FILO: unification solver for FL_0 (Extended Abstract)

Barbara Morawska[1], Sławomir Kost[1], Dariusz Marzec[1] and Michał Henne[1]

[1]University Of Opole, Plac Kopernika 11a, 45-040 Opole

Abstract

In this paper, we present FILO, the first application that decides the unification problem for the description logic $\mathcal{FL}_0$. This is a restricted description logic that allows only conjunction, the top constructor and value restrictions to construct complex concepts from a set of concept names (unary predicates) and role names (binary predicates). Unification of concepts in Description Logics has been proposed as a non-standard reasoning task that can help eliminate redundancies in ontologies. The unification problem in $\mathcal{FL}_0$ is ExpTime-complete. FILO is based on an algorithm that is exponential only in the worst case. The algorithm used by FILO is based on the one described in [1].

Keywords

Description Logic, Unification, Automated reasoning

1. Introduction

The description logic $\mathcal{FL}_0$ is a member of a family of lightweight description logics with restricted expressive power. It provides only the *intersection (conjunction)* constructor, *top* concept, and value restrictions of the form $\forall r.C$, where C is a concept and r is a role name. Here we present an implementation of a unification algorithm for concepts in $\mathcal{FL}_0$.

The unification problem in DLs was, in fact, first defined for this logic. It was shown in [2] to be an ExpTime-complete problem. The algorithm presented there reduces the problem to the emptiness problem for a root-to-frontier automaton on trees. Since the automaton is of size exponential in the size of a unification problem, this establishes the exponential upper bound. The lower bound was established in the same paper by a reduction from the intersection problem for root-to-frontier automata on trees.

For some reasons, this algorithm was never implemented. One possible reason is that researchers' focus shifted to another small DL, namely to $\mathcal{EL}$. The small description logic $\mathcal{EL}$ does not have a value restriction constructor, but provides instead an existential restriction: $\exists r.C$, which expresses the requirement that an object be related by a relation r to another object that satisfies the concept C.

In [1], a new unification algorithm was developed. It is based on a reduction to the problem of finding a special kind of model for a set of anti-Horn clauses. Now, that algorithm has been revised and adapted for implementation [3]. This implementation, called FILO, operates on unification problems in the form of ontology files, where variables are marked with the `_var` suffix. FILO detects that if an input problem is unifiable, then it outputs a unifier that can be extracted from its computations.

FILO joins a family of similar applications, such as UEL [4] which solves unification problems in the description logic $\mathcal{EL}$, $\mathcal{FL}_0$wer [5], a subsumption decider for $\mathcal{FL}_0$ with TBox, CEL [6] and JCEL [7] subsumption deciders for $\mathcal{EL}$ with TBox, and others. These systems play an important role in various knowledge representation reasoning problems.

An extended version of this paper can be found in [3].

This research is part of the project No 2022/47/P/ST6/03196 within the POLONEZ BIS programme co-funded by the National Science Centre and the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No. 945339. For the purpose of Open Access, the author has applied a CC-BY public copyright licence to any Author Accepted Manuscript (AAM) version arising from this submission.

DL 2025: 38th International Workshop on Description Logics, September 3–6, 2025, Opole, Poland

barbara.morawska@uni.opole.pl (B. Morawska); skost@math.uni.opole.pl (S. Kost); dariusz.marzec@uni.opole.pl (D. Marzec); michal.henne@uni.opole.pl (M. Henne)

2. The description logic $\mathcal{FL}_0$

Complex concepts in $\mathcal{FL}_0$ are constructed from a countable set of concept names (unary predicates) $\mathbf{N}$ and a countable set of role names (binary relations) $\mathbf{R}$ according to the following grammar: $C ::= A \mid C \sqcap C \mid \forall r.C \mid \top$, where $A \in \mathbf{N}$ and $r \in \mathbf{R}$.

