

Paraconsistent Query Answering in Hybrid Knowledge Bases for Aeronautics Applications^{*}

Arun Raveendran Nair Sheela^{1,2}, Christophe Rey³ and Florence De Grancey²

¹Université Clermont Auvergne, LIMOS, France

³Université Clermont Auvergne, LIMOS, CNRS, France

²Thales, France

Abstract

Hybrid MKNF knowledge bases provide an approach for combining Description Logics (DLs) and Logic Programming, enabling reasoning under both open-world and closed-world assumptions. However, existing approaches under the well-founded semantics do not support classical negation in the logic programming component. To address this limitation, hMKNF $\neg$ knowledge bases extend Hybrid MKNF with classical negation. A remaining challenge is how to answer queries when contradictions occur in the knowledge base.

To address this challenge, we propose a paraconsistent counterpart of hMKNF $\neg$ knowledge bases by replacing the classical DL ontology with a paraconsistent DL ontology. We then introduce a transformation pipeline that rewrites paraconsistent hMKNF $\neg$ knowledge bases into consistent Hybrid MKNF knowledge bases, enabling the reuse of existing query answering algorithms. To support efficient query answering, we restrict the DL component to Datalog-rewritable DLs and rewrite the resulting knowledge base into a set of logic programming rules. Based on this rewriting, we develop two query answering strategies: a bottom-up approach using the well-founded operator and a top-down approach using SLG resolution. Query answers are classified into four statuses: consistently true, consistently false, contradictory, or undefined. Finally, we implement the proposed approaches and evaluate them on a realistic aviation use case, demonstrating their applicability to query answering in the presence of inconsistent information.

Keywords

Hybrid MKNF, Description Logics, Logic Programming, Well-founded Semantics, Query Answering

1. Introduction

The aeronautics domain involves large amounts of dynamic and safety-critical information, including flight plans, NOTAMs, weather conditions, airport data, and airspace restrictions. As the amount of aeronautical data continuously grows, systems capable of processing and reasoning over large volumes of information become increasingly important for aeronautical stakeholders. Description Logics (DLs) are useful for representing this knowledge in a structured way, for example by organizing airports, aircraft, airspaces, and flight plans into clear concepts and relationships. However, DLs alone are not sufficient because aeronautical applications also need to handle operational procedures, special cases, numerical validations, and decisions that may change according to real-time conditions. Logic Programming (LP) systems can express such operational logic, but they are also insufficient on their own because they do not naturally provide the semantic structure and hierarchy required to properly model the domain. For this reason, a hybrid knowledge base combining DLs and logic programming is particularly suitable for aeronautical use cases. It combines structured knowledge representation with flexible reasoning, enabling the management of complex domain knowledge and operational conditions.

Many research efforts have focused on combining DLs and logic programming in order to take advantage of the strengths of both approaches. Among the existing approaches, Hybrid MKNF stands out as a highly expressive formalism, as it integrates DL ontologies with logic programming rules under the semantics of Minimal Knowledge and Negation as Failure (MKNF), extending first-order logic with the modal operators **K** and **not** [1, 2]. In [3], a three-valued semantics is introduced to capture the well-founded semantics. The well-founded semantics is particularly suitable for query answering because it supports efficient reasoning and can be computed in polynomial time, assuming polynomial-time DL

RuleML+RR'26: Companion Proceedings of the 10th International Joint Conference on Rules and Reasoning, August 24–26, 2026, Vilnius, Lithuania



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

reasoning. This is especially important in the aeronautics domain, where large amounts of continuously evolving information must be processed efficiently to provide recommendations and decision support through query answering based on real-world conditions. However, one limitation of this approach is that classical negation is not allowed in the logic programming component. This limitation is addressed in [4, 5], where the logic programming component is extended with classical negation, leading to the language hMKNF $\neg$.

In the design of a hybrid knowledge base using hMKNF $\neg$, the DL component represents the static and structural knowledge of the domain by describing the main entities and their relationships, such as airports, runways, airspaces, and infrastructure specifications. In contrast, the Logic Program component captures decision logic and supports reasoning over dynamic and evolving information obtained from real-time data sources, allowing the system to adapt its conclusions and recommendations according to real-world conditions. While this separation enables the representation of both static and dynamic knowledge, reasoning in safety-critical domains often requires distinguishing between missing information and information that is known to be false. To support this distinction, hMKNF $\neg$ allows the use of classical negation ($\neg$) within the LP component. Unlike Negation as Failure (NAF), which derives negative conclusions from the absence of information, classical negation allows the explicit representation of verified negative knowledge. This distinction is particularly important in aeronautical applications, where missing or delayed information cannot be interpreted as false.

Consider the following rule, which states that a runway is open if it is associated with an airport and is not known to be closed or obstructed:

$$\mathbf{KOpenRunway}(Y) \leftarrow \mathbf{KAirport}(A), \mathbf{KhasRunway}(A, Y), \mathbf{not\ obstacle}(A, Y), \mathbf{not\ ClosedRunway}(Y).$$

The condition **not** obstacle(A, Y) only indicates that no information about an obstacle is available; it does not imply that the runway is known to be obstacle-free. As obstacle information may be incomplete or delayed, the rule can derive OpenRunway(Y) solely from the absence of evidence of an obstacle rather than from explicit evidence that no obstacle exists. To address this limitation, classical negation can be used to explicitly represent confirmed negative information:

$$\mathbf{KOpenRunway}(Y) \leftarrow \mathbf{KAirport}(A), \mathbf{KhasRunway}(A, Y), \mathbf{K}\neg\mathbf{obstacle}(A, Y), \mathbf{not\ ClosedRunway}(Y).$$

Furthermore, classical negation in rule heads also supports the representation of integrity constraints similar to [6, 7, 8]. For example, $\mathbf{K}\neg\mathbf{Operational}(\text{rwy}) \leftarrow \mathbf{Kobstacle}(\text{rwy})$ states that a runway is not operational whenever an obstacle is detected. If the knowledge base also derives $\mathbf{KOperational}(\text{rwy})$, then an explicit contradiction arises. During query answering, such contradictions can be identified and excluded from the generated recommendations, allowing contradictory conclusions to be identified and excluded from generated recommendations.

In hMKNF $\neg$ knowledge bases, inconsistencies may arise within the DL component, from the interaction between the DL and LP components, or within the LP component itself. Existing computation mechanisms handle inconsistencies using an explosive semantics, where the presence of a contradiction causes every formula to become derivable. Consequently, this approach is unsuitable for query answering, since meaningful answers can no longer be inferred. Therefore, the main objective of this work is to develop a conjunctive query answering approach for hMKNF $\neg$ knowledge bases that can reason effectively in the presence of contradictions.

Two main strategies exist for reasoning and query answering in the presence of contradictions. The first, known as the repair-based or belief revision approach, treats contradictions as errors and attempts to restore consistency by modifying the knowledge base. While this approach is useful when dealing with noisy or unreliable data, it may also discard important information and require complex repair procedures. In addition, contradictory information may reflect genuine real-world situations that indicate potential problems, and removing such information can hide important operational issues. The second strategy is the paraconsistent approach, which allows reasoning to continue even when contradictory information is present. Instead of discarding conflicting facts, paraconsistent reasoning explicitly represents and manages them, so that they can be taken into account during query answering.

This is particularly important in dynamic and data-intensive domains such as aeronautics, where contradictions may arise from incomplete, delayed, or heterogeneous data sources. In this setting, query answering is not only used to retrieve information, but also to identify reliable conclusions and exclude answers that depend on unresolved contradictions. As a result, contradictory information is treated as an indication of possible conflicts or abnormal situations that should be analyzed, rather than simply removed. In this work, we develop a query answering procedure for hMKNF $\neg$ knowledge bases under paraconsistent semantics. Our main contributions are as follows:

1. Section 4 defines the paraconsistent counterpart of a hMKNF $\neg$ knowledge base by combining a paraconsistent DL ontology with logic programming rules that support classical negation.
2. Section 5 presents an overview of the complete query answering procedure.
3. Section 6 describes the transformation of a paraconsistent hMKNF $\neg$ knowledge base into set of logic programming rules.
4. Section 7 presents the bottom-up and top-down procedures for answering queries over the transformed hMKNF $\neg$ knowledge base.
5. Section 8 presents the implementation of the proposed query answering procedures.

2. Use Case

The case study focuses on a flight decision-support system designed to assist pilots during both pre-flight planning and in-flight operations. The system integrates information from multiple heterogeneous sources, including airport databases, airspace specifications, weather services, flight plans, aircraft status information, and NOTAM messages. The objective is to support safe and efficient flight operations by providing recommendations and answering operational queries. A typical scenario involves a pilot preparing a flight plan and assessing whether the planned trajectory satisfies operational and safety requirements. During flight, changing weather conditions, newly issued NOTAMs, or airspace restrictions may require the route to be re-evaluated and alternative actions to be considered.

