can help to find redundancies in or to integrate Knowledge Bases (KBs) [3, 6]. Matching can also be employed for queries over KBs: a domain expert unable to specify uniquely the concept he is looking for in a KB can use a concept pattern to retrieve all those concepts for which a matcher exists. The structural constraints expressible by patterns exceed the capabilities of simple "wildcards" familiar from ordinary searches [7].

Matching algorithms are well-investigated for the DLs $\mathcal{ALN}$, $\mathcal{ALE}$, and respective sublanguages [2, 1]. Only the $\mathcal{ALE}$-matching algorithm, however, has

*Supported by the DFG under grant BA 1122/4-3

Syntax	Semantics	$\mathcal{FL}_\neg$	$\mathcal{ALE}$	$\mathcal{ALN}$
$\top; \bot$	$\Delta^\mathcal{I};\ \emptyset$	x	x	x
$\neg P, P \in N_{\text{con}}$	$\Delta^\mathcal{I} \setminus P^\mathcal{I}$	x	x	x
$C \sqcap D$	$C^\mathcal{I} \cap D^\mathcal{I}$	x	x	x
$\forall r.C$	$\{x \in \Delta^\mathcal{I} \mid \forall y\colon (x,y) \in r^\mathcal{I} \Rightarrow y \in C^\mathcal{I}\}$	x	x	x
$\exists r.C$	$\{x \in \Delta^\mathcal{I} \mid \exists y\colon (x,y) \in r^\mathcal{I} \wedge y \in C^\mathcal{I}\}$		x	
$(\leq n\, r)$, $n \in \mathbb{N}$	$\{x \in \Delta^\mathcal{I} \mid \#\{y \mid (x,y) \in r^\mathcal{I}\} \leq n\}$			x
$(\geq n\, r)$, $n \in \mathbb{N}$	$\{x \in \Delta^\mathcal{I} \mid \#\{y \mid (x,y) \in r^\mathcal{I}\} \geq n\}$			x

Table 1: Syntax and semantics of concept descriptions.

been implemented [8]. In the present paper, we present an implementation of the $\mathcal{ALN}$-matching algorithm defined in [2]. While matching in $\mathcal{ALE}$ is NP-hard, matching in $\mathcal{ALN}$ is polynomial. This relation gives rise to the question whether the implementations of both matching algorithms reflect the difference in computational complexity of the theoretical problems.

In order to answer this question, we have conducted benchmarks on randomly generated matching problems. In addition to testing the $\mathcal{ALN}$-matching algorithm individually, we have directly compared the performance of both matching algorithms, i.e., the existing one for $\mathcal{ALE}$ and the new one for $\mathcal{ALN}$. To this end, randomly generated $\mathcal{FL}_\neg$-matching problems have been used, $\mathcal{FL}_\neg$ being the largest intersection between $\mathcal{ALE}$ and $\mathcal{ALN}$.

The present paper is structured as follows. Basic notions related to the DLs under consideration are defined in Section 2. The existing $\mathcal{ALN}$-matching algorithm is discussed in Section 3. The actual implementation is presented in Section 4. The results of our benchmarks can be found in Sections 4.1 and 4.2.

2 Preliminaries

Concept descriptions are inductively defined with the help of a set of concept *constructors*, starting with a set N_{con} of *concept names* and a set N_{role} of *role names*. In this paper, we consider concept descriptions built from the constructors shown in Table 1. The DL $\mathcal{FL}_\neg$ provides the constructors top-concept ($\top$), bottom-concept ($\bot$), primitive negation ($\neg P$), conjunction ($C \sqcap D$), and value restriction ($\forall r.C$). $\mathcal{ALE}$ extends $\mathcal{FL}_\neg$ by existential restrictions ($\exists r.C$) while $\mathcal{ALN}$ extends $\mathcal{FL}_\neg$ by number restrictions ($(\leq n\, r)$ and $(\geq n\, r)$).

In order to define matching problems we also need to introduce *concept patterns*. These are defined w.r.t. a finite set N_{var} of *concept variables* distinct from N_{con} and N_{role}. Concept patterns are an extension of concept descriptions in the sense that they allow for primitive concepts $A \in N_{\text{con}}$ *and* concept variables

$X \in N_{\text{var}}$ as atomic constructors. The only restriction is that concept variables may not be negated.