An interpretation I is a pair $(\Delta^I, \cdot^I)$, where Δ^I is a non-empty domain and $\cdot^I$ is an interpretation function. The interpretation of concept names ($A^I \subseteq \Delta^I$) and role names ($r^I \subseteq \Delta^I \times \Delta^I$) is extended to all concepts in the standard way: $(C_1 \sqcap C_2)^I = C_1^I \cap C_2^I$, $(\forall r.C)^I = \{e \in \Delta^I \mid \forall_{d \in \Delta^I}((e,d) \in r^I \implies d \in C^I)\}$, $\top^I = \Delta^I$.. We define $C \sqsubseteq D$ iff for every interpretation I, $C^I \subseteq D^I$ and $C \equiv D$ iff $C \sqsubseteq D$ and $D \sqsubseteq C$.

We write $\forall r_1.\forall r_2. \ldots \forall r_n.A$ as $\forall v.A$, where v is a word over $\mathbf{R}$, and call such a concept *a particle*. With respect to the equivalence, one can easily observe that the intersection of concepts is associative, commutative, idempotent and has $\top$ as its unit element. The value restriction behaves as a homomorphism: $\forall r.(E \sqcap F) \equiv \forall r.E \sqcap \forall r.F$. Thus, each concept can be identified as a set of particles, and the empty set is $\top$.

In $\mathcal{FL}_0$, subsumption between two concepts C and D can be decided in polynomial time.

Lemma 1. *Let $C = P_1 \sqcap \cdots \sqcap P_m$ and $D = P'_1 \sqcap \cdots \sqcap P'_n$.*

1. *$C \sqsubseteq D$ iff for every $1 \leq i \leq n$, $P_1 \sqcap \cdots \sqcap P_m \sqsubseteq P'_i$;*
2. *for every $1 \leq i \leq n$, $P_1 \sqcap \cdots \sqcap P_m \sqsubseteq P'_i$ iff $P'_i \in \{P_1, \cdots, P_m\}$.*

Proof. Both these statements follow from the properties of subsumption that establish a partial order with respect to the sets of particles. □

In order to define a unification problem in $\mathcal{FL}_0$, we divide the set of concept names $\mathbf{N}$ into two disjoint sets: constants, denoted $\mathbf{C}$ and variables, denoted $\mathbf{Var}$. Variables may be substituted by concepts, and constants cannot be substituted.

A substitution σ is a mapping initially defined for $\mathbf{Var}$, assigning to them possibly complex concepts. It is then extended to all concepts in the usual way: $\sigma(A) := A$, where $A \in \mathbf{C}$, $\sigma(E \sqcap F) := \sigma(E) \sqcap \sigma(F)$, $\sigma(\forall r.E) := \forall r.\sigma(E)$, $\sigma(\top) := \top$.

The unification problem is then defined by its input and output as follows.

Input: $\Gamma = \{C_1 \sqsubseteq^? D_1, \ldots, C_n \sqsubseteq^? D_n\}$, where $C_1 \ldots C_n, D_1 \ldots D_n$ are $\mathcal{FL}_0$-concepts constructed over constants and variables. We call $C \sqsubseteq^? D \in \Gamma$ an *input goal*. Due to Lemma 1, we can assume that for each input goal $C \sqsubseteq^? D$, D is a particle.

Output: "true" if there is a substitution γ such that $\gamma(C_1) \sqsubseteq \gamma(D_1), \ldots, \gamma(C_n) \sqsubseteq \gamma(D_n)$ and "false" otherwise. The substitution γ is called *a unifier* or *a solution* of Γ.

3. Solver

FILO is an application written in Java using the OWL API and Maven for dependency management. As of now it is a standalone application (FILO.jar). The compiled file is available at https://unifdl.cs.uni.opole.pl/unificator-app-for-the-description-logic-fl_0/ and the source files are available in a public GitHub repository https://github.com/barbmor/FILO.