Knowledge Representation. The use case is modeled using the hMKNF $\neg$ language. DL axioms are used to represent the entities involved in flight operations, their taxonomic hierarchy, and the relationships between them. For example, the airspace hierarchy can be represented using subclass axioms such as $\text{ControlledAirspace} \sqsubseteq \text{Airspace}$ and $\text{RestrictedAirspace} \sqsubseteq \text{Airspace}$, which specify that controlled and restricted airspaces are specialized categories of airspace. Additional axioms capture structural properties of these entities. For instance, $\text{Airspace} \sqsubseteq \exists \text{hasAltitudeLimit}.\text{Altitude}$ states that every airspace is associated with an altitude limit. Logic programming rules are used to model operational actions, exceptional situations, and recommendations that support pilot decision making. The knowledge base also contains assertions obtained from multiple sources. Some assertions describe static information, such as airports, runways, and airspace specifications, while others capture real-time information, including weather reports, NOTAM messages, flight plans, and aircraft positions. For example, $\text{ProhibitedAirspace}(a_1)$, $\text{hasAltitudeLimit}(a_1, 35000)$, and $\text{intersectsTrajectory}(a_1, f_1)$ represent information about airspace restrictions and their interaction with a planned flight trajectory.

The knowledge base also incorporates a set of different types of constraints that govern the validity of query answers, recommendations, and generated actions. Structural constraints ensure the correctness and completeness of aeronautical data. For example, airports must be associated with runways, runways must contain the required operational information, and airspaces must be defined with valid geometries and altitude limits. Operational and safety constraints regulate flight operations by ensuring that flight trajectories do not intersect prohibited airspaces, restricted airspaces are entered only under authorized conditions, and required separation distances are maintained. These constraints ensure compliance with both operational requirements and aviation safety regulations.

Reasoning Requirements. The primary reasoning task supported by the system is query answering. Pilots must be able to pose operational queries related to flight safety and mission feasibility. Typical

queries include identifying NOTAMs that affect a planned flight trajectory, detecting intersections with restricted or prohibited airspaces, verifying compliance with operational and environmental constraints, and selecting suitable diversion airports according to aircraft capabilities and mission requirements.

3. Preliminaries

3.1. Paraconsistent Description Logics

Paraconsistent DLs extend classical DLs with the ability to reason in the presence of inconsistent information. A prominent approach to handling inconsistencies in DLs employs paraconsistent four-valued semantics, with notable formalisms including $\mathcal{ALC}4$ [9] and $\mathcal{SROIQ}4$ [10]. In these logics, a concept assertion may be true (**t**), false (**f**), unknown (**u**), or contradictory (**b**). This allows contradictions to be represented explicitly while avoiding the explosive behavior of classical logic. Syntactically, paraconsistent DLs and classical DLs do not differ significantly, as complex concepts and assertions are defined in the same way. However, classical concept inclusion axioms are replaced by three forms of inclusion: material inclusion, internal inclusion, and strong inclusion. These different forms of inclusion are obtained by weakening classical concept inclusion and differ in how they propagate information and inconsistencies during the reasoning process.

1. *Strong inclusion.* Strong inclusion is the strongest form of inclusion and behaves similarly to classical implication. A *strong inclusion* from φ to ψ , denoted by $\varphi \rightarrow \psi$, states that φ implies ψ and that $\neg\psi$ implies $\neg\varphi$. It therefore supports both forward reasoning and contraposition, allowing information and inconsistencies to propagate in both directions. For example, $\text{ClosedRunway} \rightarrow \text{Runway}$ states that every closed runway is a runway, while negative information about Runway can also propagate backward to ClosedRunway.
2. *Internal inclusion.* Internal inclusion is weaker than strong inclusion and supports only forward propagation. An *internal inclusion* from φ to ψ , denoted by $\varphi \sqsubset \psi$, states that φ implies ψ , but does not support contraposition; hence, $\neg\psi$ does not imply $\neg\varphi$. For example, $\text{OpenRunway} \sqsubset \text{Runway}$ states that every open runway is a runway, so information is propagated from OpenRunway to Runway, while negative information about Runway is not propagated backward to OpenRunway. This provides more controlled propagation of information and inconsistencies.
3. *Material inclusion.* Material inclusion is the weakest form of inclusion and is defined using material implication. A *material inclusion* from φ to ψ , denoted by $\varphi \mapsto \psi$, is defined as $\varphi \mapsto \psi \equiv \neg\varphi \vee \psi$. Unlike strong and internal inclusion, material inclusion does not enforce the same propagation of contradictory information between φ and ψ . For example, $\text{RunwayOperational} \mapsto \text{FlightAllowed}$ expresses that either the runway is not operational or the flight is allowed, without introducing the stronger propagation behavior associated with strong or internal inclusion.

The semantics of four-valued DLs extends classical DL semantics by interpreting each concept and role using a pair of sets. A four-valued interpretation is a tuple $I = \langle \Delta^I, \cdot^I \rangle$, where Δ^I is a non-empty domain and $\cdot^I$ is a valuation function. For a concept C , we write $C^I = \langle P, N \rangle$, where P denotes the positive extension of C and N its negative extension. Thus, for an individual a , $C^I(a) = \mathbf{t}$ iff $a^I \in P$ and $a^I \notin N$, $C^I(a) = \mathbf{f}$ iff $a^I \notin P$ and $a^I \in N$, $C^I(a) = \mathbf{b}$ iff $a^I \in P$ and $a^I \in N$, and $C^I(a) = \mathbf{u}$ iff $a^I \notin P$ and $a^I \notin N$.

Role interpretations are defined analogously. A four-valued interpretation I 4-satisfies an axiom A if A holds *true* under the four-valued semantics. A knowledge base $\mathcal{K}$ is 4-satisfiable if there exists at least one four-valued interpretation that 4-satisfies every axiom in $\mathcal{K}$. An axiom A is entailed by $\mathcal{K}$ under paraconsistent semantics, written $\mathcal{K} \models_p A$, if A is satisfied in every four-valued model of $\mathcal{K}$. For the full formal definition of the semantics, we refer the reader to [10, 9].

3.2. hMKNF $\neg$ Knowledge Bases

The hMKNF $\neg$ knowledge base extends Hybrid MKNF knowledge bases with classical negation in the logic programming component under the well-founded semantics proposed by [3]. A hMKNF $\neg$ knowledge base is a pair $\mathcal{K} = (\mathcal{O}, \mathcal{P})$, where $\mathcal{O}$ is a decidable DL knowledge base that can be translated into function-free first-order logic with equality, and $\mathcal{P}$ is a finite set of MKNF rules of the form [11]:

$$\mathbf{K} H \leftarrow \mathbf{K} A_1, \dots, \mathbf{K} A_n, \mathbf{not} B_1, \dots, \mathbf{not} B_m. \quad (1)$$

Here, H , A_i , and B_j are function-free first-order atoms or their classical negation $\neg A$. A rule is *positive* if $m = 0$, and a *fact* if $n = m = 0$. An atom is called a *DL-atom* if its predicate occurs in $\mathcal{O}$; otherwise, it is a *non-DL atom*. A rule is *ground* if it contains no variables.

For a rule r of form 1, we use the notation $h(r) \leftarrow b^+(r), b^-(r)$, where $h(r) = \mathbf{K} H$, $b^+(r) = \{\mathbf{K} A_1, \dots, \mathbf{K} A_n\}$, and $b^-(r) = \{\mathbf{not} B_1, \dots, \mathbf{not} B_m\}$. We further define $\mathbf{K}(b^-(r)) = \{\mathbf{K} a \mid \mathbf{not} a \in b^-(r)\}$ and $b(r) = b^+(r) \cup b^-(r)$. An MKNF rule is *DL-safe* if every variable occurring in the rule also appears in a non-DL atom in the rule body. A hMKNF $\neg$ knowledge base is DL-safe if all its rules are DL-safe. Decidability follows from the decidability of the underlying DL, the use of function-free atoms, and DL-safety.

The semantics of hMKNF $\neg$ knowledge bases are defined through a translation function π that maps a hMKNF $\neg$ knowledge base $\mathcal{K} = (\mathcal{O}, \mathcal{P})$ into an MKNF formula. The translation $\pi(\mathcal{O})$ denotes the first-order representation of the ontology $\mathcal{O}$, while $\pi(\mathcal{P})$ denotes the modal translation of the rule set $\mathcal{P}$, containing expressions such as $\mathbf{K}A$, $\mathbf{not} A$, $\mathbf{K}\neg A$, and $\mathbf{not} \neg A$. The overall translation of the knowledge base, denoted by $\pi(\mathcal{K})$, is defined as $\pi(\mathcal{K}) = \mathbf{K} \pi(\mathcal{O}) \wedge \pi(\mathcal{P})$. The semantics of $\pi(\mathcal{K})$ can be explained using either answer set [12] or well-founded semantics [3]. From the use case, we saw that query answering is the primary reasoning task. Therefore, well-founded semantics is the most suitable because it provides a unique three-valued model, consisting of true, false, and undefined atoms, thereby enabling efficient top-down query answering. The formal semantics of hMKNF $\neg$ and the computation of its well-founded model are described in [4].

