

A Typed Hybrid Description Logic Programming Language with Polymorphic Order-Sorted DL-Typed Unification for Semantic Web Type Systems

Adrian Paschke

Internet-based Information Systems, Dept. of Informatics, TU Munich, Germany
adrian.paschke@gmx.de

Abstract. In this paper we elaborate on a specific application in the context of hybrid description logic programs (hybrid DLPs), namely description logic Semantic Web type systems (DL-types) which are used for *term typing of LP rules based on a polymorphic, order-sorted, hybrid DL-typed unification as procedural semantics of hybrid DLPs*. Using Semantic Web ontologies as type systems facilitates interchange of domain-independent rules over domain boundaries via dynamically typing and mapping of explicitly defined type ontologies.

Key words: Homogenous and Heterogeneous Description Logic Programs, Polymorphic Order-Sorted Typed Unification, Rule Interchange

1 On the Need for hybrid DL-typed Unification

The works on combining rules and ontologies can be basically classified into two basic approaches: *homogeneous* and *heterogeneous* integrations. Both have pros and cons and the question whether the Semantic Web should adopt a homogeneous or heterogeneous view is still very much at the beginning. Starting from the early Krypton language [1] among the heterogeneous approaches, which hybridly use DL reasoning techniques and tools in combination with rule languages and rule/LP engines are e.g. CARIN [2], Life [3], AI-log [4], non-monotonic dl-programs [5], r-hybrid KBs [6], hybrid DL-typed LPs [15] and Datalog^{DL} [16]. Among the homogeneous approaches which combine the rule component and the DL component in one homogeneous framework sharing the combined language symbols are e.g. DLP [7], KAON2 [10] or SWRL [9]. Both integration approaches have pros and cons and different integration strategies such as reductions, SLD resolution extensions or fixpoint iterations are applied with different restrictions to ensure decidability such as DL-safe rules [10], where DL variables must also occur in a non DL-atom in the rule body. Furthermore, they can be distinguished according to their information flow which might be *uni-directional* or *bi-directional*. For instance, in homogeneous approaches bi-directional information flows between the rules and the ontology part a naturally supported and new DL constructs introduced in the rule heads can be directly used in ontology reasoning. However, in these approaches the DL reasoning is typically performed completely by the rule engine and benefits of existing tableau based algorithms in DL reasoners are lost. On the other side, heterogeneous approaches, benefit from the hybrid use of both reasoning concepts exploiting the advantages of both (using LP reasoning and tableaux based DL reasoning), but lack to some extent the freedom in combining both languages, e.g. bi-directional information flow and new DL constructs in rule heads are much more difficult to implement.

In this paper we focus on the hybrid approach and elaborate on a *polymorphic order-sorted DL-typed unification* as procedural semantics for *hybrid DL-typed logic*

programs, which allows term typing using external Semantic Web (RDFS / OWL) ontologies as order-sorted type systems, where the terminological concepts (classes) of the T-Box model are the defined types (a.k.a. sorts). On a pragmatic level typing in logic programming leads to rule languages with higher expressiveness which provide typical software engineering principles such as modularization and data abstraction. Types can be considered as an approximation of the intended interpretation which reflects the intention of the rule programmer, i.e. as an instrumentation of a logic program. They can be used to integrate domain specific vocabularies, which model the domain semantics or even pragmatic meaning, into domain-independent rules. As a result, such typed rules are much easier to interchange between domain-boundaries in distributed environment such as the (Semantic) Web. From a descriptive perspective they attach additional information, which can be used as "type guards" for selecting specific goals, i.e. they constrain the level of generality in queries and lead to much smaller search spaces and therefore improve the execution efficiency of query answering.

