1 Introduction

Description Logics (DLs) are a family of formal logics designed for representing and inferring new knowledge from defined knowledge encapsulated in ontologies. However, some ontologies like SNOMED or LOINC [2] contain a vast number of axioms, making it challenging to extend them without generating unwanted inferences. Proofs, which are essentially trees consisting of axioms as vertices connected by direct consequence operators, can be useful in understanding the cause of these unwanted inferences. Recently, a new Java library called EVEE-LIBS [1] has been developed to generate proofs for DLs up to $\mathcal{ALCH}$. To achieve this, EVEE-LIBS uses Lethe [3], a consequence-based reasoning procedure, to generate new axioms. However, the underlying direct consequence operator, which is Lethes underlying calculus, plays a critical role in determining the resulting proofs. Therefore, in this student project report, I will present an implementation of the alternative calculus proposed in [3] and compare the generated proofs to those of EVEE-LIBS. This comparison will provide valuable insights into the effectiveness of different calculi in generating proofs for ontologies, which could ultimately lead to improved methods for extending and reasoning about ontologies.

2 The Description Logic $\mathcal{ALCH}$

- What is ALCH
- syntax and semantics

2.1 Syntax

The DL $\mathcal{ALCH}$ allows the operators concept negation ($\neg$), concept intersection ($\sqcap$), concept union ($\sqcup$), universal restriction ($\forall$), existential quantification ($\exists$), the top concept ($\top$), the bottom concept ($\bot$) and finally role hierarchies ($\sqsubseteq$). With those operators and two additional sets of concept names (N_C) and role names (N_R), one can then build all complex concepts of $\mathcal{ALCH}$ ($N_{\mathcal{ALCH}}$) as follows:

if $C \in N_C$	then $C \in N_{\mathcal{ALCH}}$
if $C \in N_{\mathcal{ALCH}}$	then $\neg C \in N_{\mathcal{ALCH}}$
if $C_1, C_2 \in N_{\mathcal{ALCH}}$	then $C_1 \sqcap C_2 \in N_{\mathcal{ALCH}}$
if $C_1, C_2 \in N_{\mathcal{ALCH}}$	then $C_1 \sqcup C_2 \in N_{\mathcal{ALCH}}$
if $C \in N_{\mathcal{ALCH}}$ and $r \in N_R$	then $\exists r.C \in N_{\mathcal{ALCH}}$
if $C \in N_{\mathcal{ALCH}}$ and $r \in N_R$	then $\forall r.C \in N_{\mathcal{ALCH}}$

Now that complex concepts are defined we can also formalize the notion of a TBox.

Definition 1 (TBox)**.** *A $\mathcal{ALCH}$ TBox $\mathcal{T}$ is a set of general concept inclusions and role inclusions i.e. $\mathcal{T} \subseteq \{C_1 \sqsubseteq C_2 \mid C_1, C_2 \in N_{\mathcal{ALCH}}\} \cup \{r_1 \sqsubseteq r_2 \mid r_1, r_2 \in N_R\}$*

Definition 2 (ABox)**.** *A ABox $\mathcal{A}$ is a set of concept and role assertions i.e.* $\mathcal{A} \subseteq \{a : C \mid a \in N_I, C \in N_{\mathcal{ALCH}}\} \cup \{(a, b) : r \mid a, b \in N_I, C \in N_{\mathcal{ALCH}}\}$

2.2 Semantics

The semantics of the DL $\mathcal{ALCH}$ are defined by a first order logic interpretation $I := (\Delta^I, \cdot^I)$, where Δ^I is the *domain* of the interpretation and $\cdot^I$ the interpretation function. Concepts are then interpreted as follows:

Definition 3 (Interpetation)**.** *Let $A \in N_C, C, D \in N_{\mathcal{ALCH}}, r \in N_R$ then*

$$
\begin{array}{lcl}
\top^I & := & \Delta^I \\
\bot^I & := & \emptyset \\
A^I & := & A' \text{ where } A' \subseteq \Delta^I \\
r^I & := & R \text{ where } R \subseteq \Delta^I \times \Delta^I \\
(\neg C)^I & := & C^I \\
(C \sqcap D)^I & := & C^I \cap D^I \\
(C \sqcup D)^I & := & C^I \cup D^I \\
(\exists r.C)^I & := & \{a \in \Delta^I \mid \exists b \in \Delta^I : (a, b) \in r^I \text{ and } b \in C^I\} \\
(\forall r.C)^I & := & \{a \in \Delta^I \mid \forall b \in \Delta^I : (a, b) \in r^I \text{ then } b \in C^I\}
\end{array}
$$

