

clingowl : a Tool for Ontology Querying in Answer Set Programming

Pedro Cabalar³, Stefania Costantini² and Alina Vozna^{1,2,*}

³University of A Coruña, Spain

¹University of Pisa, Largo B. Pontecorvo, Pisa, 56127, Italy

²University of L'Aquila, Department of Information Engineering, Computer Science and Mathematics, Italy

Abstract

We describe `clingowl`, an extension of the Answer Set Programming system `clingo` for ontology-aware reasoning over OWL ontologies. `clingowl` extends ASP with dedicated OWL theory atoms that enable ontology queries and entailment checks directly within ASP programs. The system combines `clingo`'s theory language, Abstract Syntax Trees (ASTs), the python library OWLAPY, and external Description Logic reasoners to transparently evaluate ontology expressions during solving. We present the input language, system architecture, and implementation of `clingowl`, and illustrate its use through a family ontology and a clinical example inspired by SNOMED CT. We also present a preliminary experimental comparison against the tool DLVHEX (also combining ASP with ontologies), showing how different integration architectures affect the runtime behaviour on ontology-aware ASP programs. The results indicate that `clingowl` has a lower translation overhead and scales favourably on realistic OWL ontologies such as SNOMED CT while preserving a declarative OWL-oriented interface.

Keywords

Answer Set Programming, Description Logic, OWL, Theory Atoms, Ontology Reasoning

1. Introduction and Motivation

The increasing adoption of Artificial Intelligence (AI) in healthcare has led to the development of intelligent systems supporting clinical decision-making, personalized treatments, and patient monitoring [1, 2, 3]. While machine learning techniques have demonstrated remarkable predictive capabilities, they often operate as black-box models, making it difficult for clinicians to understand, verify, and justify the reasoning behind their recommendations [4, 5]. In safety-critical domains such as medicine, transparency and explainability are not optional requirements but essential characteristics of trustworthy decision-support systems.

Symbolic Artificial Intelligence provides an alternative paradigm based on explicit knowledge representation and logical inference. In particular, Description Logics (DLs) have become the standard formalism for representing structured domain knowledge through ontologies [6]. Their practical adoption has been largely driven by the Web Ontology Language (OWL) ¹, which constitutes the foundation of numerous biomedical ontologies, including SNOMED CT ², the largest standardized clinical terminology currently available. DL reasoners can infer implicit knowledge from ontological axioms, supporting tasks such as concept classification, subsumption checking, and instance retrieval, making them particularly suitable for representing medical knowledge.

Complementary to DLs, Answer Set Programming (ASP) is a declarative logic programming paradigm particularly suitable for non-monotonic reasoning, constraint solving, and optimization. ASP has been successfully applied to planning, scheduling, recommendation systems, and healthcare decision support [7]. However, combining DLs and ASP is not straightforward: DL reasoners are based on classical,

Datalog 2.0 2026: 6th International Workshop on the Resurgence of Datalog in Academia and Industry, September 7, 2026, Klagenfurt, Austria

*Corresponding author.

✉ cabalar@udc.es (P. Cabalar); stefania.costantini@univaq.it (S. Costantini); alina.vozna@phd.unipi.it (A. Vozna)

ORCID 0000-0001-7440-0953 (P. Cabalar); 0000-0002-5686-6124 (S. Costantini); 0009-0009-0179-6948 (A. Vozna)

 © 2026 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

¹<https://www.w3.org/TR/owl2-overview/>

²<https://www.snomed.org/>

monotonic inference and open world assumption, whereas ASP systems are non-monotonic and work under the closed world assumption. As a result, there are few tools that allow this combination, and exploiting ontology-derived knowledge during ASP computation is unusual in a practical setting.

This limitation becomes particularly significant in biomedical domains, where large ontologies encode rich semantic relationships among diseases, procedures, anatomical structures, pharmaceuticals, and clinical findings. Existing approaches [8, 9, 10, 11] have investigated different forms of interaction between ontologies and logic programs. Nevertheless, seamlessly exploiting ontology reasoning inside modern ASP systems while preserving modularity and declarative semantics is still an open research problem.

To address these limitations, this paper presents `clingowl`, a lightweight extension of `clingo` with OWL theory atoms, enabling ASP programs to query external DL reasoners during solving. The framework is based on a translation pipeline involving Abstract Syntax Trees (ASTs)³, OWLAPY [12], and standard Description Logic reasoners, providing a flexible interface between symbolic knowledge representation and constraint-based reasoning.

Although the proposed framework is ontology-independent, this work is motivated by applications in digital health, where rich semantic resources such as SNOMED CT can significantly improve the quality and explainability of clinical reasoning. The long-term objective is to support ontology-aware decision-making systems capable of combining standardized medical knowledge with declarative rules and constraints within a unified logical framework. Unlike previous ontology-aware ASP systems, `clingowl` is also evaluated experimentally against DLVHEX using a reproducible benchmark executed on identical hardware. This comparison highlights the different execution models adopted by the two systems and provides a first quantitative assessment of the overhead introduced by ontology-aware reasoning inside ASP.