A concept description C_1 is *subsumed* by a description C_2 ($C_1 \sqsubseteq C_2$) iff $C_1^{\mathcal{I}} \subseteq C_2^{\mathcal{I}}$ holds for all interpretations $\mathcal{I}$. The concept descriptions C_1 and C_2 are *equivalent* ($C_1 \equiv C_2$) iff they subsume each other.

An $\mathcal{L}$-substitution σ is a mapping from N_{var} into the set of all $\mathcal{L}$-concept descriptions. Substitutions are extended to concept patterns by induction on the structure of the pattern, modifying only the occurrences of variables in the pattern. The notion of subsumption is extended to $\mathcal{L}$-substitutions as follows. An $\mathcal{L}$-substitution σ is subsumed by an $\mathcal{L}$-substitution τ ($\sigma \sqsubseteq \tau$) iff $\sigma(X) \sqsubseteq \tau(X)$ for all $X \in N_{\text{var}}$. With these preliminaries we can define matching problems.

Definition 1 *Let C be an $\mathcal{L}$-concept description and D be an $\mathcal{L}$-concept pattern. Then, $C \equiv^? D$ is an $\mathcal{L}$-matching problem*[1]*. An $\mathcal{L}$-substitution σ is a* matcher *iff $C \equiv \sigma(D)$. A matcher σ is the* least matcher *to $C \equiv^? D$ iff for every matcher τ to $C \equiv^? D$ it holds that $\sigma \sqsubseteq \tau$.*

3 Matching in $\mathcal{ALN}$

Matching in $\mathcal{ALN}$ has been well-investigated in [2]. In particular, it has been shown that solvable $\mathcal{ALN}$-matching problems always have exactly one least matcher unique up to equivalence that can be computed in polynomial time. As the focus of this work is on implementation rather than theory, we will present the relevant matching algorithm only as detailed as necessary. For further details, see [2]. The algorithm relies on the so-called $\mathcal{FL}_0$-normal form of $\mathcal{ALN}$-concept descriptions which must be introduced first.

Consider an arbitrary $\mathcal{ALN}$-concept description C over N_{con}, N_{role}, and over sets $N_{\geq}$ and $N_{\leq}$ of number restrictions of the form $(\geq n\, r)$ and $(\leq n\, r)$, respectively. Exhaustively applying the equivalence $\forall r.(C_1 \sqcap C_2) \equiv \forall r.C_1 \sqcap \forall r.C_2$ from left to right, we can represent C as a conjunction of concepts of the form $\forall r_1. \cdots \forall r_n.\Pi$, where Π is the bottom-concept, a (negated) primitive concept, or a number restriction. Abbreviating $\forall r_1. \cdots \forall r_n.\Pi$ by $\forall r_1 \ldots r_n.\Pi$, we can interpret $r_1 \ldots r_n$ as a word over the alphabet N_{role}. Collecting all these words separately for the bottom-concept, for every (negated) primitive concept, and for every number restriction, we obtain a representation of C of the form

$$C \equiv \bigsqcap_{\Pi \in \{\bot\} \cup N_{\text{con}} \cup \{\neg P \mid P \in N_{\text{con}}\} \cup N_{\geq} \cup N_{\leq}} \forall U_{\Pi}.\Pi,$$

where every U_{Π} is a formal language over N_{role}. Note that the occurrence of Π on top-level can be represented by including ε in the corresponding role language.

[1]In contrast to [2] we do not introduce matching modulo equivalence and matching modulo subsumption separately. Note that $C \sqsubseteq \sigma(D)$ iff $C \equiv C \sqcap \sigma(D)$.

Moreover, if Π does not occur in C at all then $U_\Pi = \emptyset$. $\mathcal{ALN}$-concept patterns can be represented in $\mathcal{FL}_0$-normal form by treating variables like primitive concepts. Hence, it suffices to extend the above representation by role languages U_X for every variable $X \in N_{\text{var}}$. The following example illustrates this.

Example 2 Let $N_{\text{con}} := \{A\}$, $N_{\text{role}} := \{r, s\}$, $N_{\geq} := \{(\geq 3\, r)\}$, $N_{\leq} := \emptyset$, and $N_{\text{var}} := \{X, Y\}$. Then the pattern $D := A \sqcap \forall r.\bot \sqcap \forall s.(\forall r.A \sqcap (\geq 3\, r) \sqcap X) \sqcap X$ can be represented in $\mathcal{FL}_0$-normal form as

$$\forall\{r\}.\bot \sqcap \forall\{\varepsilon, sr\}.A \sqcap \forall\emptyset.\neg A \sqcap \forall\{s\}.(\geq 3\, r) \sqcap \forall\{\varepsilon, s\}.X \sqcap \forall\emptyset.Y.$$

By means of the $\mathcal{FL}_0$-normal form, a matching problem can be viewed as a problem over formal languages. In order to simplify the presentation of the $\mathcal{ALN}$-matching algorithm, we introduce two auxiliary functions on formal languages.