This interpretation can then also be extended to GCIs and role inclusions. Since a GCI or role inclusions only hold or not hold under a interpretation the holds operator ($\models$)is used to express this behaviour.

Definition 4. *For $C, D \in N_{\mathcal{ALCH}} : I \models C \sqsubseteq D :\iff C^I \subseteq D^I$*
For $r, s \in N_R : I \models r \sqsubseteq s :\iff (\forall a, b \in \Delta^I : (a, b) \in r^I \Rightarrow (a, b) \in s^I)$

3 The Algorithm

In this section I will describe how my Algorithm works and therefore start with what a proof formally is.

3.1 Proofs

A proof is a list of GCIs $A_1, \ldots, A_n$, where for each $i \in \{1, \ldots n-1\}$ $\{A_1, \ldots, A_i\} \models A_{i+1}$. Since my approach uses rules to derive such proofs I now define a rule based definition of a proof. A rule based proof is not a list of GCIs but rather a list of inferences, where a inference is a tuple $((\phi_1, \ldots, \phi_n), \psi, \mathbf{R})$, where $\phi_1, \ldots, \phi_n$, are GCIs and the list containing them is called premise, ψ is also a GCI and called the conclusion and $\mathbf{R}$ is a rule. In order a list of inferences to be a proof it has to hold that for all inferences the premise and conclusion actually have to be an instance of $\mathbf{R}$.

3.2 Normalization

Before we dive into the rule based procedure, we first need to be sure, that our ontology has the right form.

Definition 5 (Normalform)**.** *An ontology $\mathcal{O}$ is said to be in normalform iff all axioms in $\mathcal{O}$ are of the form $\bigsqcap A_i \sqsubseteq \bigsqcup B_j, A \sqsubseteq \exists r.B, \exists r.A \sqsubseteq A, A \sqsubseteq \forall r.B$ or $r \sqsubseteq s$, where A_i, B_j, A, B are concept names and r, s are role names.*

An $\mathcal{ALCH}$ ontology $\mathcal{O}$ can then be easily transformed into this normalform by first using the *structural transformation* $\mathrm{st} : N_C \to N_C$. st is defined as follows:

$$\begin{array}{rr} \mathrm{st}(A) \coloneqq A & \mathrm{st}(C \sqcap D) \coloneqq [C] \\ \mathrm{st}(\top) \coloneqq \top & \mathrm{st}(C \sqcup D) \coloneqq [C] \sqcup [D] \\ \mathrm{st}(\bot) \coloneqq \top & \mathrm{st}(\exists r.C) \coloneqq [C]\exists r.[C] \\ \mathrm{st}(\neg C) \coloneqq \neg[C] & \mathrm{st}(\exists r.C) \coloneqq [C]\exists r.[C] \end{array}$$

where for a concept C [C] is just a new atom that corresponds to C. Now we obtain a new structual transformed ontology $\mathrm{st}(\mathcal{O})$ form $\mathcal{O}$ by first adding all role inclusions or $\mathcal{O}$ into it and then adding

- $\mathrm{st}(C) \sqsubseteq [C]$ for every negative $C \in \mathcal{O}$
- $[D] \sqsubseteq \mathrm{st}(D)$ for every positive $D \in \mathcal{O}$
- $[C] \sqsubseteq [D]$ for every axiom $C \sqsubset D$ occuring $C \in \mathcal{O}$

$\mathrm{st}(\mathcal{O})$ needs then to be further normalized by removing all axioms of the form $\forall r.C \sqsubseteq A$ and adding $\neg\exists r.\neg C$. Note that both axioms are actually equivalent. Finally the following normalization rules have to be applied to $\mathrm{st}(\mathcal{O})$ until no rule can be applied anymore.