2. Background

Description Logics are a family of knowledge representation formalisms that provide a formal foundation for representing structured domain knowledge and supporting automated reasoning [6, 13]. They constitute the logical basis of the Web Ontology Language (OWL 2), the W3C standard for ontology representation.

A DL knowledge base consists of two complementary components. The TBox (Terminological Box) contains the conceptual knowledge of the application domain, such as class hierarchies and logical definitions, whereas the ABox (Assertional Box) contains assertions about individuals and their relationships. Together, these components define the ontology over which reasoning is performed.

OWL 2 is based on the expressive Description Logic $\mathcal{SROIQ}(D)$ [14], which supports a rich set of constructors for defining complex concepts and relationships. The proposed framework exploits a representative subset of these constructors, namely concept conjunction ($\sqcap$), concept disjunction ($\sqcup$), negation ($\neg$), existential restrictions ($\exists R.C$), universal restrictions ($\forall R.C$), inverse roles (R^-), concept subsumption ($\sqsubseteq$), concept equivalence ($\equiv$), and class and role assertions. These constructors are sufficient to express the ontology queries currently supported by the framework and constitute the basis of the OWL theory atoms introduced in Section 5.

Due to these characteristics, Description Logics have become the reference formalism for biomedical ontologies, where complex clinical knowledge must be represented in a machine-interpretable way. Several highly optimized DL reasoners, including Pellet, HermiT, FaCT++, and ELK⁴, provide efficient reasoning services over OWL ontologies and are widely adopted in both research and industrial applications.

Reasoning over OWL ontologies is performed by dedicated Description Logic reasoners, which provide services such as concept classification, subsumption checking, consistency verification, and instance retrieval. The proposed framework delegates these inference tasks to external reasoners while

³<https://potassco.org/clingo/python-api/5.8/clingo/ast.html>

⁴A list of reasoners is available at https://www.w3.org/2001/sw/wiki/Category:OWL_Reasoner

allowing ASP programs to access the inferred knowledge through dedicated theory atoms. Further details on the logical foundations of the proposed integration can be found in our recent work *Here-and-There with Constraints for Description Logic* [15].

3. Using `clingowl`

This section illustrates how `clingowl` can be used to combine ASP with OWL reasoning. We first introduce a simple family ontology that serves as a running example, then describe the input language provided by `clingowl`, explain how a program is executed, and finally present the corresponding output. A more realistic medical example based on SNOMED CT is presented in the next section. The implementation of `clingowl`, together with the example ontologies and ASP encodings used in this paper, is publicly available.⁵

3.1. Family Ontology Example

To illustrate the proposed language, we consider a small OWL ontology describing family relationships. The ontology contains classes such as `Person`, `Adult`, `Father`, and `Mother`, together with object properties including `hasChild`. It also includes several individuals, such as `Peter`, `Susan`, `Mary`, and `John`.

Although intentionally simple, this ontology provides a convenient example for demonstrating how ASP rules can interact with OWL reasoning through the proposed theory atoms.

For example, the following rule retrieves all ontology individuals that are classified as adults and either fathers or mothers:

```
1 parent(X) :-
2   &owlquery{ adult & (father | mother) } = X.
```

Existential restrictions can be queried in the same way:

```
1 has_male_children(X) :-
2   &owlquery{ hasChild ! male } = X.
```

which returns every individual having at least one child belonging to the class `Male`. Boolean ontology entailments can also be evaluated. For instance,

```
1 check_father :-
2   &owl{ father <: person }.
```

succeeds whenever the ontology entails that `Father` is a subclass of `Person`.

This running example will be used throughout the remainder of the paper. A more realistic case study based on SNOMED CT is presented in Section 4.1.

3.2. Input Language

`clingowl` extends the standard `clingo` input language by introducing two dedicated OWL theory atoms through Clingo’s theory language mechanism. Consequently, a `clingowl` program is an ordinary ASP program enriched with ontology queries and ontology entailment checks, while all standard ASP constructs, including rules, constraints, and optimization statements, remain unchanged.

The first theory atom,

`&owl{A},`

evaluates whether an OWL axiom or assertion A is logically entailed by the ontology and returns a Boolean result. The second theory atom,

⁵<https://github.com/AlinaVozna/clingowl>

`&owlquery{E}=X`,

retrieves all ontology individuals satisfying the Description Logic class expression E , binding each matching individual to the ASP variable X .

Therefore, a `clingowl` rule has the general form

$$H :- B_1, \dots, B_n, Q_1, \dots, Q_m$$

where H is an ordinary ASP atom, the B_i are ordinary ASP literals, and each Q_j is an OWL theory atom representing either an ontology entailment check or an ontology query.