The theory of types in logic programming has been extensively studied and different approaches reaching from simple *descriptive type systems* to *prescriptive many-sorted, order-sorted* or *polymorphic typed (second-order) languages* have been proposed - see e.g. [11] for an overview. However, to the best of our knowledge there are nearly no approaches which build upon these results and implement practically running implementations in terms of (hybrid) DLP rule engines which use Semantic Web ontologies as type systems for DL-typed rules. The only works we are aware of which directly address DL-typing are AL-Log [4] and closely related Datalog-DL [16], which use the DL terminological models to define hybrid procedural type functions in constraint clauses and OO-RuleML [17] resp. OO-jDrew [18], which use RDFS to define external order-sorted type systems. These systems adopt a *descriptive typing approach* where types are included as extra DL functions which make hybrid calls on the external type systems (ontologies) and reasoner. These additional functions act as *constraints*, which restrict the values of variables and constants in the *constraint clauses* to range over the instances of the externally specified DL concepts. Although this constraint solution to some extent ensures expressiveness in the sense that it allows to reuse the binary relations (properties) defined in the external ontology, it has some serious drawbacks in the context of term typing, since this approach leaves the usual procedural semantics of resolution and unification unchanged. Hence, the constraints, i.e. the additional DL functions, apply only in the body part of a rule according to the selection function of the LP inference algorithm which typically selects the "left-most" unified literal as next subgoal in the goal refutation attempt. As a result, according to the depth-first strategy the type constraints are used relatively late in the derivation process and they are in particular do not apply during the unification between (sub)goals and rule heads, which leads to large and needless search spaces (failed search trees) and might lead to unintended results since there is no way to directly verify and exclude the unification of typed free (sub)goals and typed rule heads which do not match according to their type definitions. Furthermore, fresh typed constant terms, i.e. new DL instances a.k.a. individuals of a particular DL type (class), can not be introduced directly in rule heads since this would require special semantics and formalizations, e.g., by conjunctive rule heads consisting of a literal which defines the individual and another literal which defines its type. For instance, a rule with a variable X of type C in the rule head, would need an extra constraint, e.g. $type(X, C)$ in the body, e.g.: $p(X) \leftarrow q(X), type(X, C)$. Obviously, such a constraint rule is dependent on the order of the body atoms: if $type(X, C)$ is before $q(X)$ the variable X might not be bound to a ground individual and hence can not be verified; on the other hand, if it is after $q(X)$ it does not directly

constrain the subgoal $q(X)$, which might be possibly very deep including many variable bindings which are not of type C .

In contrast to these works we choose a *prescriptive typing perspective*, where types are direct properties of the logical formulas and apply a heterogenous integration approach, where dynamic type checking during *typed unification* as a procedural semantics is outsourced to an Semantic Web API with an external DL reasoner (the Jena API (<http://jena.sourceforge.net>) in combination with the DL reasoner Pellet (<http://www.mindswap.org/2003/pellet/>). That is with our *hybrid DL-typed logic* we present a different and in the context of the Semantic Web new approach which also formes the basis for typical Software Engineering principles such as modularization or data abstraction for DL-typed rule bases. It allows many programming errors to be detected via static (checking types at compile time during parsing the LP scripts) and dynamic type checking which might be otherwise difficult to locate. It facilitates ad-hoc polymorphism for typed variables, which enables overloading of rule heads leading to different rule variants. Moreover, rules can be kept domain independent introducing domain dependent vocabularies at runtime by means of dynamically typing of untyped variables with the type of the unified term, e.g. the type of DL individual which is bound to the untyped variable.

2 Polymorphic Order-Sorted DL-Typed Unification

We define a "has-type" relation of the form $t : r$ denoting that term t has type r . Note, that this may not eliminate programs that have a denotation in the untyped framework, because constants and variables are allowed to be untyped denoted by $t : \top$ or simply t . An untyped term t is implicitly assumed to be of type *Resource* which is the common super class from which all other DL classes inherit and hence unifies with all other typed terms. In particular, we define the type of complex functions and lists to be untyped by default. Hence, they only unify with untyped variables for which the type test can be omitted. The external type alphabet T is a finite set of monomorphic type symbols t built over the distinct set of terminological concepts C in the DL language L , i.e. the set of classes in the T-Box model of the external DL ontology. We assume that the type alphabet is fixed, but arbitrary, i.e. no new terminological concepts can be introduced in the T-Box. This ensure that we can also apply static type checking on the used types at compile time during parsing the DL-typed LP script. The smallest set of constants is built over the set of individual names in L , but we do not fix the constant names and allow arbitrary (under UNA) fresh constants (individuals) to be introduced within facts and rule heads. However, only uni-directional information flows from the DL part to the rules are supported, which is desirable, because new individuals introduce in rule heads should not apply globally as the individuals defined in the DL A-box model do, but should be used locally in a particular refutation attempt to prove a particular typed goal.