- $\dfrac{[C \sqcap D] \sqsubseteq [C] \sqcap [D]}{[C \sqcap D] \sqsubseteq [C], \quad [C \sqcap D] \sqsubseteq [D]}$
- $\dfrac{[C] \sqcup [D] \sqsubseteq [C \sqcap D]}{[C] \sqsubseteq [C \sqcup D], \quad [D] \sqsubseteq [C \sqcup D]}$
- $\dfrac{[\neg C] \sqsubseteq \neg[C]}{[\neg C] \sqcap [C] \sqsubseteq \bot}$
- $\dfrac{\neg[C] \sqsubseteq [\neg C]}{\top \sqsubseteq [\neg C] \sqcap [C]}$

3.3 Inference rules

Definition 6 (Inference Rules)**.**

On a normalized ontology we can now apply the following inference rules.

$$\mathbf{R}^{+}_{\mathbf{A}} \frac{}{H \sqsubseteq A} : A \in H \tag{1}$$

$$\mathbf{R}^{-}_{\mathbf{A}} \frac{H \sqsubseteq N \sqcup A}{H \sqsubseteq N} : \neg A \in H \tag{2}$$

$$\mathbf{R}^{\mathbf{n}}_{\sqcap} \frac{\{H \sqsubseteq N_i \sqcup A_i\}_{i=1}^{n}}{H \sqsubseteq \bigsqcup_{i=1}^{n} N_i \sqcup M} : \prod_{i=1}^{n} A_i \sqsubseteq M \in \mathcal{O} \tag{3}$$

$$\mathbf{R}^{+}_{\exists} \frac{H \sqsubseteq N \sqcup A}{H \sqsubseteq N \sqcup \exists r.B} : A \sqsubseteq \exists r.B \in \mathcal{O} \tag{4}$$

$$\mathbf{R}^{-}_{\exists} \frac{H \sqsubseteq M \sqcup \exists r.K, \quad K \sqsubseteq N \sqcup A}{H \sqsubseteq M \sqcup B \sqcup \exists r.(K \sqcap \neg A)} : \exists s.A \sqsubseteq B \in \mathcal{O} \quad r \sqsubseteq_{\mathcal{O}} s \tag{5}$$

$$\mathbf{R}^{\perp}_{\exists} \frac{H \sqsubseteq M \sqcup \exists r.K, \quad K \sqsubseteq \perp}{H \sqsubseteq M} \tag{6}$$

$$\mathbf{R}_{\forall} \frac{H \sqsubseteq M \sqcup \exists r.K, \quad H \sqsubseteq N \sqcup A}{H \sqsubseteq M \sqcup N \sqcup \exists r.(K \sqcap B)} : A \sqsubseteq \forall s.B \in \mathcal{O} \quad r \sqsubseteq_{\mathcal{O}} s \tag{7}$$

where M, N are disjunctions of atoms and H, K are conjunctions of literals.

Algorithm 1 Main Loop

```
procedure MAIN(ontology, goal)
    Normalizer.normalize(ontology)
    proofHandler ← new ProofHandler(query)
    rules ← getRules(ontology, proofHandler)
    while notFinished(proofHandler) do
        for (rule in rules) do
            setNewRule(rule.apply())
        end for
    end while
end procedure
```

The algorithms main loop consists of two stages: normalization and rule application. The normalization stage implements the algorithm as outlined in section 3.2. The normalized ontology and a new instance of a ProofHandler are then used to create instances of each rule type. Every rule, except for 1, 2 and 6 which do not need ontology axioms, need these two as arguments. The ontology is preprocessed, such that all relevant ontology axioms are saved in a way which allows for an easier rule application and prevents searching for relevant axioms every time the rule is applied. The ProofHandler provides active axioms from which rules are able to create new axioms and processes them, as described in 3.4, to generate new inferences. The created rules are then applied one after each other until the halting condition is met. The halting condition is set to true either when the goal concept is reached or when all rules are unable to generate a new axiom in an iteration.