Definition 3 *For arbitrary formal languages U, V over N_{role} and $r \in N_{\text{role}}$, define $U \ominus V := \bigcap_{u \in U}\{v' \mid uv' \in V\}$ and $U \cdot r^{-1} := \{u' \mid u'r \in U\}$.*

We can now introduce one main result from [2] which shows how the least matcher to a solvable $\mathcal{ALN}$-matching problem can be constructed. To simplify notation, let $\neg N_{\text{con}} := \{\neg A \mid A \in N_{\text{con}}\}$.

Lemma 4 *Let $C \equiv^? D$ be an $\mathcal{ALN}$-matching problem over N_{con}, N_{role}, and over number restrictions $N_{\geq}$ and $N_{\leq}$. Let the $\mathcal{FL}_0$-normal form of C be represented by role languages of the form U_Π quantified over every $\Pi \in \{\bot\} \cup N_{\text{con}} \cup \neg N_{\text{con}} \cup N_{\geq} \cup N_{\leq} \cup N_{\text{var}}$. Analogously, let D be represented by role languages V_Π.*

Then, either $C \equiv^? D$ is not solvable or it has a least matcher σ that assigns to each variable X the concept description $\sigma(X)$ defined by

$$\sigma(X) := \forall W_\bot^X.\bot \;\sqcap \bigsqcap_{\Pi \in N_{\text{con}} \cup \neg N_{\text{con}} \cup N_{\geq} \cup N_{\leq}} \forall((V_X \ominus W_\Pi^X) \setminus (V_X \ominus E_C)).\Pi,$$

where $E_C = \{w \in N_{\text{role}}^* \mid C \sqsubseteq \forall w.\bot\}$, $W_\bot^X$ is a role language of polynomial size in C with $W_\bot^X \cdot N_{\text{role}}^* = V_X \ominus E_C$, and all other role languages of the form W_Π^X are defined as follows.

$$W_\Pi^X := \begin{cases} U_\Pi \cup E_C & \text{if } \Pi \in N_{\text{con}} \cup \neg N_{\text{con}} \\ \bigcup_{m \geq n} U_{(\geq m r)} \cup E_C & \text{if } \Pi =: (\geq n\, r) \in N_{\geq} \\ \bigcup_{m \leq n} U_{(\leq m r)} \cup E_C \cdot r^{-1} & \text{if } \Pi =: (\leq n\, r) \in N_{\leq} \end{cases}$$

There are two obvious strategies to decide whether the substitution σ defined above actually solves the matching problem $C \equiv^? D$. We might either ascertain the solvability of $C \equiv^? D$ *before* computing σ, or we might compute σ first

and decide the equivalence $C \equiv \sigma(D)$ *afterwards*. In [2], the former strategy is taken: a system of formal language equations, so-called 'solvability equations', is proposed which is solvable iff $C \equiv^? D$ is solvable. To decide solvability of these equations, however, necessitates computing exactly those role languages which occur in the $\mathcal{FL}_0$-normal form of $\sigma(X)$ constructed in Lemma 4.

As the second strategy is computationally equivalent but more easily explained, we deviate from the original in [2] by computing a candidate solution first and testing for equivalence afterwards. To this end, we utilize a characterization of equivalence from [2] based on $\mathcal{FL}_0$-normal forms.