Definition 1: A *hybrid DL-typed rule* $r: H \leftarrow B_1 \wedge \dots \wedge B_n$ where H and B_i are literals (positive or negative atoms with default and explicit negation) with n-ary sequences of typed $t : r$ or untyped t constants, variables or finite complex terms/functions.

Definition 2: A *hybrid DL-typed fact* f is a either a ground rule with an empty body $H \leftarrow$, where the head H is a literal with a n-ary sequence of typed or untyped constants or ground complex terms or it is a non-ground rule with an empty body where the head literal contains variables which are interpreted as queries on the DL knowledge base.

Definition 3: A *hybrid DL-type KB* K is a tuple $\langle K^{LP}, K^{DL} \rangle$ where K^{DL} is a DL knowledge

base (T-Box and optional A-Box) defined in L and K^{LP} is an LP Π with hybrid DL-typed rules and facts.

Variables in non-ground facts are interpreted as queries on the DL knowledge base K^{DL} . For instance, a variable of type C which is bound by unification with a subgoal to the individual a is interpreted as an instantiation query on K^{DL} for the individual a to be of class C . A free variable X of type C in a fact is interpreted as a universally quantified query deriving all *known* individuals in the A-Box which are an instance of C failing if no individual of the respective type can be found. We exclude implied unknown individuals in K^{DL} and ensure that each variable is bound to individuals which are explicitly defined either in the A-Box or directly in KB^{LP} . We first define the rules for untyped unification in terms of equation-solving transformations for elimination (E), decomposition (D), variable binding (B) and orientation (O).

(E) $\frac{E \& Y \doteq Y}{E}$, where Y is a variable

(D) $\frac{E \& f(t_1, \dots, t_n) \doteq f(t'_1, \dots, t'_n)}{E \& t_1 \doteq t'_1 \& \dots \& t_n \doteq t'_n}$

(B) $\frac{E \& Y \doteq t}{\sigma(E) \& Y \doteq t}$, where Y is a variable, t is a constant or variable term, and Y occurs in E but not in t , and where $\sigma = \{Y/t\}$

(O) $\frac{E \& t \doteq Y}{E \& Y \doteq t}$, where Y is a variable and t is not a variable

The equation $E = \{t_1 \doteq t'_1, \dots, t_n \doteq t'_n\}$ describes the substitutions of two terms $\{t_i/t'_i\}$ and transform E using the four rules into a set of equations $E' = \{Y_i | i \in \{1, \dots, n\}\}$ where Y_i are distinct variables which do not occur elsewhere in E' , i.e. $\sigma = \{Y_1/t_1, \dots, Y_n/t_n\}$ is the mgu of the unification problem given by the original set of equations E . Unification fails, if there is an equation $f(t_1, \dots, t_n) \doteq g(t'_1, \dots, t'_n)$ in E with $f \neq g$ or if there is an equation $Y \doteq t$ in E such that $Y \doteq t$ and Y occurs in t . We now extend this basic set of unification rules to a hybrid polymorphic order-sorted DL-typed unification. We restrict type checking to finding the lower bound of two types (r_1, r_2) under the partial order \leq of the DL taxonomy model with an upper bound \top , i.e. $\text{Resource} \equiv \text{untyped}$, and a lower bound $\perp \equiv \text{empty}$ and replace the type of a term with the more specific type concept. We define a *lower* operation by:

$\text{lower}(r_1, r_2) = (r_2/r_1) \rightarrow r_1$, if $r_1 \leq r_2$ resp. $\text{lower}(r_1, r_2) = (r_1/r_2) \rightarrow r_2$, if $r_1 > r_2$

$\text{lower}(r_1, \top) = (\top/r_1) \rightarrow r_1$ resp. $\text{lower}(\top, r_2) = (\top/r_2) \rightarrow r_2$, where $\top = \text{untyped}$

$\text{lower}(r_1, r_2) = \perp$, otherwise, where $\perp = \text{empty type}$.