3.4 Proof Handler

The ProofHandler is the main component of the algorithm. It is responsible for handling new produced axioms, provides the rules with these new axioms, checks whether the halting condition is met and creates a proof in case the algorithm finds one. The ProofHandler is initialized with the query axiom. The sub concept of this axiom is added to the list of active axioms and the super concept is saved in order to check whether the goal was found. The handling of newly produced axioms contains the following steps. First the ProofHandler is provided with all axioms which were used to generate, the new axiom the newly generated axiom and the name of the rule which calls the method. In case of rules which also use ontology axioms the provided axioms are split into premises and ontology premises. This information is then used to create a new Inference and add the new axiom to the list of active Axioms. Since some rules do generate some axioms multiple times the ProofHandler also checks whether the axiom is already in the list of active axioms and depending on the outcome of this check returns the information whether a new Inference was created. While doing so it is also checked whether the super concept of the new axiom is the goal axiom. In case the algorithm stops and the goal was found the ProofHandler also provides a method to create a proof. In order to prevent to just return all Inferences which were created during the algorithm the ProofHandler uses the Java library EVEE to create a meaningful proof. EVEE not only provides data structures like Inferences and Proofs but also has different algorithms to create different types of proofs. In this project I use the algorithm which creates a proof with the lowest depth possible.

3.5 Inference Rules

In this subsection I will describe how each inference rule is implemented and what data structures they use.

3.5.1 The rule $\mathbf{R_A^+}$

The rule 1 is one of the simplest rules, since it does not use any active axioms and also no axioms from the ontology. The only thing that has to be taken care of is creating new axioms from active concepts. In order to do that the implementation iterates through all active concepts and searches for conjunctions or atoms, which are seen as conjunctions of size 1. In case a conjunction is found all atoms within the conjunction are saved to a list. Each element is then used as super concept of a new axiom with its origin axiom as sub concept. After the handling of all active concepts the implementation empties the list of active concepts in order to reduce redundant axiom creation.

3.5.2 The rule $\mathbf{R_A^-}$

This rule also does not use any axioms from the ontology but does use active axioms. The implementation, first searches for axioms with a conjunction as

sub concept, which contain negative literals. In case such an axiom was found the implementation searches for a positive literal in the super concept of the axiom. If the super concept contains the positive literal the implementation creates a new axiom with the same sub concept as the origin axiom and the super concept of the origin axiom without the positive literal. Note that used active axioms are not removed from the list of active axioms, which allows the possibility of redundant axiom creation. In order to tackle this problem I created an alternative rule implementation which checks before handling an active axiom whether it was already used by this rule. To achieve this the rule stores all used axioms in a set. This set can become very large and in worst case can contain all active axioms. That's why I made it an alternative implementation in order to allow for a time optimized version of the algorithm and a space optimized version of the algorithm.

3.5.3 The rule $\mathbf{R}^{\mathbf{n}}_{\sqcap}$

The rule 3 is the first rule which uses axioms from the ontology. On creation of its implementation first the ontology gets filtered for axioms of the form $\bigsqcap_{i=1}^{n} A_i \sqsubseteq M$ These axioms get saved into a list of tuples where the first element of the tuple is the conjunction stored as a set and the second element is the entire axiom, which will not only be used to get its super concept for creation of a new axiom but also in order to pass it as ontology premise to the ProofHandler.

On application of the rule the following function $m \coloneqq N_{\mathcal{ALCH}} \times N_C \to \{N \mid N \subseteq N_C\}$ is defined by using the current active axioms. First all active axioms get filtered for axioms of the form $H \sqsubseteq N \sqcup A$, where H is conjunctions of literals N disjunctions of atoms and A an atom. Every axiom is then used to define the function m in the following way: For each active axiom $\alpha = H \sqsubseteq N \sqcup A : m(H, A) \coloneqq \{N_c | N_c \in N\}$. After the function is defined it will be used to compose derived axioms. In order to do so the implementation first iterates through all ontology axioms and checks, whether the LHS of a given axiom is subsumed by the right entry of the domain of m. If this is the case, m can be used to find for all A_i in the matched ontology axiom the corresponding N_i and in the end the new axiom $H \sqsubseteq \bigsqcup_{i=1}^{n} N_i \sqcup M$ is created.