Lemma 5 *Let C_1 and C_2 be $\mathcal{ALN}$-concept descriptions over N_{con}, N_{role}, and over number restrictions $N_{\geq}$ and $N_{\leq}$. Let the $\mathcal{FL}_0$-normal forms of C_1 and C_2 be represented by role languages of the form U_Π and V_Π, respectively. Then, $C \equiv D$ iff for every $\Pi \in N_{\text{con}} \cup \neg N_{\text{con}}$, for every $(\geq n\, r) \in N_{\geq}$, and for every $(\leq n\, r) \in N_{\leq}$ it holds that*

$$E_{C_1} = E_{C_2} \tag{$\bot$}$$

$$U_\Pi \cup E_{C_1} = V_\Pi \cup E_{C_2} \tag{Π}$$

$$\bigcup_{m \geq n} U_{(\geq m r)} \cup E_{C_1} = \bigcup_{m \geq n} V_{(\geq m r)} \cup E_{C_2}$$

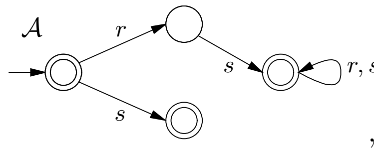
$$\bigcup_{m \leq n} U_{(\leq m r)} \cup E_{C_1} \cdot r^{-1} = \bigcup_{m \leq n} V_{(\leq m r)} \cup E_{C_2} \cdot r^{-1},$$

where $E_{C_i} = \{w \in N_{\text{role}}^* \mid C_i \sqsubseteq \forall w.\bot\}$ for $i = 1, 2$.

Informally, the $\mathcal{ALN}$-matching algorithm can now be described as follows. Upon input $C \equiv^? D$, (i) transform C and D into $\mathcal{FL}_0$-normal form, (ii) construct the candidate solution σ defined in Lemma 4, and (iii) test whether C and $\sigma(D)$ satisfy the formal language equations shown in Lemma 5. If they do, return the least matcher σ, otherwise return 'fail'. It remains to provide a method by which to solve Steps (ii) and (iii) in polynomial time.

To this end, so-called 'tree-like automata' [2], can be utilized. Intuitively, these are deterministic finite automata whose structure differs from a tree only in that they either have ordinary leaves or leaves with an r-transition to themselves for every $r \in N_{\text{role}}$. Consider the following example.

Example 6 Let $N_{\text{role}} = \{r, s\}$. Then the role language $\{\varepsilon, s\} \cup \{rs\} \cdot N_{\text{role}}^*$ can be represented by a tree-like automaton $\mathcal{A}$ of the form



,

where $\longrightarrow$ denotes the initial state and double circles denote final states.

It has been shown in [2] that tree-like automata have the following properties.

- A tree-like automaton $\mathcal{A}$ that accepts E_C can be constructed in polynomial time in the size of C. From $\mathcal{A}$, a language U of polynomial size in C with $E_C = U \cdot N_{\text{role}}^*$ can be constructed in linear time.
- The operations union, intersection, and complement on treelike automata can be defined in such a way that the size of the resulting automaton does not exceed the maximum of the sizes of the input automata. Moreover, all operations can be performed in linear time.
- If U, V, W are finite languages, then a tree-like automaton that accepts $U \ominus (V \cup W \cdot N_{\text{role}}^*)$ can be constructed in polynomial time in the size of the input.

As a consequence, tree-like automata can be used to construct the candidate solution σ defined in Lemma 4 in polynomial time. It remains to show how tree-like automata can be used to test whether σ actually is a solution.

Consider a matching problem $C \equiv^? D$ in $\mathcal{FL}_0$-normal form with a candidate solution σ as defined in Lemma 4. Instantiating the entire system of equations from Lemma 5 by C and $\sigma(D)$ is beyond the scope of this paper. Nevertheless, as a typical example, we discuss Equation (Π) defined for every $\Pi \in N_{\text{con}} \cup \neg N_{\text{con}}$. Inserting the role languages from C and $\sigma(D)$, we obtain the following equation.

$$U_\Pi \cup E_C = V_\Pi \cup E_{\sigma(D)} \cup \bigcup_{X \in N_{\text{var}}} V_X \cdot \big(V_X \ominus (U_\Pi \cup E_C)\big) \qquad (*)$$

Assume that Equation ($\bot$) has already been tested, i.e., $E_C = E_{\sigma(D)}$. By definition of $\ominus$, the union over all $X \in N_{\text{var}}$ on the right-hand side of ($*$) is always a subset of the left-hand side of the equation. Hence, Equation ($*$) holds iff (i) $V_\Pi \subseteq U_A \cup E_C$ and (ii) for all $u \in U_\Pi$ either (iia) $u \in V_X \cup E_{\sigma(D)}$ or (iib) $u \in V_X \cdot V_X \ominus U_\Pi$ or (iic) $u \in V_\Pi \cdot V_X \ominus E_C$. Condition (i) can be decided by testing the tree-like automaton of $V_\Pi \cap \overline{(U_\Pi \cup E_C)}$ for emptyness. For Condition (iia), merely the word problem w.r.t. the tree-like automaton for $V_\Pi \cup E_{\sigma(D)}$ must be decided for every $u \in U_\Pi$. Since there is no concatenation defined for tree-like automata, the remaining Conditions (iib) and (iic) cannot be solved by means of one single treelike automaton. Nevertheless, one can show that $u \in V_X \cdot V_X \ominus U_\Pi$ iff $\{u\} \ominus V_X \cap V_X \ominus U_\Pi$ is not empty, which again can be decided by a tree-like automaton in polynomial time. Case (iic) is analogous. The other equations from Lemma 4 can be decided similarly.