Note that, the operation *lower* requires at most two queries to the external DL reasoner to compute the lower bound of two types having a lower bound at all. If the type system consists of more than one DL ontology, the ontologies might be merged into a combined ontology with the common super class *Resource* and possible cross links between the component ontologies defined by e.g. *owl : equivalentClass* or *owl : disjointWith*. Note that, this may introduce conflicts between terminological definitions, which are out of the scope of this paper. To enable polymorphic typing of variables during unification, i.e. a variable may change its type dynamically, we introduce a set $P = \{t_1 : r_1, \dots, t_n : r_n\}$ of type restrictions, denoting that the term t_i (currently) has type r_i , as a prefix to the set of equations E : $P \& E$. The modified and extended type rules are:

(E) $\frac{P \& E \& Y \doteq Y}{P \& E}$, where Y is a variable

(D) $\frac{P \& E \& f(t_1, \dots, t_n) \doteq f(t'_1, \dots, t'_n)}{P \& E \& t_1 \doteq t'_1 \& \dots \& t_n \doteq t'_n}$

(B') $\frac{P \& Y : r \& E \& Y \doteq t}{P' \& \sigma(E) \& Y \doteq t}$, where Y is a variable, t is a variable or non-variable term, and Y occurs in E but not in t , and where $\sigma = \{Y/t\}$. $P \& t : r$ reduces to P' using the auxiliary type rules ET and BT

(O) $\frac{P \& E \& t \doteq Y}{P \& E \& Y \doteq t}$, where Y is a variable and t is not a variable

The auxiliary rules for polymorphic unification of types are:

$$(ET) \frac{P \& f(t_1, \dots, t_n) : r}{P \& f(t_1, \dots, t_n) : \top}, \text{ if } f : r_1 \dots r_n \rightarrow r_2 \text{ and } r_2 \leq r \text{ and } (ET') \frac{P \& f(t_1, \dots, t_n) : \top}{P \& Y : r_1 \& Z : r_2}$$

$$(BT) \frac{P \& Y : r_1 \& Z : r_2}{P \& Y : lower(r_1, r_2)}$$

DL-typed unification fails (1) if there is an equation $f(t_1, \dots, t_n) \doteq g(t'_1, \dots, t'_m)$ in E with $f \neq g$ or (2) if there is an equation $Y \doteq t$ in E such that $Y \doteq t$ and $Y \in t$ or (3) if there is an equation $Y \doteq t$ in E such that $Y : r_1$ and $t : r_2$, where t is a constant term and $r_2 > r_1$ or (4) if there is an equation $Y \doteq Z$ in E such that $Y : r_1$ and $Z : r_2$ and $lower(r_1, r_2) = \perp$, where Y and Z are variable terms. Otherwise, if $E' = \{Y_i | i \in \{1, \dots, n\}\}$ then $\sigma = \{Y_1/t_1, \dots, Y_n/t_n\}$ is the mgu of the unification problem given by the original set of equations E .

In contrast to the unsorted unification, (B') now involves polymorphic unification of order-sorted types with a subtype resp. equivalence test $r_2 \leq r_1$ and a computation of the lower bound of two types $lower(r_1, r_2)$ in the auxiliary rules, possibly assigning the more specific type (i.e. the lower type) to a variable. According to the hybrid approach the DL inference task incl. possible ontology mapping (equivalent classes) is solved by an external DL reasoner which is queried within the unification algorithm. The polymorphic variables may change their type during unification according to the rule (BT) and the *lower* operation. (ET') is introduced to reduce unification to special cases of the binding rule (B) in the untyped case without type checking, i.e. to efficiently process untyped variables. In the order-sorted case where all terms are of type "Resource" resp. untyped, unification is performed as in the ordinary untyped case. Informally the polymorphic order-sorted unification rules state:

Untyped Unification: Ordinary untyped unification without type checking

Untyped-Typed Unification: The untyped query variable assumes the type of the typed target

Variable-Variable Unification: (1) If the query variable is of the same type as the target variable or belongs to a subtype of the target variable, the query variable retains its type (according to *lower*), i.e. the target variable is replaced by the query variable. (2) If the query variable belongs to a super-type of the target variable, the query variable assumes the type of the target variable (according to *lower*), i.e. the query variable is replaced by the target variable. (3) If the query and the target variable are not assignable ($lower = \perp$) the unification fails

Variable-Constant Term Unification: (1) If a variable is unified with a constant of its super-type, the unification fails. (2) If the type of the constant is the same or a sub-type of the variable, it succeeds and the variable becomes instantiated.