Example 1. Consider the hMKNF $\neg$ knowledge base $\mathcal{K} = (\mathcal{O}, \mathcal{P})$, where the ontology $\mathcal{O}$ consists of:

$$\text{ClosedAS} \equiv \text{RestrictedAS} \sqcap \exists \text{closedBy}. \text{ClosureNOTAM}, \quad (2)$$

$$\text{ClosedAS} \equiv \neg \text{OpenAS}, \quad (3)$$

$$\text{OpenAS} \equiv \neg \text{ClosedAS}, \quad (4)$$

$$\text{EmergencyAS} \sqsubseteq \text{OpenAS}, \quad (5)$$

$$\text{OpenRunway} \sqsubseteq \text{Runway}, \quad (6)$$

$$\begin{aligned} &\text{RestrictedAS}(a), \quad \text{ClosureNOTAM}(n), \quad \text{closedBy}(a, n), \\ &\text{EmergencyAS}(a), \quad \text{Runway}(rw1), \quad \text{Crosses}(f, a), \quad \text{PriorityFlight}(f). \end{aligned} \quad (7)$$

The logic programming component $\mathcal{P}$ is

$$\mathbf{K} \neg \text{SafeToCross}(A) \leftarrow \mathbf{K} \text{ClosedAS}(A), \mathbf{not} \text{SafeToCross}(A). \quad (8)$$

$$\mathbf{K} \text{reroute}(F, A) \leftarrow \mathbf{K} \text{Crosses}(F, A), \mathbf{K} \neg \text{SafeToCross}(A), \mathbf{not} \text{continue}(F). \quad (9)$$

$$\mathbf{K} \text{continue}(F) \leftarrow \mathbf{K} \text{PriorityFlight}(F), \mathbf{not} \text{reroute}(F, A). \quad (10)$$

$$\mathbf{K} \text{delayed}(F) \leftarrow \mathbf{K} \text{reroute}(F, A), \mathbf{not} \text{continue}(F). \quad (11)$$

$$\mathbf{K} \text{OpenRunway}(R) \leftarrow \mathbf{K} \text{Runway}(R), \mathbf{not} \text{ClosedRunway}(R). \quad (12)$$

Axioms (2)–(5), together with the assertions in (7), introduce contradictory information about airspace a . In particular, Axiom (2) derives $\text{ClosedAS}(a)$, since a is restricted and associated with a closure NOTAM. At the same time, Axiom (5) derives $\text{OpenAS}(a)$, since a is an emergency airspace. Consequently, Axioms (3) and (4) derive $\neg \text{OpenAS}(a)$ and $\neg \text{ClosedAS}(a)$, respectively. Hence, the ontology contains contradictory information about the status of airspace a . Rules (8)–(11) use this information to determine whether the airspace is safe to cross and whether the flight should be rerouted, continued, or delayed. Rule (12) illustrates default reasoning by considering a runway open unless it is known to be closed.

4. Paraconsistent hMKNF $\neg$

One of the main challenges in query answering over a hMKNF $\neg$ knowledge base is the presence of contradictory information. A knowledge base may become MKNF-inconsistent for several reasons: the ontology $\mathcal{O}$ may itself be inconsistent, contradictions may arise from the interaction between the ontology $\mathcal{O}$ and the rules $\mathcal{P}$, or conflicting conclusions such as A and $\neg A$ may be derived by the rules. Example 1 illustrates the first case, where contradictory information in the DL ontology makes the knowledge base MKNF-inconsistent. Under the standard MKNF semantics, an inconsistency arising from even a small part of the knowledge base may lead to the absence of a well-founded MKNF model. Consequently, query answering may no longer be possible, even though the remaining knowledge is consistent and useful. This motivates a paraconsistent approach that supports query answering without allowing local contradictions to invalidate the entire knowledge base.

Paraconsistent reasoning supports reasoning in the presence of contradictory information without invalidating the entire knowledge base. Based on this principle, we introduce a paraconsistent version of hMKNF $\neg$. In our approach, the logic programming component remains unchanged, while the DL ontology is transformed into a paraconsistent DL ontology by replacing classical inclusions with *internal inclusions*.

Formally, given a hMKNF $\neg$ knowledge base $\mathcal{K} = (\mathcal{O}, \mathcal{P})$, its paraconsistent counterpart is defined as $\mathcal{K}^p = (\mathcal{O}^p, \mathcal{P})$, where $\mathcal{O}^p$ is obtained from $\mathcal{O}$ by replacing each classical concept inclusion $C \sqsubseteq D$ with the corresponding internal inclusion $C \sqsubset D$. The logic programming component $\mathcal{P}$ remains unchanged. The semantics of the paraconsistent version of hMKNF $\neg$ is obtained by translating the knowledge base into MKNF formulae and is closely related to the paraconsistent well-founded semantics presented in [13].

Example 2. Consider the hMKNF $\neg$ knowledge base $\mathcal{K} = (\mathcal{O}, \mathcal{P})$ from Example 1. Its paraconsistent counterpart is $\mathcal{K}^p = (\mathcal{O}^p, \mathcal{P})$, where $\mathcal{O}^p$ contains the following axioms:

$$\text{ClosedAS} \equiv \text{RestrictedAS} \sqcap \exists \text{closedBy.ClosureNOTAM} \quad (13)$$

$$\text{ClosedAS} \equiv \neg \text{OpenAS} \quad (14)$$

$$\text{OpenAS} \equiv \neg \text{ClosedAS} \quad (15)$$

$$\text{EmergencyAS} \sqsubset \text{OpenAS} \quad (16)$$

$$\text{OpenRunway} \sqsubset \text{Runway}. \quad (17)$$

Here, $\sqsubset$ denotes internal concept inclusion. The logic programming component $\mathcal{P}$ remains the same as in Example 1.

Intuitively, in the paraconsistent counterpart of a hMKNF $\neg$ knowledge base, contradictions are propagated through the knowledge base, and each atom is classified as consistently true, consistently false, undefined, or contradictory. The paraconsistent counterpart is obtained by weakening the logic, in particular by restricting the DL component to internal inclusions. Consequently, when the original hMKNF $\neg$ knowledge base is consistent, its paraconsistent counterpart may derive fewer conclusions than under the standard hMKNF $\neg$ semantics.

5. Query Answering Idea

This section presents the overall idea of the proposed query answering approach. The main idea is to transform the paraconsistent counterpart of a hMKNF $\neg$ knowledge base into a form that can be evaluated using existing reasoning procedures.

The transformation builds on existing approaches for paraconsistent DLs and extended logic programs. The paraconsistent DL ontology is transformed into a consistent classical DL ontology by replacing classically negated concepts and roles with fresh symbols [14, 10]. The same idea is applied to the logic

program, where classically negated atoms are replaced by fresh predicates [9, 15]. In this way, classical negation is removed while positive and negative information remain represented separately.

The resulting knowledge base consists of a classical DL ontology and a logic program without classical negation. It can therefore be treated as a three-valued Hybrid MKNF knowledge base and evaluated using existing query answering procedures. In particular, query answering can be performed bottom-up using the well-founded operator of [16], or top-down using the query-driven SLG(O) procedure of [17]. However, these procedures may require repeated interaction between the logic programming engine and a DL reasoner during query evaluation.

To avoid this interaction, we focus on $\mathcal{SROEL}$, for which the DL ontology can be rewritten as Datalog rules. The transformed ontology is therefore rewritten into Datalog and combined with the transformed logic program, resulting in a single logic program. Knowledge base doubling [15] is then applied to preserve the coherency principle between classical and default negation. The resulting program can be evaluated entirely at the logic programming level using either the bottom-up or the top-down reasoning approach.

The overall query answering workflow is illustrated in Figure 1.