The complete definition of the theory language, together with the supported Description Logic constructors and ontology axioms, is presented in Section 5.1. Since theory atoms integrate into the grounding and solving process as literals in rule bodies, ontology reasoning can be seamlessly combined with standard ASP rules, constraints, and optimization statements.

3.3. Running `clingowl`

A `clingowl` application consists of three components:

- an OWL ontology;
- an ASP program containing OWL theory atoms;
- the `clingowl` Python engine.

The ontology is loaded by the Python engine, which initializes the underlying Description Logic reasoner before the ASP program is evaluated.

During execution, `clingowl` loads the specified ontology, parses the ASP program, evaluates all OWL theory atoms through the Description Logic reasoner, and finally computes the stable models using the standard `clingo` solving pipeline.

From the user's perspective, ontology interaction is completely transparent. The developer only writes ASP rules enriched with the `&owl` and `&owlquery` theory atoms, while ontology loading, query evaluation, and communication with the reasoner are automatically managed by `clingowl`.

3.4. Example Output

Executing the previous program over the family ontology produces an answer set similar to the following:

```
1 parent(susan)
2 parent(diego)
3 parent(peter)
4 has_male_children(diego)
5 check_father
```

The resulting stable model contains both atoms inferred by the ASP solver and atoms obtained through ontology reasoning. In particular, predicates such as `parent/1` and `male_children/1` originate from OWL queries, whereas `check_father` corresponds to ontology entailment checks.

This example illustrates how `clingowl` allows ontology-derived knowledge to participate naturally in ASP computation while preserving the declarative nature of both OWL and ASP.

4. The target Ontology: SNOMED CT

Among biomedical ontologies, SNOMED CT is currently the most comprehensive standardized clinical terminology, providing a structured representation of diseases, procedures, body structures, pharmaceutical products, organisms, clinical findings, and many other healthcare concepts [16]. Its Description

Logic foundation enables semantic interoperability between heterogeneous healthcare systems while supporting automated reasoning over clinical concepts.

The ontology contains hundreds of thousands of concepts connected through hierarchical and semantic relationships, making it suitable for advanced inference tasks but also challenging from a computational perspective. Efficient reasoning over SNOMED CT typically requires specialized Description Logic reasoners capable of processing its large and expressive ontology [17].

Several approaches have been proposed to facilitate the practical use of SNOMED CT. Expression Constraint Language (ECL) allows users to define subsets of concepts through declarative expressions, although efficient execution generally depends on dedicated terminology services or graph-based infrastructures [18]. Furthermore, studies have identified quality issues within the ontology, including inconsistencies between lexical descriptions and logical definitions, as well as varying structural quality across different hierarchies, highlighting the importance of robust reasoning mechanisms [19, 20].

Although SNOMED CT has been successfully adopted in several national healthcare systems, practical deployments often require subset extraction, terminology mapping, and manual validation before the ontology can be effectively exploited in clinical workflows. These limitations motivate the development of reasoning frameworks capable of directly interacting with SNOMED CT while preserving its semantic structure.

In this work, SNOMED CT is considered the target ontology for evaluating the proposed integration framework. Rather than translating the ontology into ASP facts, the proposed approach delegates semantic reasoning to external Description Logic reasoners and enables ASP programs to query ontological knowledge through dedicated theory atoms.

4.1. The SNOMED Example

To illustrate the practical use of the proposed framework on a realistic biomedical ontology, we consider a set of ontology-driven reasoning tasks over the full SNOMED CT ontology. In this setting, the ASP program does not encode the medical knowledge itself; instead, it invokes OWL theory atoms to query subclass relations and complex concept definitions directly from SNOMED CT. SNOMED CT labels are normalized before being used in ASP: since ASP identifiers cannot directly contain spaces or parentheses, labels are converted into lowercase terms with underscores replacing non-alphanumeric characters. For example, “Allergy to penicillin (finding)” becomes `allergy_to_penicillin`. The implementation also supports direct references through `id(NUMBER)`, where `NUMBER` is the SCTID of the target concept or property.

The following rule checks a basic taxonomic relation:

```
1 check_basic_parent :-  
2     &owl{allergy_to_penicillin <: propensity_to_adverse_reaction}.
```

This query asks whether the concept `allergy_to_penicillin` is entailed to be a subclass of `propensity_to_adverse_reaction`. Since this relation is present in SNOMED CT, the predicate `check_basic_parent` is derived.

A more expressive example verifies a richer Description Logic definition involving conjunction and existential restrictions:

```
1 check_penicillin_definition :-  
2     &owl{allergy_to_penicillin <:  
3         (propensity_to_adverse_reaction &  
4         (role_group !  
5             ((causative_agent ! substance_with_penicillin_structure) &  
6             (has_realization ! allergic_process))))}.  
6
```

This rule checks whether SNOMED CT entails that `allergy_to_penicillin` is subsumed by a concept describing a propensity to adverse reaction together with a role group stating that the causative agent is some `substance_with_penicillin_structure` and that the realization is an

allergic_process. The example shows how the framework can handle non-trivial OWL class expressions directly inside ASP rules.