Constant-Constant Term Unification: Both constants are equal and the type of the query constant is equal to the type of the target constant.

Complex terms such as lists are untyped by default and hence are only allowed to be unified with untyped variables resp. variables of type "Resource".

3 Implementation and Discussion

The main motivation for introducing Semantic Web based types into declarative logic programs comes from Software Engineering, where principles such as data abstraction, modularization or consistency checks are vital for the development and maintenance of large rule-based systems and from distributed system engineering and collaboration, where domain-independent rules need to be interchanged and given a domain-dependent meaning in their target environments. From a computational view, the use of order-sorted types can drastically reduce the search space, hence increasing the runtime efficiency and the expressive power, e.g. enabling overloading. In this paper we have contributed with a hybrid polymorphic order-sorted DL-typed unification algorithm as a procedural semantics for hybrid DL-typed LPs where external Semantic

Web ontologies which are serialized in RDFS or OWL are used as type systems. In contrast to other DL typing/integration approaches we follow a prescriptive typing approach and incorporate terminological type information directly into the names of symbols in the rule language. The approach permits ad-hoc polymorphism for typed and un-typed terms, where variables might change their types, akin to coercion in object-oriented type systems and allow overloading. We have implemented the hybrid DL-typed unification within the OWL2Prova API of the ContractLog KR based on Prova, an open-source Java-based backward-reasoning rule engine with a Prolog like scripting syntax. The hybrid unification algorithm calls an external description logic reasoner the Jena API (<http://jena.sourceforge.net>) in combination with Pellet (<http://www.mindswap.org/2003/pellet/>) to answer DL queries against the ontology models at runtime during typed unification. The major advantage of this hybrid approach is that highly optimized tableaux algorithms are used for reasoning during dynamic type checking. The ContractLog KR with the OWL2Prova API based on the Prova rule engine provides a Prolog related scripting syntax to write DL-typed LPs, e.g.:

```
import("./dl.typing/WineProjectOWL.owl"). % import external type system
reasoner("dl"). % select OWL-DL reasoner
serve(X:vin.Wine):- wine(X:vin.Wine). % DL-typed rule, vin is namespace prefix
wine(vin.White.Wine:Chardonnay). wine(X:vin.Red.Wine). % DL-typed facts
```

The hybrid DL-typed unification algorithm is generally applicable to different logic classes such as normal LPs with finite functions or Datalog LPs which assure decidability. Although full LPs with functions are in general *r.e.-complete* much better complexity results can be achieved. In our reference implementation we compute well-founded semantics for normal LPs which has been proven to be \prod_1^1 -complete (see e.g. [12]). The general untyped unification problem in logic programming is *P-complete under logspace reductions* and *P-hardness* was proven e.g. by [13]. We, have optimized the basic untyped unification algorithm to almost linear time [19]. We allow the DL languages which are used to describe the order-sorted type hierarchies to range over decidable DLs reaching from *ALC* which corresponds to the fragment of FOL obtained by restricting the syntax to formulas containing only two variables (closely related to the multi-modal language K [14]) up to *SHIQ* (i.e. *ALC_{R+}* extended with property hierarchies (H), inverse roles (I) and qualified number restriction (Q)), which is in *EXPTIME* and *SHOIN(D)* with nominals and unqualified number restrictions, which has *NEXPTIME* complexity. As a result the hybrid DL-typed unification, i.e. the problem of solving equations over terms, of our DL-typed unification algorithm is in *EXPTIME* for OWL Lite type systems resp. *NEXPTIME* for OWL-DL type systems. Our favorite is SHIQ(D) (OWL Lite) which is a good compromise between complexity of reasoning and expressiveness. Although this are rather high worst-case complexity bounds, the used external DL reasoner "Pellet" implements a highly optimized tableaux algorithm which behaves adequately in practice. In fact, this is another strong argument for our hybrid approach which uses an external tableaux reasoner for type checking and not the LP rule engine, as in homogeneous DLP approaches, which is typically not optimized for DL reasoning. It should be noted that our DL-typed LP language also supports RDFS vocabularies to describe order-sorted type systems, which due to the reduced expressiveness, e.g. lack of equivalence entailment, are computationally much more efficient. From a semantics point-of-view of a polymorphic order-sorted type system, due to the fact that RDFS is not just a smaller subset of the OWL Lite or OWL DL languages, another advantage of RDFS is the usage of its recursive meta-model, in particular the non-limited type relations.