A unification problem which can be solved by FILO must be provided in the form of an ontology file. Such an ontology may be created using the ontology editor Protégé[1]. The concept names in such an ontology are treated by FILO as constants, unless they have the `_var` suffix. Input goals are defined as general class axioms or as concept subsumptions expressed through class hierarchy statements. FILO recognizes concept names (constants and variables), intersections of concepts, value restrictions and the $\top$ constructor. It reports an error if an unsupported constructor is encountered while reading the input file.

[1] https://protege.stanford.edu/

4. Overview of FILO computation

FILO is based on the algorithm presented in [1], but at first glance, it may seem quite different. This is because the algorithm there is formulated for a class of first-order clauses, solving the problem of the existence of a *finite Herbrand model* for this class.

Nevertheless, the algorithm in [1] can be easily modified to work directly on $\mathcal{FL}_0$-subsumptions, without translating them into first-order clauses. Therefore, in [3], the correctness of the algorithm is re-proved in the notation of $\mathcal{FL}_0$. The main differences concern the way *shortcuts* are defined. Here, we treat them as sets of variables, whereas in [1], they are pairs consisting of one predicate and a set of predicates. There are also differences in how shortcuts are computed: here, we check sets of variables, while in [1], shortcuts are computed via *possibilities*. Other differences stem from programming considerations, for example, the way we treat choices for variables. All these details are addressed in the correctness proofs in [3].

4.1. Flattening I – reading input

During this stage, the internal structures representing concepts from the input ontology are created. Concept intersections from the input are represented by sets of *flat particles* of the form $\forall r.C$, where C is a variable or a constant. The internal representation of the input after Flattening I is called a *Filo model*.

Example 1. *Let an input goal in the notation of $\mathcal{FL}_0$ be: $\forall rss.A_1 \sqcap \forall rsr.A_2 \sqcap \forall r.X_{var} \sqcap \forall rr.A_2 \equiv^? \forall r.A_1 \sqcap \forall rr.A_2 \sqcap \forall rs.X_{var}$ (Example 16 in FILO). After the Flattening I stage, we get the following set of equivalences:*

$$Var2 \equiv \forall s.A_1, \quad Var5 \equiv \forall r.A_2, \quad Var0 \equiv \forall r.A_2, \quad Var6 \equiv \forall s.X_{var},$$
$$Var4 \equiv \forall s.Var2, \quad Var1 \equiv \forall r.A_2, \quad Var3 \equiv \forall s.Var1,$$
$$\forall r.Var4 \sqcap \forall r.X_{var} \sqcap \forall r.Var0 \sqcap \forall r.Var3 \equiv \forall r.Var5 \sqcap \forall r.A_1 \sqcap \forall r.Var6.$$

4.2. Main loop

After successfully reading the input and creating a Filo model, FILO enters a loop: *for each constant* in the model. If there are no constants, only variables, then FILO returns success and the problem has a solution sending all variables to $\top$. Otherwise, FILO attempts to solve the problem for each constant *separately*–in the manner presented in [2]–then combines solutions into one using intersection constructor. If it returns **failure** for any one of them, it breaks the main loop immediately returning **failure**. Such a problem is not unifiable. Let us fix the *current constant* A_2.

4.3. Flattening II

Depending on the current constant, FILO performs further flattening. The goal of this stage is to obtain *flat subsumptions* of the form $X_1 \sqcap \cdots \sqcap X_n \sqsubseteq^? Y$, where all of the concept names are variables or the current constant, and additional *increasing subsumptions* of the form $X \sqsubseteq^? \forall r.X^r$. The set of flat subsumptions and increasing subsumptions is called a *generic goal*.