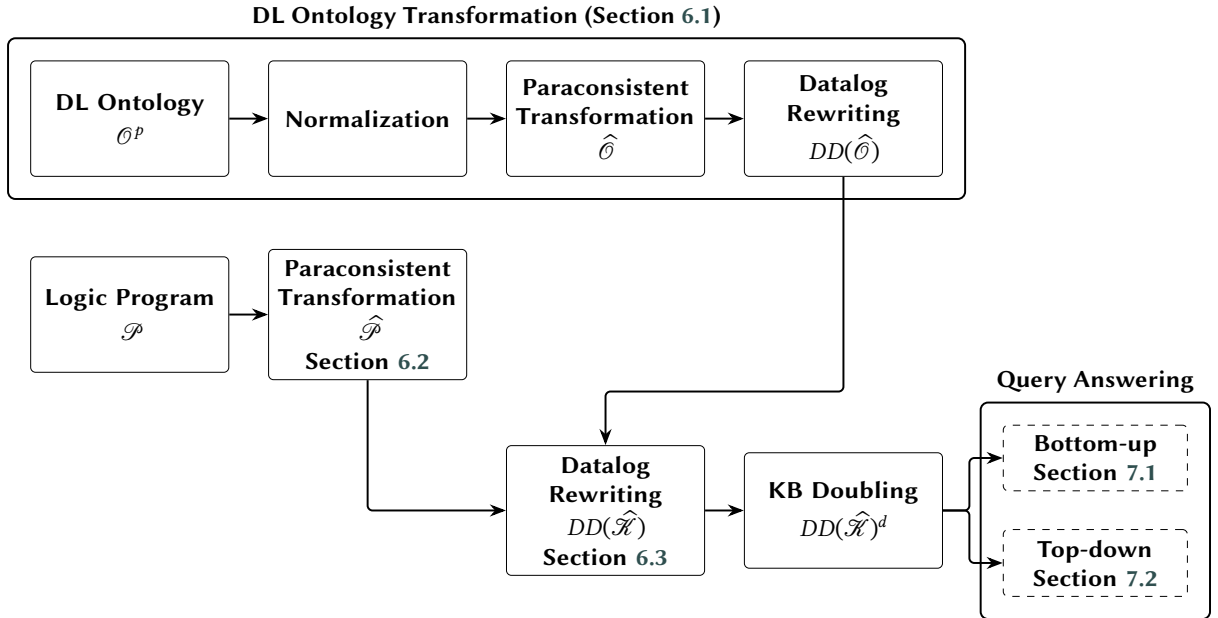


Figure 1: Query answering workflow

After the transformation, query answering is performed entirely over the resulting logic program. In the bottom-up approach, the DL entailment checks required by the well-founded operator are no longer necessary, since the consequences of the DL ontology are already represented by the rewritten Datalog rules. The standard well-founded operator for logic programs [18, 19] can therefore be applied directly to the transformed program. The least fixpoint of this operator, called the *paraconsistent well-founded partition*, is computed and stored in a database. Queries are then evaluated against the stored data.

In the top-down approach, the oracle resolution used by SLG(O) [17] to obtain DL entailment results is no longer required. Since the transformed ontology has already been rewritten as Datalog rules and integrated with the logic program, standard SLG resolution [20] can be applied directly to answer queries. For both approaches, the classification conditions are applied to the computed results to determine whether each query answer is *consistently true*, *consistently false*, *contradictory*, or *undefined*. In the bottom-up approach, queries are evaluated over the precomputed results stored in the database, whereas the top-down approach computes the results directly for each query.

6. Transformation

In this section, we describe a transformation pipeline for paraconsistent hMKNF $\neg$ knowledge bases. The objective is to transform a paraconsistent hMKNF $\neg$ knowledge base into a set of logic programming rules that can be evaluated using existing logic programming reasoning procedures. The transformation consists of three main parts: the DL transformation, which transforms the paraconsistent DL ontology and rewrites it as Datalog rules; the logic program transformation, which eliminates classical negation from the logic programming component; and the hMKNF $\neg$ transformation, which combines the transformed ontology and logic program and applies knowledge base doubling to preserve the coherency principle.

6.1. DL Transformation

The DL transformation converts a paraconsistent DL ontology into a classical DL ontology and subsequently rewrites it as a set of Datalog rules while preserving query answers. The transformation proceeds in three steps: normalization, paraconsistent transformation, and Datalog rewriting:

$$\mathcal{O}^P \xrightarrow{\text{Normalization}} \text{Norm}(\mathcal{O}^P) \xrightarrow{\text{Paraconsistent Transformation}} \hat{\mathcal{O}} \xrightarrow[\hat{\mathcal{O}} \in \mathcal{SR}\mathcal{OEL}]{\text{Datalog Rewriting}} DD(\hat{\mathcal{O}}).$$

The first step normalizes the paraconsistent DL ontology into a form suitable for the subsequent transformation. We then apply the paraconsistent transformation established in [14, 21] to obtain a consistent classical DL ontology. The main idea is to replace each classically negated concept or role with a fresh symbol, so that positive and negative information are represented separately. The resulting classical DL ontology $\hat{\mathcal{O}}$ preserves the consequences of the original paraconsistent ontology under the transformation. For example, the axiom $\text{ClosedAirspace} \sqsubseteq \neg \text{OpenAirspace}$ is transformed by replacing $\neg \text{OpenAirspace}$ with the fresh concept NOpenAirspace , yielding $\text{ClosedAirspace} \sqsubseteq \text{NOpenAirspace}$.

Finally, $\hat{\mathcal{O}}$ is rewritten as the Datalog program $DD(\hat{\mathcal{O}})$. In this work, we focus on $\mathcal{SR}\mathcal{OEL}$ and use the Datalog rewriting algorithm proposed in [22]. This step represents the transformed ontology as Datalog rules, allowing its reasoning to be carried out at the logic programming level.

6.2. Logic Program Transformation

The logic program transformation follows the same principle as the paraconsistent transformation of the DL ontology. Each classically negated atom $\neg A$ is replaced by a fresh predicate N_A , while positive atoms remain unchanged. For example, $\text{K}\neg \text{SafeToCross}(AS)$ is transformed into $\text{KN}_{\text{SafeToCross}}(AS)$. The resulting program $\hat{\mathcal{P}}$ therefore contains no explicit occurrences of classical negation.

6.3. hMKNF $\neg$ Transformation

This section combines the transformations of the DL ontology and the logic program to obtain a single logic program representing the paraconsistent hMKNF $\neg$ knowledge base. Let $\hat{\mathcal{K}} = (\hat{\mathcal{O}}, \hat{\mathcal{P}})$ denote the transformed knowledge base. The Datalog rewriting of $\hat{\mathcal{O}}$ is combined with the transformed logic program $\hat{\mathcal{P}}$ as follows: $DD(\hat{\mathcal{K}}) = \text{KDD}(\hat{\mathcal{O}}) \cup \hat{\mathcal{P}}$. At this stage, both the DL ontology and the logic program are represented uniformly as rules and facts.

Example 3 (Example 2 continued). We now apply the DL transformation to the paraconsistent ontology $\mathcal{O}^P$ from Example 2. After normalization and paraconsistent transformation, the resulting classical DL ontology $\hat{\mathcal{O}}$ is rewritten into Datalog. Since $\hat{\mathcal{O}}$ belongs to $\mathcal{SR}\mathcal{OEL}$, we apply the $\mathcal{SR}\mathcal{OEL}$ Datalog rewriting procedure from [22]. The resulting ontology rules are:

$$\begin{aligned} \text{KRestrictedAS}(X) &\leftarrow \text{KClosedAS}(X). \\ \text{KclosedBy}(X, d_{\text{closedBy}, \text{ClosureNOTAM}}) &\leftarrow \text{KClosedAS}(X). \end{aligned}$$

$$\begin{aligned}
\mathbf{KClosureNOTAM}(d_{\text{closedBy}, \text{ClosureNOTAM}}) &\leftarrow \mathbf{KClosedAS}(X). \\
\mathbf{KClosedAS}(X) &\leftarrow \mathbf{KRestrictedAS}(X), \mathbf{KclosedBy}(X, Y), \\
&\quad \mathbf{KClosureNOTAM}(Y). \\
\mathbf{KNOpenAS}(X) &\leftarrow \mathbf{KClosedAS}(X). \\
\mathbf{KClosedAS}(X) &\leftarrow \mathbf{KNOpenAS}(X). \\
\mathbf{KNClosedAS}(X) &\leftarrow \mathbf{KOpenAS}(X). \\
\mathbf{KOpenAS}(X) &\leftarrow \mathbf{KNClosedAS}(X). \\
\mathbf{KOpenAS}(X) &\leftarrow \mathbf{KEmergencyAS}(X). \\
\mathbf{KRunway}(X) &\leftarrow \mathbf{KOpenRunway}(X).
\end{aligned}$$

Here, $d_{\text{closedBy}, \text{ClosureNOTAM}}$ is the fresh auxiliary individual introduced for the existential restriction $\exists \text{closedBy}. \text{ClosureNOTAM}$.

The ABox assertions are directly represented as facts in $DD(\hat{\mathcal{O}})$. The logic programming component is transformed separately as described above, yielding $\hat{\mathcal{P}}$. The rewritten ontology and the transformed logic program are then combined into the single logic program $DD(\hat{\mathcal{K}}) = \mathbf{KDD}(\hat{\mathcal{O}}) \cup \hat{\mathcal{P}}$.