References

1. R.J. Brachman, P.V. Gilbert, H.J. Levesque. An essential hybrid reasoning system: Knowledge and symbol level accounts for krypton. in Int. Conf. on Artificial Intelligence. 1985.
2. A. Y. Levy and M.-C. Rousset. CARIN: A Representation Language Combining Horn Rules and Description Logics (1996) European Conference on Artificial Intelligence
3. H. Ait-Kaci and A. Podelski. Towards the meaning of LIFE. in Int. Symposium on Programming Language Implementation and Logic Programming. 1991: Springer.
4. F. M. Donini, M. Lenzerini, D. Nardi, A. Schaerf. AL-log: integrating Datalog and description logics, *J. of Intelligent and Cooperative Information Systems*, 10/1998, pages 227-252.
5. T. Eiter, T. Lukasiewicz, R. Schindlauer, H. Tompits. Combining Answer Set Programming with Description Logics for the Semantic Web. KR 2004, pages 141-151.
6. R. Riccardo. On the decidability and complexity of integrating ontologies and rules. *Journal of Web Semantics*, 3(1), 2005.
7. B. Groszof, I. Horrocks, R. Volz, S. Decker. Description Logic Programs: Combining Logic Programs with Description Logics. In Proc. of WWW 2003, Budapest, Hungary, May 2003, pp. 48-57. ACM, 2003.
8. B. Motik, U. Sattler, R. Studer. Query Answering for OWL-DL with Rules. *Journal of Web Semantics*, 3(1), 2005.
9. I. Horrocks, P. F. Patel-Schneider, H. Boley, S. Tabet, B. Groszof, M. Dean. SWRL: A Semantic Web Rule Language Combining OWL and RuleML. W3C Member Submission. 21-May-2004. URL: <http://www.w3.org/Submission/SWRL/>.
10. B. Motik, U. Sattler, R. Studer. Query Answering for OWL-DL with Rules. *Journal of Web Semantics*, 3(1), 2005.
11. F. Pfenning, editor. *Types in Logic Programming*. MIT Press, 1992.
12. J. Schlipf. The Expressive Powers of Logic Programming Semantics. *J. Computer and System Sciences*, 1995. 51(1): p. 64-86.
13. C. Dwork, P.C., Kanellakis, J.C. Mitchell. On the sequential nature of unification. *J. Log. Program.* 1, 1 (Sep. 1984), pages 35-50.
14. J. Y. Halpern and Y. Moses. A guide to completeness and complexity for modal logics of knowledge and belief. *Artificial Intelligence*, 54:319-379, 1992.
15. A. Paschke: Typed Hybrid Description Logic Programs with Order-Sorted Semantic Web Type Systems based on OWL and RDFS, IBIS, TUM, Technical Report 12/05, 2005.
16. J. Mei, H. Boley, J. Li, V.C. Bhavsar, Z. Lin. Datalog-DL: Datalog Rules Parameterized by Description Logics, Proc. of CSWWS'06, Semantic Web and Beyond, pp. 171-188, 2006.
17. H. Boley. Object-Oriented RuleML: User-Level Roles, URI-Grounded Clauses, and Order-Sorted Terms, RuleML 03, pp. 1-16.
18. M. Ball, H. Boley, D. Hirtle, J. Mei, B. Spencer. The OO jDrew Reference Implementation of RuleML, Proc. RuleML'05, pp. 218-223.
19. A., The ContractLog Inference Engine: A configurable inference service for logic programming with linear resolution, goal memoization, loop prevention, and hybrid Semantic Web typed unification supporting selectable SLDNF related and well-founded semantics. 05/05, IBIS, TUM, Technical Report: Munich.