When the rule is applied to the set of active axioms the rule first maps all axioms with an equal subconcept to a tuple, where the first entry is a list of concept names, which where these concept names corresponds to all A_i which can be found for a given subconcept H and the second entry is a map which maps a concept name which is a particular A_i and maps it to a tuple where its first entry is a list of all concept names in the corresponding N_i and the second entry is just the hole axiom $H \sqsubseteq N_i \sqcup A_i$.

After the map is created the rule starts to iterate through the ontology axioms and the created entries of the just created map and looks for entries in the map where the set of A_is from the list, which was created from the ontology, is subsumed by some set in the set in first entry of the map. When such a match was found we have all need elements to create the new axiom $H \sqsubseteq \bigsqcup_{i=1}^{n} N_i \sqcup M$ in order to get all requiered N_is the list of A_is of the ontology axiom is used to

filter the map in the second entry of the entry of the just created map.

3.5.4 The rule $\mathbf{R}_{\exists}^{+}$

The rule (4) is also uses axioms from the ontology so on creation of the rule implementation the ontology gets filtered for axioms of the form $A \sqsubseteq \exists r.B \in \mathcal{O}$. When the rule implementation is applied, the active axioms just get filtered for

3.5.5 The rule $\mathbf{R}_{\exists}^{-}$

The rule (5) not only uses axioms from the ontology but also uses role inclusions, which follow from the ontology. The role inclusions are calculated separately since rule (7) also uses them. So the only thing that has to be done on creation is, that axioms of the form $\exists s.A \sqsubseteq B$ need to be filtered. On rule application the rule implementation then first filters the active axioms for axioms where the superconcept is a disjunction of concept names and one of them matches some A in some ontology axiom. The matched axioms then are mapped to their subconcepts K. After this is done the just created map is used to find axioms of the form $H \sqsubseteq M \sqcup \exists r.K$

3.5.6 The rule $\mathbf{R}_{\exists}^{\perp}$

The rule (6) does not use no axioms from the original ontology. Usually there are way more axioms containing existential restrictions than axioms with the bottom concept on the right hand side and therefore the implementation first filters for axioms of the form $K \sqsubseteq \perp$ safes them in a list l. Afterwards the rule implementation filters for rules of the form $H \sqsubseteq M \sqcup \exists r.K$ and checks if a axiom of this form is found, if the K already occures in some LHS of an axiom in l. From this found rule then the new axiom $H \sqsubseteq M$ can then be created immediately.

3.5.7 The rule $\mathbf{R}_{\forall}$

The rule (7) is no the final rule now again uses from the ontology following role inclusions and also axioms from the ontology of the form $A \sqsubseteq \forall s.B$. Therefore, on creation of the rule implementation first the ontology gets filtered for such axioms, where the role s is on the right hand side of some implied role inclusion. On application of the rule implementation first searches for active axioms of the form $H \sqsubseteq M \sqcup \exists r.K$ where r has to match some RHS of some matched role inclusion. The found axioms are then used to define a function $m_l \coloneqq N_{\rightarrow}$

Example 1. *Let* $\mathcal{O} = \{B \sqsubseteq \exists r.D, A \sqsubseteq \forall s.E, \exists r.E \sqsubseteq C, r \sqsubseteq s\}$ *then a proof for* $A \sqcap B \sqsubseteq C$ *would look like the following:*
$((), A \sqcap B \sqsubseteq B, \mathbf{R}_{\mathbf{A}}^{+})$,
$((A \sqcap B \sqsubseteq B, B \sqsubseteq \exists r.D), A \sqcap B \sqsubseteq \exists r.D, \mathbf{R}_{\exists}^{+})$,
$((A \sqcap B \sqsubseteq \exists r.D, A \sqsubseteq \forall s.E, r \sqsubseteq s), A \sqcap B \sqsubseteq \exists r.(D \sqcap E), \mathbf{R}_{\forall})$,
$((), (D \sqcap E) \sqsubseteq E, \mathbf{R}_{\mathbf{A}}^{+})$,