However, replacing a classically negated atom $\neg A$ with a fresh predicate N_A removes the direct relationship between classical and default negation. In particular, deriving N_A does not automatically imply **not** A , and deriving A does not automatically imply **not** N_A . To restore the coherency principle, we apply the knowledge base doubling transformation of [15] to $DD(\hat{\mathcal{K}})$.

The main idea is to introduce, for each predicate A , a duplicated predicate A^d . For a rule of the form

$$\mathbf{KH} \leftarrow \mathbf{KA}_1, \dots, \mathbf{KA}_n, \mathbf{not} B_1, \dots, \mathbf{not} B_m, \quad (18)$$

the doubling transformation produces the following two rules:

$$\mathbf{KH} \leftarrow \mathbf{KA}_1, \dots, \mathbf{KA}_n, \mathbf{not} B_1^d, \dots, \mathbf{not} B_m^d, \quad (19)$$

$$\mathbf{KH}^d \leftarrow \mathbf{KA}_1^d, \dots, \mathbf{KA}_n^d, \mathbf{not} B_1, \dots, \mathbf{not} B_m, \mathbf{not} \bar{H}, \quad (20)$$

where $\bar{H}$ denotes the complementary literal of H . In particular, if $H = A$, then $\bar{H} = N_A$, and if $H = N_A$, then $\bar{H} = A$.

The additional condition **not** $\bar{H}$ in the duplicated rule enforces the coherency principle. Moreover, default negation in the original rule is evaluated with respect to the duplicated predicate. Thus, the truth of **not** A depends on the failure to derive A^d , rather than on the failure to derive A directly.

The resulting doubled program, denoted by $DD(\hat{\mathcal{K}})^d$, provides the final logic programming representation used for query answering.

Example 4. Consider the paraconsistent $\text{hMKNF}\neg$ knowledge base from Example 2. After applying the transformations described above, the knowledge base is represented by the doubled logic program $DD(\hat{\mathcal{K}})^d$. A representative part of the resulting program is shown below:

$$\text{RestrictedAS}(X) \leftarrow \text{ClosedAS}(X), \quad (21)$$

$$\text{RestrictedAS}^d(X) \leftarrow \text{ClosedAS}^d(X), \mathbf{not}, N\text{RestrictedAS}(X), \quad (22)$$

$$N\text{OpenAS}(X) \leftarrow \text{ClosedAS}(X), \quad (23)$$

$$N\text{OpenAS}^d(X) \leftarrow \text{ClosedAS}^d(X), \mathbf{not}, \text{OpenAS}(X), \quad (24)$$

$$N\text{SafeToCross}(A) \leftarrow \text{ClosedAS}(A), \mathbf{not}, \text{SafeToCross}^d(A), \quad (25)$$

$$N\text{SafeToCross}^d(A) \leftarrow \text{ClosedAS}^d(A), \mathbf{not}, \text{SafeToCross}(A), \mathbf{not}, N\text{SafeToCross}(A). \quad (26)$$

For example, the fact $\text{RestrictedAS}(a)$ and its doubled version are represented as

$$\text{RestrictedAS}(a), \quad \text{RestrictedAS}^d(a) \leftarrow \mathbf{not}, N\text{RestrictedAS}(a). \quad (27)$$

Here, A denotes the original version of an atom and A^d its doubled version. The remaining ontology rules, logic programming rules, and facts are transformed in the same way.

7. Query Answering

Query answering is performed over the resulting doubled logic program $DD(\widehat{\mathcal{K}})^d$. We consider two evaluation strategies: a bottom-up approach that computes the *paraconsistent well-founded partition* of the program, and a top-down approach based on SLG resolution that evaluates queries directly.

7.1. Bottom-up Computation

The bottom-up approach computes the paraconsistent well-founded partition of the transformed knowledge base. Let $DD(\widehat{\mathcal{K}})_G^d$ denote the ground doubled logic program obtained from $DD(\widehat{\mathcal{K}})^d$. The paraconsistent well-founded partition is computed using an adaptation of the standard well-founded operator [17]. Given a partial partition (T, F) , where T and F are sets of ground atoms of $DD(\widehat{\mathcal{K}})_G^d$, the immediate-consequence operator computes the set of ground atoms that can be derived from $DD(\widehat{\mathcal{K}})_G^d$ with respect to (T, F) . It is defined as follows:

$$T_{DD(\widehat{\mathcal{K}})_G^d}^F(T) = \{h(r) | r \in DD(\widehat{\mathcal{K}})_G^d, b^+(r) \subseteq T, \mathbf{K}(b^-(r)) \subseteq F\}. \quad (28)$$

Thus, the head of a rule is derived when all atoms in its positive body are in T and all atoms occurring under default negation are in F .

The unfounded-set operator identifies atoms that cannot be supported by any applicable rule. A set UF of ground atoms is unfounded with respect to (T, F) if, for every $\mathbf{K}H \in UF$ and every rule $r \in DD(\widehat{\mathcal{K}})_G^d$ such that $h(r) = \mathbf{K}H$, at least one of the following conditions holds:

$$\mathbf{K}(b^-(r)) \cap T \neq \emptyset, b^+(r) \cap F \neq \emptyset, b^+(r) \cap UF \neq \emptyset. \quad (29)$$

These conditions state that a rule is blocked because an atom occurring under default negation is true, a positive body atom is false, or the rule depends positively on an atom that is itself unfounded. The greatest such unfounded set is denoted by $F_{DD(\widehat{\mathcal{K}})_G^d}(T, F)$.

Accordingly, the well-founded operator for the doubled ground logic program is

$$W_{DD(\widehat{\mathcal{K}})_G^d}^p(T, F) = (T_{DD(\widehat{\mathcal{K}})_G^d}^F(T), F_{DD(\widehat{\mathcal{K}})_G^d}(T, F)). \quad (30)$$

The operator is applied iteratively, starting from $(T_0, F_0) = (\emptyset, \emptyset)$ and, for every $i \geq 0$,

$$(T_{i+1}, F_{i+1}) = W_{DD(\widehat{\mathcal{K}})_G^d}^p(T_i, F_i). \quad (31)$$

Since $DD(\widehat{\mathcal{K}})_G^d$ is finite and the well-founded operator is monotonic, the computation reaches its least fixpoint after finitely many iterations. The resulting paraconsistent well-founded partition is denoted by $(T_W^p, F_W^p) = W_{DD(\widehat{\mathcal{K}})_G^d}^p \uparrow \omega$.

The status of each ground atom A is then determined from the paraconsistent well-founded partition by considering A together with its duplicated counterpart A^d :

1. If $A \in T_W^p$ and $A^d \notin F_W^p$, then A is *consistently true*.
2. If $A \notin T_W^p$ and $A^d \in F_W^p$, then A is *consistently false*.
3. If $A \in T_W^p$ and $A^d \in F_W^p$, then A is *contradictory*.
4. If $A \notin T_W^p$ and $A^d \notin F_W^p$, then A is *undefined*.

These conditions allow the bottom-up approach to distinguish consistently true, consistently false, contradictory, and undefined query answers.

7.2. Top-Down Computation.

Top-down evaluation is based on SLG(O) resolution [17]. After Datalog rewriting and knowledge base doubling, ontology reasoning is compiled into logic programming rules, reducing query answering to standard SLG evaluation [23]. Unlike bottom-up evaluation, top-down evaluation derives only the information required to answer the query. Furthermore, tabling guarantees termination and avoids redundant derivations. To classify the answers obtained by SLG resolution, a conjunctive query of the form:

$$Q \leftarrow A_1, \dots, A_n, \text{ not } B_1, \dots, \text{ not } B_m$$

is transformed into two distinct queries similar to knowledge base doubling which is illustrated as follows:

$$q(X) \leftarrow A_1, A_2, \dots, A_n, \text{ not } B_1^d, \text{ not } B_2^d, \dots, \text{ not } B_m^d. \quad (32)$$

$$q^d(X) \leftarrow A_1^d, A_2^d, \dots, A_n^d, \text{ not } B_1, \text{ not } B_2, \dots, \text{ not } B_m. \quad (33)$$

The results of $q(X)$ and $q^d(X)$ are then used to classify each query answer as below:

1. A query answer is classified as *consistently true* when $q(X)$ succeeds and $q^d(X)$ is not false. The consistent answers can be obtained as follows: $q(X) \wedge q^d(X)$ or $q(X) \wedge \text{undef}(q^d(X))$, where $\text{undef}(q^d(X))$ denotes that $q^d(X)$ is undefined in the well-founded model.
2. A query answer is classified as *contradictory* if $q(X)$ succeeds and $q^d(X)$ fails: $q(X) \wedge \text{not } q^d(X)$
3. A query answer is classified as *undefined* when both $q(X)$ and $q^d(X)$ are undefined in the well-founded model; operationally, this corresponds to answers satisfying: $\text{undef}(q(X)) \wedge \text{undef}(q^d(X))$.