This completes our overview of matching in $\mathcal{ALN}$. In the following section, we will explain how the theoretical algorithm described above can be implemented.

4 Implementation

In order to implement the $\mathcal{ALN}$-matching algorithm introduced previously, appropriate data structures for the representation of concept descriptions, concept patterns, and tree-like automata are necessary.

As the algorithm is defined w.r.t. the role languages of the $\mathcal{FL}_0$-normal form of its input, it seems expedient to begin by translating the input matching problem into an array of sets of lists over symbols, the symbols representing the alphabet N_{role}. Our data structure for tree-like automata resembles the inductive representation of trees: a vector the elements of which are either atomic objects or again vectors. In our case, we only additionally have to discriminate non-final from final nodes and ordinary leaves from those accepting N_{role}^*. In order to decide word-problems more quickly, vectors representing non-leaf nodes are implemented as arrays instead of lists.

The overall strategy of the implementation corresponds to the steps described in Section 3. As implementation language, we chose Common LISP because it proved well-suited to realize our representation of tree-like automata. Moreover choosing LISP makes our implementation compatible to the system Sonic [9] which provides an interface between the KB editor OilEd [4] and non-standard reasoning services. This may help to make our algorithm available to users.

4.1 Benchmarks

In order to test the performance of our implementation on a sufficiently large set of data, we had to resort to randomly generated matching problems. A similar approach was used for the implementation of an $\mathcal{ALE}$-matching algorithm [8].

Randomly generating C and D independently of each other makes it very unlikely that a matcher for $C \equiv^? D$ exists. Hence, we randomly generate a concept C and then construct a concept pattern D from C by randomly replacing sub-concepts of C by variables. Matching problems obtained thus are not necessarily solvable because of multiple occurrences of the same variable. As a simple example, consider $C := \forall r.A \sqcap \forall s.B$ and $D := \forall r.X \sqcap \forall s.X$. Then, $C \equiv^? D$ has no solution. Note also that assuming the concept pattern D to be smaller than C seems justified especially when viewing matching as querying over KBs.

The generated random matching problems were influenced by a vector of probabilities controlling the depth and width of the resulting concept C as well as the frequency of the different constructors available in $\mathcal{ALN}$ and the variables in D. Our benchmarks comprise a total of about 22,000 matching problems in 220 groups, each of which was generated with a unique probability vector. Moreover, we have generated another 12,000 matching problems which, though random, were constructed to be always solvable. The maximum problem size, i.e., the sum of the sizes of C and D, was limited by 1000. The benchmarks were

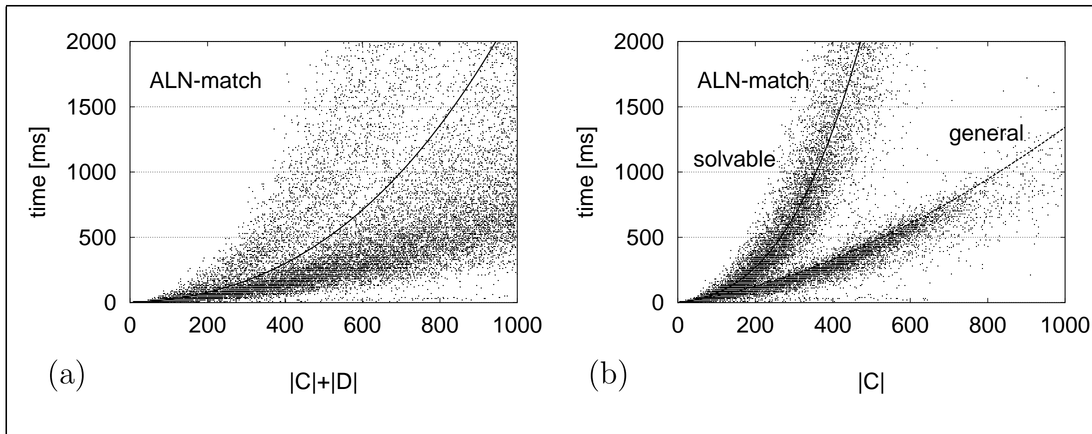


Figure 1: Benchmarks for matching in $\mathcal{ALN}$

measured on a standard PC with one 1.7GHz Pentium-4 processor and 512MB of memory. Computing overall averages, the algorithm takes 0.8 seconds to solve a matching problem of size 528 with D being two thirds the size of C.