$((A \sqcap B \sqsubseteq \exists r.(D \sqcap E), (D \sqcap E) \sqsubseteq E, \exists r.E \sqsubseteq C), A \sqcap B \sqsubseteq \exists r.(D \sqcap E \sqcap \neg E) \sqcup C, \mathbf{R}^{-}_{\exists})$,
$((), D \sqcap E \sqcap \neg E \sqsubseteq E, \mathbf{R}^{+}_{\mathbf{A}})$,
$((D \sqcap E \sqcap \neg E \sqsubseteq E), D \sqcap E \sqcap \neg E \sqsubseteq \bot, \mathbf{R}^{-}_{\mathbf{A}})$,
$((A \sqcap B \sqsubseteq \exists r.(D \sqcap E \sqcap \neg E) \sqcup C, D \sqcap E \sqcap \neg E \sqsubseteq \bot), A \sqcap B \sqsubseteq C, \mathbf{R}^{\bot}_{\exists}))$

Notes that in the algorithm not only these concepts would be created but also other rules. For example even in the first step not only $A \sqcap B \sqsubseteq B$ would have been created but also $A \sqcap B \sqsubseteq A$.

3.6 Problems

While implementing the algorithm I encountered some problems, which I will discuss in the following.

- no nested conjunctions

WHY DOES MY APPROACH WORK

- proofs definition
- calcül
- example
- pseudo code
- normalform
- normalization
- rule application
- probleme discutieren !!!!

4 Proof Generation

When comparing the proofs generated by the ALCH-Reasoner and the Lethe-Reasoner, one can see that the proofs differ quite substantially. While for example the ALCH-Reasoner uses only the $mathbf R^n_\sqcap$ rule on a normalized task 1 ontology, the Lethe-Reasoner only uses quite a few different types of elimination rules.

Nicht zwingened notwendig

- UML diagrams
- normalization
- inference rules
- loop
- proof building
- export

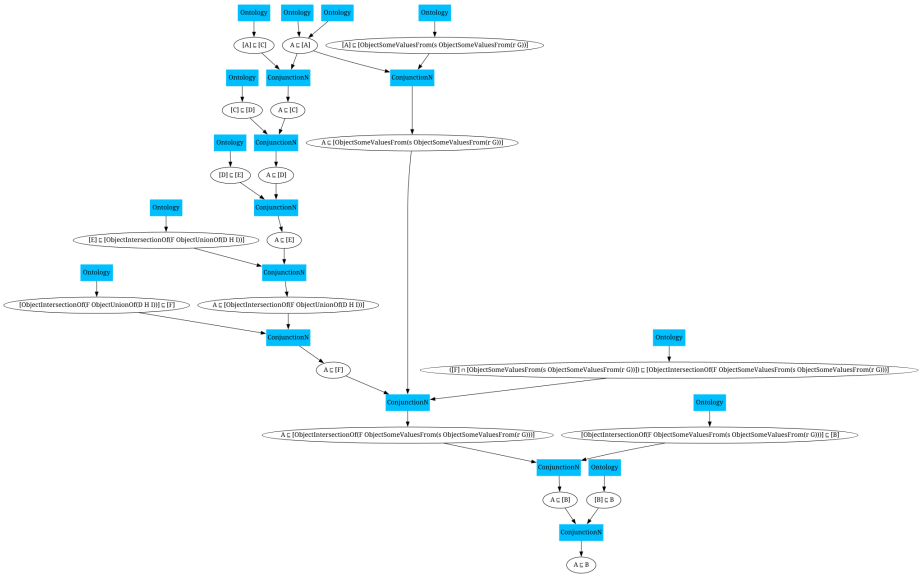

Figure 1: ALCH-Reasoner Proof for Task 1

5 Results

- test on Pizza
- nice diagrams

proof generator main task file

References

[1] Christian Alrabbaa et al. *On the Eve of True Explainability for OWL Ontologies: Description Logic Proofs with Evee and Evonne (Extended Version)*. 2022. DOI: 10.48550/ARXIV.2206.07711. URL: https://arxiv.org/abs/2206.07711.

[2] P Frazier et al. "The creation of an ontology of clinical document names". In: *Studies in health technology and informatics* 84 (Feb. 2001), pp. 94–8.