8. Implementation

This section describes the implementation of the proposed query answering system for paraconsistent hMKNF $\neg$ knowledge bases. The current implementation supports $\mathcal{SROEL}$ ontologies represented in OWL/XML and logic programming rules written in Prolog syntax, where classical negation is represented using the prefix -. It provides both bottom-up and top-down query answering. Figure 2 illustrates the system architecture, which is organized into four main steps:

1. *Preprocessing Step*: The system first parses the DL ontology and the logic programming component of the input knowledge base. The ontology is then normalized using structural transformation [24] to obtain a form suitable for the subsequent transformations.
2. *Paraconsistent Transformation Step*: The normalized paraconsistent DL ontology is transformed into a classical DL ontology using the transformation of [21, 10]. Classical negation in the logic programming component is similarly replaced by fresh predicate symbols.
3. *Program Transformation Step*: The resulting $\mathcal{SROEL}$ ontology is rewritten as Datalog rules following [22] and combined with the transformed logic program. Knowledge base doubling is then applied to preserve the coherency principle. For each predicate A , two distinct predicates are introduced: aA , representing its original version, and dA , representing its doubled version. Similarly, a classically negated predicate $\neg A$ is represented by anA and dnA , corresponding to its original and doubled versions, respectively. The resulting rules form a single logic program used for query answering.
4. *Query Answering Step*: Query answering is performed using either the bottom-up or the top-down approach. In the bottom-up approach, the paraconsistent well-founded partition of the transformed program is computed and stored in a database, and queries are evaluated over the stored data. In the top-down approach, queries are evaluated directly over the transformed program using SLG resolution. In both approaches, the query answers are classified as *consistently true*, *consistently false*, *contradictory*, or *undefined*.

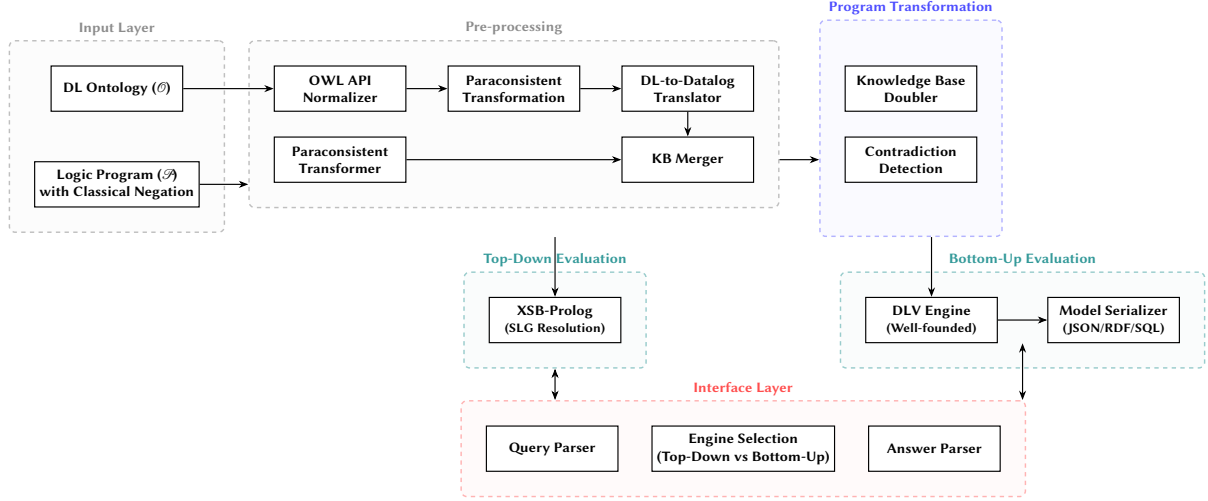


Figure 2: Architecture for $hMKNF_{\neg}$ Query Answering System

Bottom-up Engine. The bottom-up engine is implemented using DLV [19]. The ground doubled logic program $DD(\hat{\mathcal{K}}_G^d)$ generated during the program transformation phase is provided as input to DLV, which computes its well-founded model. DLV returns the atoms that are true and undefined, while the remaining ground atoms are considered false. These results are used to construct the paraconsistent well-founded partition (T_W^p, F_W^p) .

The computed results are then classified as *consistently true*, *consistently false*, *contradictory*, or *undefined* according to the conditions described in Section 7.1. The classified results are stored in an SQLite database, which is subsequently used to answer queries.

Example 5. Consider the doubled program obtained from Example 4. The ground program is provided as input to the DLV engine, which computes its well-founded model. A representative part of the DLV output is shown below:

Listing 1: Partial output of the DLV engine

```

True: {
aemergencyas(a), acrosses(f,a), apriorityflight(f),
arestrictedas(a), aclosedas(a), drestrictedas(a),
...
anclosedas(a), anopenas(a), aopenas(a),
ansafetocross(a), acontinue(f),
arunway(rw1), drunway(rw1),
aopenrunway(rw1), dopenrunway(rw1)
}

Undefined: {
dland(f), adivert(f), aland(f), ddivert(f),
areroute(f,a), dcontinue(f), adelayed(f),
...
}

```

The DLV engine returns the atoms that are true and undefined in the well-founded model, while the remaining ground atoms are false. The model parser uses the original and doubled versions of each atom to apply the classification conditions described in Section 7.1. The classified results are then materialized in the database. A representative part of the stored result is shown in Table 1, where $\vdots$ denotes omitted rows.

Top-down Engine. The top-down engine is implemented using XSB-Prolog, which provides an implementation of SLG resolution [20]. When a query is submitted, the query doubling transformation described in Equations (32) and (33) generates its original and doubled forms. Both forms are evaluated

Atom	Argument 1	Argument 2	Status
<i>restrictedas</i>	a	–	Consistently True
<i>closedas</i>	a	–	Contradictory
<i>openas</i>	a	–	Contradictory
<i>nsafetocross</i>	a	–	Consistently True
⋮	⋮	⋮	⋮
<i>reroute</i>	f	a	Undefined
<i>continue</i>	f	–	Consistently True
<i>delayed</i>	f	–	Undefined
<i>openrunway</i>	rw1	–	Consistently True

Table 1

Partial materialization of the computed query answering results.

independently using SLG resolution, and their results are classified according to the conditions described in Section 7.2 as *consistently true*, *consistently false*, *contradictory*, or *undefined*.

8.1. Use Case Implementation

The use case described in Section 2 was implemented as an hMKNF $\neg$ knowledge base and evaluated using the proposed query answering system. The evaluation of the pilot assistant use case is based on the knowledge bases summarized in Table 2 and the queries presented in Table 3. A partial performance evaluation is provided based on query execution time, with the aim of comparing the performance characteristics of the bottom-up and top-down query answering approaches.

Dataset	TBox axioms	RBox axioms	ABox axioms	Logic programs	Total
PA1	148	243	8,130	70	8,591
PA2	148	243	16,130	70	16,591
PA3	148	243	32,130	70	32,591
PA4	148	243	48,130	70	48,591
PA5	148	243	96,130	70	96,591
PA6	148	243	128,130	70	128,591
PA7	148	243	144,130	70	144,591
PA8	148	243	160,130	70	160,591

Table 2

Statistics of the pilot assistant knowledge bases used for evaluation.

Query	Purpose
Q1	Retrieve the hazardous airspaces intersecting a flight plan.
Q2	Retrieve suitable diversion airports for a mission and flight plan.
Q3	Determine whether a flight plan is free from hazards and operational constraints.
Q4	Determine whether a mission is feasible for the associated flight plan.
Q5	Determine whether a mission is subject to arrival risks.
Q6	Retrieve the NOTAMs that affect a flight plan.

Table 3

Evaluated queries.

Preprocessing. The preprocessing results are summarized in Figure 3 and Table 4.

Top-down Reasoning. The performance results for top-down query answering are presented in Table 5. Queries Q1–Q5 are answered efficiently across all pilot-assistant datasets. The execution times of Q4 and Q5 remain nearly constant as the dataset size increases, whereas those of Q1, Q2, and Q3

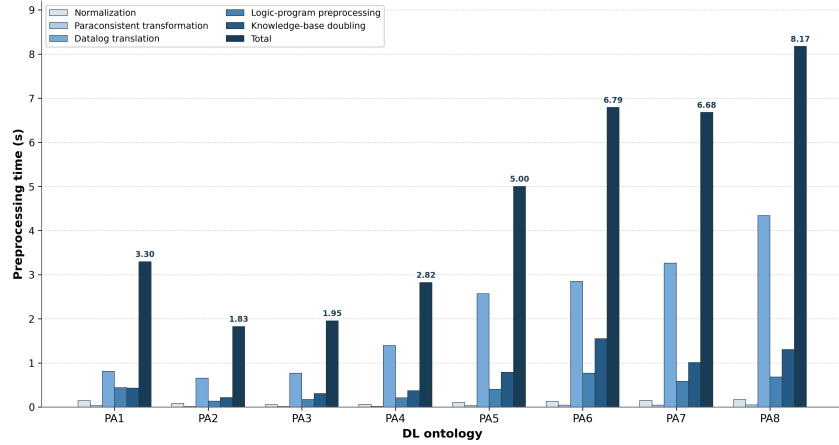


Figure 3: Preprocessing statistics for the pilot assistant ontologies

Ontology	Violated Axioms	Generated Rules	Memory (MB)	CPU (%)
PA1	38	8,665	264.75	100.00
PA2	38	16,665	368.17	55.56
PA3	38	32,665	584.29	39.17
PA4	38	48,665	1,258.59	32.48
PA5	38	96,665	1,231.78	53.50
PA6	38	128,665	2,845.06	38.33
PA7	38	144,665	1,498.08	44.30
PA8	38	160,665	2,711.92	74.03

Table 4

Preprocessing statistics for the pilot assistant.

increase gradually. Query Q6 is more computationally expensive because it retrieves all NOTAMs affecting the flight plans, and its execution time increases more noticeably as the number of NOTAMs grows.