Finally, the following rule illustrates negative reasoning at the ASP level:

```
1 not_a_product :-  
2   not &owl{allergy_to_penicillin <: product_containing_amoxicillin}.
```

Here, the ontology is queried to test whether an allergy concept is subsumed by a medicinal product concept. Since such a subsumption is not entailed, the theory atom fails and the ASP rule derives not_a_product. This demonstrates how monotonic OWL entailment can be combined with ASP negation-as-failure.

When executed on the full SNOMED CT ontology, the framework derives the following atoms:

```
1 check_basic_parent  
2 check_penicillin_definition  
3 not_a_product
```

These results confirm that the framework is able to evaluate both simple and structurally rich ontology queries over a real-world biomedical ontology. In particular, the medical knowledge is not re-encoded as ASP facts; instead, it is accessed directly from SNOMED CT through theory atoms and translated into OWL-based reasoning calls. This illustrates the complementary roles of the two paradigms: Description Logic reasoning provides access to the formal ontology, while ASP offers a flexible rule-based layer in which ontology-derived results can be combined with symbolic and non-monotonic reasoning.

5. Implementation

This section describes the implementation of `clingowl`, focusing on the internal architecture that enables the interaction between ASP and OWL reasoning. Unlike the previous section, which introduced the language from the user's perspective, here we discuss how theory atoms are parsed, translated, and evaluated through an external Description Logic reasoner.

The implementation has been developed entirely in Python and builds upon four main software components: `clingo`, `clingoX`, `OWLAPY`, and an OWL reasoner (currently Pellet). Rather than extending the ASP solver itself, `clingowl` exploits Clingo's theory interface to introduce ontology-aware atoms while preserving the standard grounding and solving pipeline. Ontological reasoning is delegated to the external reasoner, whereas ASP remains responsible for rule evaluation and stable-model computation.

Figure 1 illustrates the overall architecture of `clingowl`. The system acts as an intermediate layer between the ASP solver and an OWL reasoner, providing a transparent interface for querying ontological knowledge during program evaluation.

The execution pipeline consists of five main components.

1. **ASP Program.** The user writes a standard ASP program enriched with the theory atoms introduced by `clingowl`. These atoms allow ontology queries and entailment checks to be embedded directly inside ASP rules without requiring explicit API calls or ontology preprocessing.
2. **Theory Parser and Translator.** The ASP program is parsed using Clingo's AST. During this phase, every OWL theory atom is recognized through a custom `#theory` definition and translated into an internal representation suitable for evaluation. The translation replaces the high-level Description Logic syntax with executable callback functions while preserving the logical structure of the original expression.
3. **Ontology Interface.** The translated expressions are evaluated through an interface implemented using `OWLAPY`. This component converts the internal symbolic representation into OWL objects, including classes, object properties, individuals, and complex class expressions. It also manages the communication with the external Description Logic reasoner.
4. **OWL Reasoner.** Semantic reasoning is delegated to a standard OWL reasoner. In the current implementation, Pellet is employed to perform inference services such as concept satisfiability,



Figure 1: Architecture of `clingowl`.

subsumption checking, class equivalence, instance retrieval, and object property reasoning. Since `clingowl` communicates with the reasoner exclusively through OWLAPY, alternative reasoners supporting the same interface can be adopted without modifying the ASP layer.

5. **ASP Solver.** The results produced by the ontology reasoner are translated back into ASP symbols and returned to Clingo through Python callbacks. From the solver’s perspective, ontology queries behave like ordinary external functions and can therefore participate in grounding and stable-model computation together with standard ASP predicates.

The OWL reasoner is exclusively responsible for semantic inference over the ontology, while Clingo performs non-monotonic reasoning, constraint satisfaction, and optimization. Such a modular design allows `clingowl` to operate over arbitrary OWL ontologies without translating ontology axioms into ASP facts or modifying existing reasoning engines.

The following sections describe the main technical components of the implementation, namely the definition of the theory language and the translation of OWL theory atoms into executable callbacks.

5.1. Theory Language

The integration between ASP and OWL reasoning is realized through Clingo’s theory language mechanism. Rather than extending the ASP syntax at the parser level, `clingowl` defines a custom theory that enables Description Logic expressions to appear inside ordinary ASP rules. Consequently, OWL expressions are parsed as theory atoms and become part of Clingo’s AST, allowing them to be processed during grounding.

The custom theory is specified using Clingo’s `#theory` directive, which defines both the available operators and their precedence.

The theory definition introduces a collection of operators that closely mirror the syntax of Description Logics. The complete list of currently supported class constructors and ontology axioms is reported in Appendix A, whereas the main text focuses on representative examples.

Both OWL theory atoms rely on the theory language mechanism of `clingo`, which parses OWL expressions as structured theory terms rather than plain text. Each operator occurring inside a theory atom is recognized according to the precedence specified in the `#theory` definition and represented directly in Clingo’s AST.