Example 2. *Following Example 1, flattening II yields the generic goal w.r.t. the constant A_2:*
***flat subsumptions**: $Var2 \equiv \top$, $Var5^r \equiv A_2$, $Var0^r \equiv A_2$, $Var6^s \equiv X_{var}$, $Var4 \equiv \top$, $Var1^r \equiv A_2$, $Var3^s \equiv Var1$, $Var4 \sqcap X_{var} \sqcap Var0 \sqcap Var3 \sqsubseteq Var5$, $Var4 \sqcap X_{var} \sqcap Var0 \sqcap Var3 \sqsubseteq Var6$, $Var5 \sqcap Var6 \sqsubseteq Var4$, $Var5 \sqcap Var6 \sqsubseteq X_{var}$, $Var5 \sqcap Var6 \sqsubseteq Var0$, $Var5 \sqcap Var6 \sqsubseteq Var3$;*
***increasing subsumptions**: $Var5 \sqsubseteq \forall r.Var5^r$, $Var0 \sqsubseteq \forall r.Var0^r$, $Var6 \sqsubseteq \forall s.Var6^s$, $Var1 \sqsubseteq \forall r.Var1^r$, $Var3 \sqsubseteq \forall s.Var3^s$, where $Var5^r, Var0^r, Var6^s, Var1^r, Var3^s$ are* decomposition variables. *Note that $Var4$ is $\top$ since $Var2$ is.*

4.4. Choice and Implicit Solver

Now FILO has to make choices for each variable predicting their values in a solution: TOP if it is an empty conjunction, or CONSTANT if some of the particles contained in a solution is the current constant, or NEITHER otherwise. If there are no variables, only the current constant, FILO checks solvability of the generic goal by Implicit Solver (it must return either **success** or **failure**). Otherwise, FILO checks all *consistent* choices for all variables in the generic goal, i.e. choices in which the decomposition variables and their parents satisfy appropiate relations. Hence, here FILO enters another loop.

Implicit Solver checks which subsumptions are solved by the current choice for variables. It may also detect contradictions. If this is the case, FILO aborts the computation, returns **failure** for the current choice and proceeds to check the next choice and starting from the generic goal, to construct another goal again. If the set of flat subsumptions is empty, FILO returns **success** for the current constant. If there are no choices left, the loop for choices is ended and the program returns **failure**. Let us fix the current consistent choice.

Example 3. *Continuing with Example 2, FILO searches for a consistent choice, and finds:* $\{Var2, Var4\} \mapsto TOP, \{Var5^r, Var0^r, Var1^r\} \mapsto CONSTANT$ *and the rest of variables is NEITHER.*

For this choice, $Var2 \equiv \top, Var5^r \equiv A_2, Var0^r \equiv A_2, Var4 \equiv \top, Var1^r \equiv A_2, Var5 \sqcap Var6 \sqsubseteq Var4$ *are trivially satisfied. For the last subsumption, note that its right-hand is TOP. The rest of subsumptions is already flat.*

4.5. Shortcuts

If the set of flat subsumptions is not empty, FILO starts computation with *shortcuts*, i.e. sets of variables that *satisfy* all the flat subsumptions of the goal. It enters a loop computing shortcuts. Now, FILO attempts to extend the set of already computed shortcuts using the so-called *resolving relation*: S is resolved by S' w.r.t. a role name r iff $X^r \in S$ implies $X \in S'$ and if Y^r is defined for $Y \in S'$, then $Y^r \in S$. If the *initial shortcut*, i.e. the set of the form $\{X \mid X$ is assigned the choice CONSTANT$\}$, is computed, this means **success** for the current constant–FILO aborts computing new shortcuts. Otherwise, if the initial shortcut cannot be reached, it means **failure** for the current choice.

Example 4. *Continuing with Example 3, a shortcut* of height 0*, i.e. without decomposition variables, is* $\{Var6, Var3\}$*. Then, FILO proceeds to compute next shortcuts:* $\{X_{var}, Var5, Var0, Var1, Var3, Var6^s, Var3^s\}$ *and* $\{Var1^r, Var5^r, Var0^r\}$ *which is also the initial shortcut. At this moment, the computation is terminated with* ***success*** *for the given constant. The partial solution is:* $\{Var1^r, Var5^r, Var0^r\} \mapsto A_2, \{X_{var}, Var5, Var0, Var1, Var3, Var6^s, Var3^s\} \mapsto \forall r.A_2, \{Var6, Var3\} \mapsto \forall sr.A_2$.