Doubled logic program	Q1	Q2	Q3	Q4	Q5	Q6
PA1	0.3293	0.4402	0.4398	0.3268	0.3321	0.3272
PA2	0.4402	0.4446	0.5591	0.3340	0.3331	0.4446
PA3	0.7893	0.6592	0.8784	0.3242	0.3337	0.8963
PA4	1.2488	0.7718	1.1286	0.3219	0.3330	1.4682
PA5	4.0795	1.3378	2.0310	0.3167	0.3430	5.1163
PA6	6.6926	1.7112	2.6012	0.3300	0.3356	8.6422
PA7	8.1750	1.9268	2.8255	0.3160	0.3260	10.1602
PA8	10.0419	2.2477	3.0066	0.3324	0.4406	13.5691

Table 5

Top-down query answering time in seconds for Q1–Q6.

Bottom-up Reasoning. The bottom-up results are presented in Figure 4 and Table 6. Although computing the paraconsistent well-founded partition requires additional processing time (approximately 18 seconds for PA8), query answering is efficient once the partition is stored. However, updates require

recomputing the partition, which may limit the efficiency of this approach in dynamic aeronautical use cases where knowledge changes frequently.

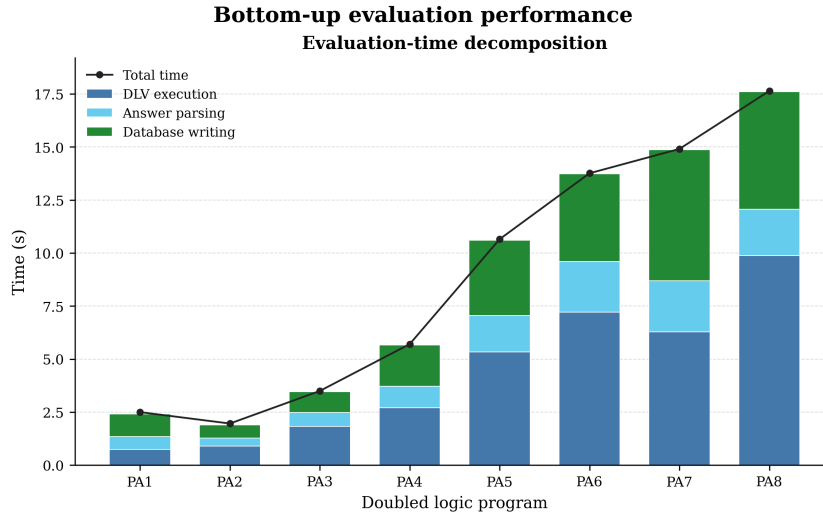


Figure 4: Bottom-up evaluation performance for the doubled PA logic programs. The total evaluation time includes DLV execution, answer parsing, and database writing.

Doubled logic program	Q1	Q2	Q3	Q4	Q5	Q6
PA1	0.0292	0.0181	0.0063	0.0006	0.0085	0.0073
PA2	0.0145	0.0185	0.0167	0.0010	0.0188	0.0335
PA3	0.0476	0.0318	0.0474	0.0008	0.0566	0.0693
PA4	0.0912	0.0898	0.0651	0.0040	0.0701	0.0726
PA5	0.1562	0.1845	0.1811	0.0023	0.1678	0.1446
PA6	0.1817	0.1809	0.1739	0.0007	0.1960	0.2201
PA7	0.2161	0.1472	0.1135	0.0005	0.1202	0.1847
PA8	0.2051	0.2265	0.2248	0.0013	0.2971	0.2701

Table 6

SQL query answering time in seconds for Q1–Q6 over the SQLite databases generated from the partition computed from the doubled PA logic programs.

9. Discussion

The proposed approach enables query answering over inconsistent hMKNF^- knowledge bases by combining paraconsistent DLs with logic programming rules extended with classical negation. This is particularly relevant in aeronautics, where information obtained from multiple sources may be incomplete, delayed, or contradictory. An important advantage is that contradictions remain localized, allowing query answering to continue over the unaffected parts of the knowledge base. To support this query answering process in practice, two complementary implementations were investigated: a bottom-up approach and a top-down approach.

The experimental evaluation highlights different performance characteristics of these two approaches. The bottom-up approach provides efficient query answering once the well-founded partition has been computed and stored in the database, but requires more memory and recomputation after knowledge base updates. In contrast, the top-down approach generally requires less memory and supports incremental

updates, which is beneficial for dynamic knowledge bases. This behavior is also observed in the pilot-assistant evaluation, where the top-down approach provides acceptable performance for moderate-sized knowledge bases. However, further evaluation with larger knowledge bases and a broader range of queries is required before drawing definitive conclusions about their relative performance.

The proposed system can also be compared with existing query answering systems for hybrid knowledge bases. Two prominent systems are NoHR, which is based on Hybrid MKNF [25], and DReW, which is based on DL-programs [26, 27]. The proposed system follows a workflow similar to NoHR but extends its query answering capabilities with paraconsistent query answering and support for classical negation in the logic programming component. Unlike NoHR, which requires a consistent DL ontology, the proposed system supports query answering in the presence of contradictions. Both systems are restricted to Datalog-rewritable DLs and use XSB-Prolog as the rule engine.

DReW follows a different approach to combining DL ontologies and logic programs. In DL-programs, the ontology and rule base remain separate components and interact through specialized operators that explicitly exchange information between them. In contrast, the proposed approach transforms the knowledge base into a logic program for query answering. Nevertheless, DReW provides an efficient implementation of DL-programs for Datalog-rewritable DLs and uses DLV to perform query answering under either answer-set or well-founded semantics [26, 27].

10. Conclusion and Future Work

In this paper, we presented a query answering approach for $\text{hMKNF}\neg$ knowledge bases under paraconsistent semantics. We define a paraconsistent counterpart for $\text{hMKNF}\neg$ based on [13] and a transformation pipeline that converts inconsistent $\text{hMKNF}\neg$ knowledge bases into a consistent form while preserving their consequences. By relying on Datalog rewriting and knowledge base transformations, the approach allows efficient reasoning using both bottom-up and top-down strategies.

As future work, we plan to extend the approach to more expressive DLs and to investigate real-time query answering for dynamic knowledge bases. We also plan to explore the implementation of additional plugins based on meta-programming that can be used to support various features, such as explanation generation for query answering and the identification of the root causes of contradictory atoms. In addition, we aim to perform a comprehensive comparison of our approach with existing systems, such as the NoHR reasoner, which is based on Hybrid MKNF, and the DReW system, which is based on DL-programs, evaluating both their performance and reasoning capabilities.

Declaration on Generative AI

During the preparation of this work, the author(s) used ChatGPT (OpenAI) for grammar and spelling checking, paraphrasing and rewording, and improving writing style. After using this tool, the author(s) reviewed and edited the content as needed and take full responsibility for the content of the publication.