The parser associates every operator with a corresponding Description Logic constructor while preserving the precedence defined by the theory specification. Consequently, complex OWL expressions are represented as structured syntax trees, which considerably simplifies the subsequent translation phase, since each AST node directly corresponds to a semantic constructor.

This design provides two main advantages. First, it allows OWL expressions to be parsed directly by Clingo’s grammar, avoiding custom string-processing routines. Second, the resulting AST preserves the hierarchical structure of Description Logic expressions, enabling a straightforward recursive translation into OWLAPY objects, as described in the following section.

Original theory atom	Translated form
<code>&owlquery{ ~female } = X</code>	<code>@belongsto(negation(female)) = X</code>
<code>&owlquery{ hasChild ! male } = X</code>	<code>@belongsto(exist(hasChild,male)) = X</code>
<code>&owl{ peter :: father }</code>	<code>@axiom(instance(peter,father)) = 1</code>
<code>&owl{ mother <: person }</code>	<code>@axiom(subset(mother,person)) = 1</code>

Table 1

Examples of OWL theory atom translation.

5.2. Translation of Theory Atoms

Once an ASP program has been parsed, each OWL theory atom is translated into an internal representation that can be evaluated through Python callback functions during grounding. The translation is implemented by traversing Clingo’s AST: each Description Logic operator is recursively mapped to a symbolic constructor, preserving the hierarchical structure of the original expression. Table 1 illustrates this process on representative examples.

The translated program is an ordinary Clingo program containing external function calls. During grounding, these functions invoke the ontology interface to evaluate the requested reasoning task. Consequently, no modifications to the Clingo solver are required: ontology reasoning is embedded into the standard ASP execution pipeline.

This intermediate representation also provides a uniform mechanism for handling nested Description Logic expressions. Since theory atoms are recursively decomposed into symbolic constructors, the translation phase is independent of the ontology domain and can be applied to any OWL ontology supported by the underlying reasoner.

6. Evaluation

The objective of this evaluation is twofold. First, we measure the overhead introduced by the translation pipeline implemented by `clingowl`. Second, we compare `clingowl` with DLVHEX, a well-known ASP system supporting Description Logic interaction through external atoms.

Experiments were conducted on a machine equipped with an AMD Ryzen 5 7520U processor, 16 GB of RAM, running Microsoft Windows 11 Home. Both systems were executed inside Docker containers using Docker Desktop with a WSL2 Linux backend, ensuring reproducible and isolated execution environments on the same hardware. `clingowl` was executed using Clingo, OWLAPY, and the Pellet reasoner, whereas DLVHEX 2.5.0 was evaluated in a container reproducing its original software environment based on Ubuntu 16.04, GCC 4.8, and the Racer Description Logic reasoner. Each benchmark was repeated ten times, and the reported values correspond to mean runtime and standard deviation.

Two benchmark ontologies were considered. The first benchmark uses the running family ontology example introduced earlier in the paper. The second benchmark uses a fragment extracted from the SNOMED CT International release containing 1,424 concepts together with their complete role-group definitions. For both ontologies, synthetic ASP programs containing 10, 50, 100, and 500 ontology-aware atoms were automatically generated. This synthetic setup is intended to isolate the cost of ontology interaction rather than to model a complete application scenario.

6.1. Results

Table 2 reports the results of the experiments. Figure 2 provides a visual comparison of the runtime trends observed on the two benchmarks.

The experimental results reveal two substantially different execution models. For the family ontology, DLVHEX exhibits competitive performance and eventually outperforms `clingowl` as the number of ontology queries increases. This behaviour is expected since the benchmark operates on a very small ontology where ontology initialization costs are negligible.

Table 2

Overall execution time (ms). Speedup is computed as DLVHEX runtime divided by `clingowl` runtime.

Ontology	Atoms	<code>clingowl</code>	DLVHEX	Speedup
Family	10	34	195	5.7x
	50	140	172	1.2x
	100	269	212	0.8x
	500	1243	487	0.4x
SNOMED	10	31	15516	496x
	50	130	16509	127x
	100	249	16307	66x
	500	1131	16274	14x