Figure 1 gives a more detailed account of our findings. Diagram (a) shows the result of our benchmarks as a scatterplot together with a fitting function computed by the least-squares method. One dot in the diagram represents one matching problem $C \equiv^? D$. In the diagram, the horizontal position of every dot represents the sum of the sizes C and D while the vertical position represents the time necessary to solve the problem.

The fitting function in Figure 1(a) not only matches the overall average fairly well, but also shows the general trend of the expected computation time for larger problems. A problem of size, e.g., 800 increases the computation time to about 1.5 seconds. Nevertheless, the 'darker' cluster below the fitting function indicates that the majority of the problems are solved in less than one second.

Astonished by the strong dispersion of the scatterplot in Figure 1(a), we have rearranged the plot so that the horizontal position of every dot representing a matching problem $C \equiv^? D$ is determined by the size of C alone, thus ignoring the size of D. This rearrangement produces the scatterplot in Figure 1(b).

Comparing diagrams (a) and (b), the first immediate observation is that the the size of C influences the computation time stronger than the size of D—although on average the size of D is two thirds the size of C. Moreover, we observe one cluster of simpler matching problems and another cluster of 'hard' ones, where a problem of size 400 on average already seems to take 2 seconds to solve. Analysis of our data revealed that the 'hard' cases comprise exactly those problems which, though random, were designed to be solvable.

As we do not have the means to verify these findings by matching problems from realistic applications, we cannot rule out that the above findings are specific to randomly generated matching problems. Nevertheless, it seems expedient

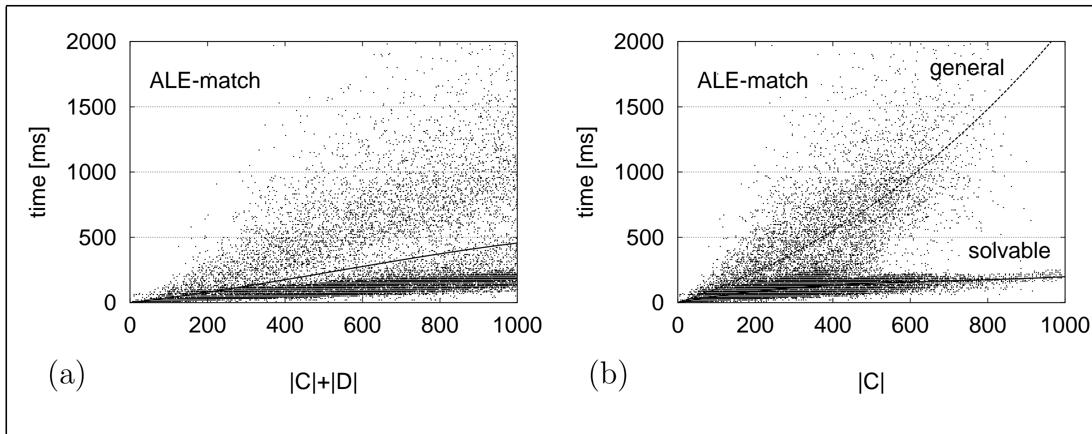


Figure 2: Benchmarks for the $\mathcal{ALE}$-matching algorithm in $\mathcal{FL}_\neg$

to aim future optimizations of the $\mathcal{ALN}$-matching algorithm at improving the computation time for solvable matching problems.