[3] P. Koopmann and R. A. Schmidt. "LETHE: A Saturation-Based Tool for Non-Classical Reasoning". In: *Proceedings of the 4th International Workshop on OWL Reasoner Evaluation (ORE-2015)*. Ed. by M. Dumontier et al. Vol. 1387. CEUR Workshop Proceedings. CEUR-WS.org, 2015. URL: http://www.cs.man.ac.uk/~schmidt/publications/KoopmannSchmidt15c.html.

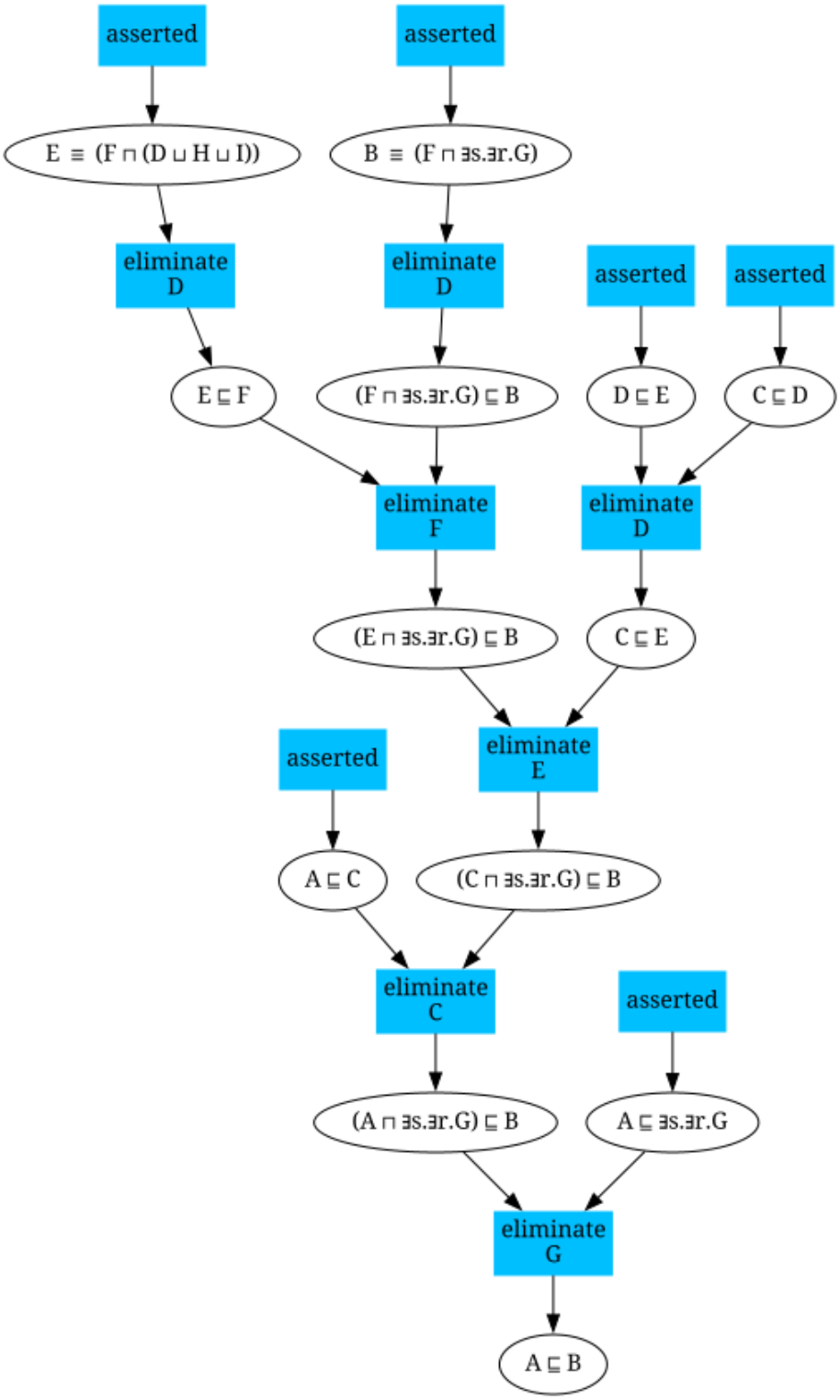

Figure 2: Lethe Proof for Task 1