Assuming we obtain the partial solution for the constant A_1*:* $\{A, Var2^s, Var6^s\} \mapsto A_1, \{X_{var}, Var6, Var4^s, Var2, Var6^s, Var4\} \mapsto \forall s.A_1, \{Var4, Var6\} \mapsto \forall ss.A_1$*, we obtain the following* ***solution*** *for the input goal:* $X_{var} \mapsto \forall r.A_2 \sqcap A_1 \sqcap \forall s.A_1$.

5. Examples

FILO provides more than 20 examples, available in the dropdown menu in its interface. Among the examples, Example 16 is taken from [2]. FILO computes the same solution as presented in this paper, and the computation takes 3700 ms. During the computation, FILO worked with at most 29 variables, rejected 1290 goals in the pre-processing stage, and entered the shortcut computation stage 8 times. We utilized a machine with the following specifications: Intel(R) Core(TM) i7-1355U (1.70 GHz), 32.0 GB RAM, 64-bit operating system, running Windows 11 Pro (24H2).

Out of the examples provided, Example 8 is most difficult. It contains two constants and is not unifiable for either of them. Hence it terminates with failure after checking the generic goal produced for one constant only. Nevertheless it takes 7545 ms to terminate. The maximal number of variables is 33 and 1525 goals are rejected in the pre-processing stage, while the shortcuts were computed 12 times.

References

[1] S. Borgwardt, B. Morawska, Finding finite Herbrand models, in: N. Bjørner, A. Voronkov (Eds.), Proc. of the 18th Int. Conf. on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR-18), volume 7180 of *Lecture Notes in Computer Science*, Springer, 2012, pp. 138–152. URL: https://doi.org/10.1007/978-3-642-28717-6_13. doi:10.1007/978-3-642-28717-6_13.

[2] F. Baader, P. Narendran, Unification of concept terms in description logics, Journal of Symbolic Computation 31 (2001) 277–305. doi:10.1006/jsco.2000.0426.

[3] B. Morawska, D. Marzec, S. Kost, M. Henne, Filo – automated unification in $\mathcal{FL}_0$, https://arxiv.org/abs/2502.14130, 2025. URL: https://arxiv.org/abs/2502.14130. arXiv:2502.14130.

[4] F. Baader, S. Borgwardt, J. Mendez, B. Morawska, UEL: unification solver for EL, in: Y. Kazakov, D. Lembo, F. Wolter (Eds.), Proceedings of the 2012 International Workshop on Description Logics, DL-2012, Rome, Italy, June 7-10, 2012, volume 846 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2012. URL: https://ceur-ws.org/Vol-846/paper_8.pdf.

[5] F. Baader, P. Koopmann, F. Michel, A. Turhan, B. Zarrieß, Efficient tbox reasoning with value restrictions using $\mathcal{FL}_0$wer reasoner, Theory Pract. Log. Program. 22 (2022) 162–192. doi:10.1017/S1471068421000466.

[6] F. Baader, C. Lutz, B. Suntisrivaraporn, CEL - A polynomial-time reasoner for life science ontologies, in: U. Furbach, N. Shankar (Eds.), Automated Reasoning, Third International Joint Conference, IJCAR 2006, Seattle, WA, USA, August 17-20, 2006, Proceedings, volume 4130 of *Lecture Notes in Computer Science*, Springer, 2006, pp. 287–291. doi:10.1007/11814771_25.

[7] J. Mendez, jcel: A modular rule-based reasoner, in: I. Horrocks, M. Yatskevich, E. Jiménez-Ruiz (Eds.), Proceedings of the 1st International Workshop on OWL Reasoner Evaluation (ORE-2012), Manchester, UK, July 1st, 2012, volume 858 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2012. URL: https://ceur-ws.org/Vol-858/ore2012_paper12.pdf.