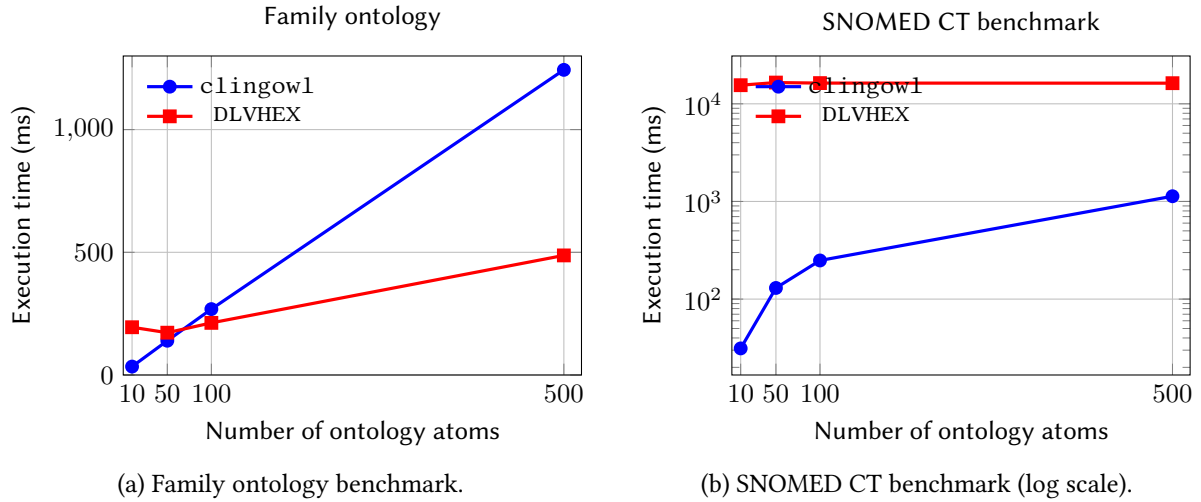


Figure 2: Runtime comparison between `clingowl` and DLVHEX. On the Family ontology (left), both systems show comparable performance on a small ontology. On the SNOMED CT benchmark (right), `clingowl` scales approximately linearly with the number of OWL theory atoms, whereas DLVHEX exhibits an almost constant runtime, suggesting a dominant ontology-initialization cost in our setup.

The behaviour changes significantly for the SNOMED CT benchmark. Here, `clingowl` consistently requires between approximately 30 ms and 1.1 s, whereas DLVHEX requires approximately 16 s, almost independently of the number of ontology queries. This behaviour likely reflects the different architectures adopted by the two systems. In our setup, DLVHEX appears to incur a substantial ontology initialization cost before ASP solving, which dominates execution time on the SNOMED benchmark. In contrast, `clingowl` performs lightweight AST translation and directly delegates ontology queries through OWLAPY, resulting in a runtime that grows approximately linearly with the number of OWL theory atoms.

Overall, the results suggest that `clingowl` is particularly well suited to ontology-aware applications involving repeated OWL queries, while maintaining low translation overhead.

7. Discussion & Related Work

The integration of Description Logics and rule-based reasoning has been extensively investigated over the last two decades.

One of the earliest and most influential approaches is DLVHEX, proposed by Eiter et al. [10]. The system extends Answer Set Programming with external atoms, allowing logic programs to exchange information with external knowledge sources, including Description Logic knowledge bases and RDF repositories. In this setting, ontology interaction is not part of the ASP language itself, but is delegated

to external plugins.

External knowledge is accessed through generic external atoms of the form

$$\&g[Input](Output)$$

whose semantics is defined outside the logic program. For example, the DLVHEX user guide introduces the following rule to retrieve all nodes reachable from a given source through an external computation:

```
1 reached(X) :- &reach[edge,a](X).
```

In this case, the meaning of the external atom is provided by a Python or C++ plugin rather than by the ASP language itself.

In `clingowl`, the same ontology-aware interaction is expressed through dedicated OWL theory atoms. For instance:

```
1 parent(X) :- &owlquery{ adult & (father | mother) } = X.
```

Unlike DLVHEX, `clingowl` provides OWL-specific theory atoms whose syntax closely mirrors Description Logic expressions. Ontology queries are therefore represented explicitly in the ASP program rather than being hidden inside external plugins. This results in a more declarative and readable interface while preserving the standard `clingo` programming model.

Another closely related approach is DataKlog [11], which introduces an epistemic extension of Disjunctive Datalog to query the knowledge bases of description logic. Unlike DLVHEX, which relies on generic external atoms, DataKlog proposes a new query language in which the interaction with the knowledge base is mediated by an epistemic operator, with the primary goal of preserving decidability while supporting recursion and non-monotonic reasoning. In contrast, `clingowl` focuses on providing a lightweight implementation that exposes OWL reasoning through dedicated theory atoms within the standard `clingo` workflow.

A different integration strategy is adopted by OntoDLV [21]. Rather than querying an external ontology, OntoDLV introduces its own ontology-oriented language, OntoDLP, with constructs such as classes, inheritance, attributes, and relations. Ontological knowledge is therefore modeled directly inside the ASP-based system, instead of being reused from an existing OWL ontology. In contrast, `clingowl` keeps the ontology external and accesses it through OWL theory atoms, allowing standard OWL ontologies to be reused without translation or reformulation.

The comparison above highlights four representative strategies for integrating Description Logics and Answer Set Programming. DLVHEX offers a general mechanism for interfacing ASP with arbitrary external sources through generic external atoms. DataKlog introduces an epistemic query language for interacting with Description Logic knowledge bases. OntoDLV embeds ontology-oriented constructs into a dedicated ASP-based language. `clingowl` instead adopts a domain-specific implementation based on OWL theory atoms, directly supporting Description Logic-style expressions inside standard `clingo` programs.