References

- [1] V. Lifschitz, Nonmonotonic databases and epistemic queries, in: Proceedings of the 12th International Joint Conference on Artificial Intelligence. Sydney, 1991, pp. 381–386. URL: <http://ijcai.org/Proceedings/91-1/Papers/059.pdf>.
- [2] B. Motik, R. Rosati, Reconciling description logics and rules, *J. ACM* 57 (2010) 30:1–30:62. URL: <https://doi.org/10.1145/1754399.1754403>. doi:10.1145/1754399.1754403.
- [3] M. Knorr, J. J. Alferes, P. Hitzler, Local closed world reasoning with description logics under the well-founded semantics, *Artif. Intell.* 175 (2011) 1528–1554. URL: <https://doi.org/10.1016/j.artint.2011.01.007>. doi:10.1016/J.ARTINT.2011.01.007.
- [4] A. Raveendran Nair Sheela, C. Rey, F. De Grancey, Hybrid mknf with classical negation in the rule component, in: W. Faber, L. Giordano, R. Rocha, V. Santos Costa (Eds.), Proceedings 42nd International Conference on Logic Programming, Lisbon, Portugal, 20-23rd July 2026, volume 450

- of *Electronic Proceedings in Theoretical Computer Science*, Open Publishing Association, 2026, pp. 374–402. doi:10.4204/EPTCS.450.28.
- [5] N. A. Raveendran Sheela, F. De Grancey, C. Rey, V. Charpenay, Hybrid mknf for aeronautics applications: Usage and heuristics, in: 41st ICLP, Rende, Italy, 2025, volume 439 of *Electronic Proceedings in Theoretical Computer Science*, 2026, pp. 349–366. doi:10.4204/EPTCS.439.24.
 - [6] P. Godfrey, J. Grant, J. Gryz, J. Minker, Integrity constraints: Semantics and applications, in: J. Chomicki, G. Saake (Eds.), *Logics for Databases and Information Systems (the book grew out of the Dagstuhl Seminar 9529: Role of Logics in Information Systems, 1995)*, Kluwer, 1998, pp. 265–306.
 - [7] L. Cruz-Filipe, M. Franz, A. Hakhverdyan, M. Ludovico, I. Nunes, P. Schneider-Kamp, Active integrity constraints: From theory to implementation, in: A. L. N. Fred, J. L. G. Dietz, D. Aveiro, K. Liu, J. Filipe (Eds.), *Knowledge Discovery, Knowledge Engineering and Knowledge Management - 7th International Joint Conference, IC3K 2015, Lisbon, Portugal, November 12–14, 2015, Revised Selected Papers*, volume 631 of *Communications in Computer and Information Science*, Springer, 2015, pp. 399–420. URL: https://doi.org/10.1007/978-3-319-52758-1_22. doi:10.1007/978-3-319-52758-1_22.
 - [8] G. Feuillade, A. Herzig, C. Rantsoudis, Knowledge base repair: From active integrity constraints to active tboxes, in: S. Borgwardt, T. Meyer (Eds.), *Proceedings of the 33rd International Workshop on Description Logics (DL 2020) co-located with the 17th International Conference on Principles of Knowledge Representation and Reasoning (KR 2020), Online Event [Rhodes, Greece], September 12th to 14th, 2020*, volume 2663 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2020. URL: <https://ceur-ws.org/Vol-2663/invited-1.pdf>.
 - [9] Y. Ma, P. Hitzler, Z. Lin, Algorithms for paraconsistent reasoning with OWL, in: E. Franconi, M. Kifer, W. May (Eds.), *The Semantic Web: Research and Applications, 4th European Semantic Web Conference, ESWC 2007, Innsbruck, Austria, June 3–7, 2007, Proceedings, Lecture Notes in Computer Science*, Springer, 2007, pp. 399–413. URL: https://doi.org/10.1007/978-3-540-72667-8_29. doi:10.1007/978-3-540-72667-8_29.
 - [10] Y. Ma, P. Hitzler, Z. Lin, Algorithms for paraconsistent reasoning with owl, in: *Extended Semantic Web Conference, 2007*. URL: <https://api.semanticscholar.org/CorpusID:14453180>.
 - [11] B. Motik, R. Rosati, A faithful integration of description logics with logic programming, in: *Proceedings of the 20th IJCAI, Hyderabad, India, January 6–12, 2007*, pp. 477–482. URL: <http://ijcai.org/Proceedings/07/Papers/075.pdf>.
 - [12] B. Motik, R. Rosati, Reconciling description logics and rules, *J. ACM* 57 (2010) 30:1–30:62. URL: <https://doi.org/10.1145/1754399.1754403>. doi:10.1145/1754399.1754403.
 - [13] T. Kaminski, M. Knorr, J. Leite, Well-founded paraconsistent semantics for hybrid theories composed of rules and ontologies, in: *JOWO@IJCAI, 2015*. URL: <https://api.semanticscholar.org/CorpusID:5829935>.
 - [14] F. Maier, Y. Ma, P. Hitzler, Paraconsistent OWL and related logics, *Semantic Web* 4 (2013) 395–427. URL: <https://doi.org/10.3233/SW-2012-0066>. doi:10.3233/SW-2012-0066.
 - [15] C. V. Damásio, L. M. Pereira, A paraconsistent semantics with contradiction support detection, in: J. Dix, U. Furbach, A. Nerode (Eds.), *Logic Programming and Nonmonotonic Reasoning, 4th International Conference, LPNMR'97, Dagstuhl Castle, Germany, July 28–31, 1997, Proceedings, Lecture Notes in Computer Science*, Springer, 1997, pp. 224–243. URL: https://doi.org/10.1007/3-540-63255-7_18. doi:10.1007/3-540-63255-7_18.
 - [16] J. Ji, F. Liu, J.-H. You, Well-founded operators for normal hybrid mknf knowledge bases, 2017. URL: <https://arxiv.org/abs/1707.01959>. arXiv:1707.01959.
 - [17] J. J. Alferes, M. Knorr, T. Swift, Query-driven procedures for hybrid MKNF knowledge bases, *ACM Trans. Comput. Log.* 14 (2013) 16:1–16:43. URL: <https://doi.org/10.1145/2480759.2480768>. doi:10.1145/2480759.2480768.
 - [18] A. V. Gelder, The alternating fixpoint of logic programs with negation, *Proceedings of the eighth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems (1989)*. URL: <https://api.semanticscholar.org/CorpusID:249435543>.

- [19] M. Alviano, W. Faber, N. Leone, Well-founded semantics for logic programs with aggregates: Implementation and experimentation, in: RCRA@CPAIOR, 2010. URL: <https://api.semanticscholar.org/CorpusID:794775>.
- [20] T. Swift, D. S. Warren, XSB: extending prolog with tabled logic programming, *Theory Pract. Log. Program.* 12 (2012) 157–187. URL: <https://doi.org/10.1017/S1471068411000500>. doi:10.1017/S1471068411000500.
- [21] Y. Ma, P. Hitzler, Z. Lin, Paraconsistent reasoning for expressive and tractable description logics, in: F. Baader, C. Lutz, B. Motik (Eds.), *Proceedings of the 21st International Workshop on Description Logics (DL2008)*, Dresden, Germany, May 13–16, 2008, CEUR Workshop Proceedings, CEUR-WS.org, 2008. URL: <https://ceur-ws.org/Vol-353/MaHitzlerLin.pdf>.
- [22] M. Krötzsch, S. Rudolph, P. Hitzler, Description logic rules, in: *European Conference on Artificial Intelligence*, 2010. URL: <https://api.semanticscholar.org/CorpusID:267945043>.
- [23] W. Chen, D. S. Warren, Query evaluation under the well founded semantics, in: C. Beeri (Ed.), *Proceedings of the Twelfth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, May 25–28, 1993, Washington, DC, USA, ACM Press, 1993, pp. 168–179. URL: <https://doi.org/10.1145/153850.153865>. doi:10.1145/153850.153865.
- [24] Y. Kazakov, Consequence-driven reasoning for horn SHIQ ontologies, in: B. C. Grau, I. Horrocks, B. Motik, U. Sattler (Eds.), *Proceedings of the 22nd International Workshop on Description Logics (DL 2009)*, Oxford, UK, July 27–30, 2009, CEUR Workshop Proceedings, CEUR-WS.org, 2009. URL: https://ceur-ws.org/Vol-477/paper_69.pdf.
- [25] V. Ivanov, M. Knorr, J. Leite, A query tool for *EL* with non-monotonic rules, in: H. Alani, L. Kagal, A. Fokoue, P. Groth, C. Biemann, J. X. Parreira, L. Aroyo, N. F. Noy, C. Welty, K. Janowicz (Eds.), *The Semantic Web - ISWC 2013 - 12th International Semantic Web Conference*, Sydney, NSW, Australia, October 21–25, 2013, *Proceedings, Part I*, volume 8218 of *Lecture Notes in Computer Science*, Springer, 2013, pp. 216–231. URL: https://doi.org/10.1007/978-3-642-41335-3_14. doi:10.1007/978-3-642-41335-3_14.
- [26] G. Xiao, T. Eiter, S. Heymans, The drew system for nonmonotonic dl-programs, in: J. Li, G. Qi, D. Zhao, W. Nejdl, H. Zheng (Eds.), *Semantic Web and Web Science - 6th Chinese Semantic Web Symposium and 1st Chinese Web Science Conference, CSWS 2012*, Shenzhen, China, November 28–30, 2012, Springer, 2012, pp. 383–390. URL: https://doi.org/10.1007/978-1-4614-6880-6_33. doi:10.1007/978-1-4614-6880-6_33.
- [27] 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. URL: <https://www.sciencedirect.com/science/article/pii/S0004370208000520>. doi:<https://doi.org/10.1016/j.artint.2008.04.002>.