4.2 A comparison to the $\mathcal{ALE}$-matching algorithm

The fact that a matching algorithm for the DL $\mathcal{ALE}$ has already been implemented offers the unique opportunity to compare the $\mathcal{ALN}$- to the $\mathcal{ALE}$-matching algorithm head-to-head on $\mathcal{FL}_\neg$-matching problems, the largest intersection of $\mathcal{ALN}$ and $\mathcal{ALE}$. This comparison might be interesting for two reasons. Firstly, both algorithms take a totally different approach to solving a matching problem $C \equiv^? D$. While the $\mathcal{ALN}$-algorithm solves a system of formal language equations, the $\mathcal{ALE}$-algorithm tries to construct homomorphisms from the description tree of D into that of C. Secondly, the $\mathcal{ALN}$-algorithm exploits the fact that an $\mathcal{ALN}$-matching problem has at most one solution while the $\mathcal{ALE}$-algorithm might look for several ones.

For our comparison, we have generated a set of 34.000 $\mathcal{FL}_\neg$-matching problems in the manner described above. The results for the $\mathcal{ALN}$-algorithm are similar to the ones discussed in Section 4.1. On average, a problem of size 539 was solved in 1.2 seconds by the $\mathcal{ALN}$-algorithm, compared to just 0.25 seconds by the algorithm for $\mathcal{ALE}$. The resulting scatterplots for the $\mathcal{ALN}$-algorithm are not shown here because they closely resemble the ones in Figure 1. The scatterplots for the $\mathcal{ALE}$-algorithm are shown in Figure 2.

The plot in Figure 2(a) shows that the majority of matching problems is solved in less than 0.25 seconds with relatively fewer cases strongly deviating upwards. Moreover, the fitting function indicates that even a problem of size 1000 is usually solved in about 0.5 seconds.

The discrimination by ordinary matching problems and those designed to be solvable, see Figure 2(b), shows that our findings from the $\mathcal{ALN}$-algorithm are

exactly *reversed*. The $\mathcal{ALE}$-algorithm apparently had no difficulty with solvable matching problems while the 'hard' cases are comprised of those problems of which many have no solution.

5 Conclusion

In the present paper, we have presented an implementation of the $\mathcal{ALN}$-matching algorithm defined in [2]. Upon input $C \equiv^? D$, the algorithm first computes a candidate solution σ and verifies its validity afterwards. More precisely, the algorithm reduces matching problems to problems over formal languages and decides them in polynomial time with the help of tree-like automata.

Our benchmarks show that even large $\mathcal{ALN}$-matching problems can be solved relatively quickly. Analysis indicates that solvable matching problems tend to consume much more time than those without a solution. This suggests for potential optimizations to aim at constructing solutions more quickly and not at trying to identify unsolvable problems earlier.

The validity of our findings is weakened by the fact that only randomly generated data was available for benchmarks. It is an open questions whether both implementations, the one for $\mathcal{ALN}$ as well as the one for $\mathcal{ALE}$, behave similar on matching problems from realistic applications. Nevertheless, our comparison suggest that without major optimization the $\mathcal{ALE}$-matching algorithm seems the more auspicious starting point for an extension to matching in $\mathcal{ALEN}$.

References

[1] F. Baader and R. Küsters. Matching in description logics with existential restrictions. In *Proc. of KR2000*, Morgan Kaufmann Publishers, 2000.

[2] F. Baader, R. Küsters, A. Borgida, and D. McGuinness. Matching in description logics. *Journal of Logic and Computation*, 9(3):411–447, 1999.

[3] F. Baader and P. Narendran. Unification of concept terms in description logics. In *Proc. of ECAI-98*, John Wiley & Sons Ltd, 1998.

[4] S. Bechhofer, I. Horrocks, C. Goble, and R. Stevens. OilEd: A reason-able ontology editor for the semantic Web. *Lecture Notes in Computer Science*, 2174, 2001.

[5] A. Borgida, R. J. Brachman, D. L. McGuinness, and L. A. Resnick. CLASSIC: A Structural Data Model for Objects. In *Proc. of ACM SIGMOD*, ACM Press, 1989.

[6] A. Borgida and R. Küsters. What's not in a name: Some Properties of a Purely Structural Approach to Integrating Large DL Knowledge Bases. In *Proc. of DL2000*, CEUR-WS, 2000.

[7] S. Brandt and A.-Y. Turhan. Using non-standard inferences in description logics—what does it buy me? In *Proc. of KIDLWS'01*, CEUR-WS, 2001.

[8] S. Brandt. Implementing matching in $\mathcal{ALE}$—first results. In *Proc. of DL2003*, CEUR-WS, 2003.

[9] A.-Y. Turhan and C. Kissig. Sonic—non-standard inferences go OilEd. In *Proc. of IJCAR'04*, Springer-Verlag, 2004. 2004.