Table 3 summarizes the main design differences between `clingowl` and DLVHEX. While both systems support ontology-aware reasoning in ASP, they follow substantially different integration models. DLVHEX relies on generic external atoms, whereas `clingowl` introduces dedicated OWL theory atoms supporting expressive ontology queries directly in the language.

Overall, `clingowl` follows a different design philosophy from previous systems. Rather than introducing a new ontology language or relying on generic external interfaces, it extends the standard `clingo` language with OWL-specific theory atoms. This provides a lightweight integration layer between OWL and ASP, keeps existing ontologies external to the logic program, and delegates ontological inference to standard Description Logic reasoners through OWLAPY, while ASP remains responsible for rule evaluation, constraint solving, and non-monotonic reasoning.

Table 3Comparison between `clingowl` and `DLVHEX`.

Feature	<code>clingowl</code>	<code>DLVHEX</code>
Dedicated OWL syntax	Yes	No
Complex OWL class expressions	Yes	Plugin-dependent
OWL theory atoms	Yes	No
Generic external atoms	No	Yes
External reasoner	Pellet/HermiT/etc.	Racer
Ontology translation required	No	No
Clingo theory language	Yes	No

8. Conclusion and Future Work

This paper presented `clingowl`, an extension of `clingo` that introduces dedicated OWL theory atoms for ontology-aware reasoning over OWL ontologies. By combining Clingo’s theory language, AST-based translation, OWLAPY, and external Description Logic reasoners, `clingowl` enables ASP programs to query ontological knowledge without translating OWL ontologies into ASP facts. The system preserves the standard ASP programming model while providing a lightweight and modular interface between declarative reasoning and semantic knowledge.

The experimental results provide a first empirical assessment of the proposed approach. In particular, they show that `clingowl` introduces low translation overhead and scales favourably on ontology-aware benchmarks over expressive OWL ontologies such as SNOMED CT. The comparison with `DLVHEX` further highlights the impact of the underlying integration architecture: while both systems support ontology-aware ASP reasoning, `clingowl` benefits from a direct OWL-oriented interface and exhibits significantly lower runtime on larger ontological benchmarks.

Several directions remain open for future work. First, the experimental study should be extended with additional benchmark families, including more realistic ASP encodings, larger ontology fragments, and a broader set of reasoning tasks. Second, a more comprehensive comparison with alternative ontology-aware ASP approaches, including OntoDLV when available, would provide further insight into trade-offs between expressiveness, usability, and performance. Since `clingowl` delegates ontology reasoning through OWLAPY, it can also switch to different DL reasoners with minimal changes in the code. In particular, for large ontologies in the OWL 2 EL profile such as SNOMED CT, a specialized reasoner such as ELK may be more appropriate than Pellet or HermiT, although it would not support the more expressive family benchmark queries considered in this paper. Third, the current framework can be enriched with additional ontology-querying constructs, such as mechanisms for retrieving subclasses or other concept-level results directly from OWL ontologies. Finally, an interesting direction is the integration of explanation facilities, combining ontology-aware ASP reasoning with explainable symbolic techniques.

Declaration on Generative AI

During the preparation of this work, the authors used Grammarly and ChatGPT (OpenAI) to assist with grammar and spelling checks, as well as paraphrasing and rewording of certain sentences. After using these tools, the authors carefully reviewed and edited the content to ensure accuracy, coherence, and compliance with academic standards. The authors take full responsibility for the content of this publication.

Acknowledgments

Research partially supported by the PNRR Project CUP E13C24000430006 “Enhanced Network of intelligent Agents for Building Livable Environments - ENABLE”, and by PRIN 2022 CUP E53D23007850001 Project TrustPACTX - Design of the Hybrid Society Humans-Autonomous Systems: Architecture, Trustworthiness, Trust, Ethics, and EXplainability (the case of Patient Care), and by PRIN PNRR CUP E53D23016270001 ADVISOR - ADaptiVe legIble robotS for trustwORthy health coaching. This research was also partially supported by the MICIU/AEI/10.13039/501100011033 grant PID2023-148531NB-I00 (Spain). The authors would like to thank Yare Brea Espinosa for his valuable help in setting up the DLVHEX environment and in preparing the benchmark scripts used for the comparative evaluation.

References

- [1] M. Elhaddad, S. Hamam, Ai-driven clinical decision support systems: an ongoing pursuit of potential, *Cureus* 16 (2024) e57728.
- [2] J. Hu, L. Ren, T. Wang, P. Yao, Artificial intelligence-assisted clinical decision-making: a perspective on advancing personalized precision medicine for elderly diabetes patients, *Journal of Multidisciplinary Healthcare* (2025) 4643–4651.
- [3] J. V. Cordeiro, Digital technologies and data science as health enablers: an outline of appealing promises and compelling ethical, legal, and social challenges, *Frontiers in medicine* 8 (2021) 647897.
- [4] V. L. Raposo, The fifty shades of black: about black box ai and explainability in healthcare, *Medical Law Review* 33 (2025) fwaf005.
- [5] R. Guntupalli, Explainable ai in clinical decision support: Interpretable neural models for trustworthy healthcare automation, *TPM–Testing, Psychometrics, Methodology in Applied Psychology* 32 (2025) 462–471.
- [6] F. Baader, I. Horrocks, U. Sattler, Description logics, in: *Handbook on ontologies*, Springer, 2009, pp. 21–43.
- [7] A. Vozna, A. Monaldini, S. Costantini, V. Pitoni, Scalable and explainable diet recommendations via answer set programming, in: *Proceedings of the First International Workshop on Human-AI Collaborative Systems (HAIC 2025)*, volume 4072 of *CEUR Workshop Proceedings*, Bologna, Italy, 2025.
- [8] T. Eiter, G. Ianni, T. Lukasiewicz, R. Schindlauer, H. Tompits, Combining answer set programming with description logics for the semantic web, *Artificial intelligence* 172 (2008) 1495–1539.
- [9] T. Eiter, M. Fink, D. Stepanova, Computing repairs of inconsistent dl-programs over el ontologies, *Journal of Artificial Intelligence Research* 56 (2016) 463–515.
- [10] T. Eiter, G. Ianni, R. Schindlauer, H. Tompits, et al., dlhex: A tool for semantic-web reasoning under the answer-set semantics, in: *Proceedings of International Workshop on Applications of Logic Programming in the Semantic Web and Semantic Web Services*, volume 196, 2006, pp. 33–39.
- [11] G. Cima, M. Console, M. Lenzerini, A. Poggi, Epistemic disjunctive datalog for querying knowledge bases, in: *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, 2023, pp. 6280–6288.
- [12] A. Baci, L. Friedrichs, C. Demir, A.-C. N. Ngomo, Owlapy: A pythonic framework for owl ontology engineering, 2025. URL: <https://arxiv.org/abs/2511.08232>. *arXiv:2511.08232*.
- [13] M. Krötzsch, *Description logic rules*, volume 8, IOS Press, 2010.
- [14] I. Horrocks, O. Kutz, U. Sattler, The even more irresistible sroiq., *Kr* 6 (2006) 57–67.
- [15] P. Cabalar, S. Costantini, A. Vozna, Here-and-there with constraints for description logic, in: *Proceedings of the 17th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR 2026)*, 2026.
- [16] S. El-Sappagh, F. Franda, F. Ali, K.-S. Kwak, Snomed ct standard ontology based on the ontology for general medical science, *BMC medical informatics and decision making* 18 (2018) 76.

- [17] Y. Kazakov, M. Krötzsch, F. Simancik, Elk reasoner: architecture and evaluation., in: ORE, 2012.
- [18] V. M. Giménez-Solano, J. A. Maldonado, D. Boscá, S. Salas-García, M. Robles, Definition and validation of snomed ct subsets using the expression constraint language, *Journal of Biomedical Informatics* 117 (2021) 103747.
- [19] A. Rector, L. Iannone, Lexically suggest, logically define: Quality assurance of the use of qualifiers and expected results of post-coordination in snomed ct, *Journal of Biomedical Informatics* 45 (2012) 199–209. doi:10.1016/j.jbi.2011.10.002.
- [20] F. Abad-Navarro, M. Quesada-Martínez, A. Duque-Ramos, J. T. Fernández-Breis, Analysis of readability and structural accuracy in snomed ct, *BMC Medical Informatics and Decision Making* 20 (2020) 284.
- [21] F. Ricca, L. Gallucci, R. Schindlauer, T. Dell’Armi, G. Grasso, N. Leone, Ontodlv: an asp-based system for enterprise ontologies, *Journal of Logic and Computation* 19 (2009) 643–670.

A. Supported Syntax

The theory definition of `clingowl` introduces a collection of operators that closely mirror the syntax of Description Logics. Table 4 summarizes the supported class constructors, whereas Table 5 reports the ontology axioms currently supported by `clingowl`.

Table 4

Supported OWL class constructors.

Syntax	Description
$A \ \& \ B$	Class intersection ($A \sqcap B$)
$A \ \ B$	Class union ($A \sqcup B$)
$\sim A$	Class complement ($\neg A$)
$R \ ! \ C$	Existential restriction ($\exists R.C$)
$R \ ? \ C$	Universal restriction ($\forall R.C$)
$\neg R$	Inverse object property
$\{a\}$	Nominal
$\{a, b, c\}$	Enumeration of individuals
<code>thing</code>	OWL Thing
<code>nothing</code>	OWL Nothing

Table 5

Supported OWL axioms.

Syntax	Description
$C \prec D$	Subclass axiom
$C = D$	Equivalent classes
$a \ :: \ C$	Class assertion
$(a, b) \ :: \ R$	Object property assertion