Abstract

A large and complex knowledge base that models some aspect of the real world can rarely be fully specified. Two examples of such underspecification are that (i) some of the cardinality constraints are omitted; (ii) some properties of all individual instances of a class are specialized across a class hierarchy, but specific references to which particular values are specialized are omitted. Such knowledge bases are of great practical interest as they are the basis of an empirically tested knowledge acquisition system that has been used to construct a knowledge base from a significant portion of a biology textbook. In this paper, we formalize an underspecified knowledge base using answer set programming, and give a set of rules called UMAP that support inheritance reasoning in such a knowledge base.

1 Introduction

Building large and complex knowledge bases (KBs) has been an intensive topic of artificial intelligence research in general and knowledge representation and reasoning research in particular [14]. Clark & Porter proposed a method toward this goal in which a KB can be built from reusable components that can be composed automatically in response to novel questions [7, 6]. Their AAAI’97 paper on this topic won the best paper prize and this approach was implemented in a knowledge representation and reasoning system called KM [5], which has been the foundation of two knowledge acquisition systems, SHAKEN [3] and AURA [11]. Both of these systems have been empirically tested in acquiring knowledge from domain experts. The success is attributed to two key features of the KM system: prototypes and heuristic unification. There has, however, been no published computational characterization of these features. Such characterization is of interest for at least two reasons: other representation languages could support these features for similar problems, and the substantial KBs built using this approach could be used for reasoning systems other than KM.

A knowledge base in KM is a collection of frames representing classes and individuals. Classes are organized into a class hierarchy. Each frame is associated with a set of slots and slot values. Each slot value can be a class, individual, or a value. For example, the statement “Every Car has an engine and a tank connected to the engine” can be represented by the following statement in
KM:

\[
\begin{align*}
\text{(every Car has} \quad & \text{(has-engine ((a Engine called E1)))} \\
\text{has-tank ((a Tank with} \quad & \text{connected-to (((the has-engine} \\
\text{of Self) called E1))))})
\end{align*}
\]

This expression applies to all individual instances of the class Car and has two slots has-engine and has-tank. The slot has-engine (resp. has-tank) is associated with a frame that refers to a Skolem instance of class Engine (resp. Tank). The Skolem instance Tank has a further slot connected-to that has a value that is same as the Skolem instance of Engine that is the value of the has-engine slot. The use of “called E1” is similar to using a variable name in traditional logical syntax. KM also provides an alternative syntax called prototypes to encode such rules. The rules such as (1) are the most frequently occurring axiom pattern in the biology textbook KB.

While constructing a KB, it is common to specialize properties of classes along the class hierarchy. While doing so, one may need to refer to a Skolem instance that was introduced in one of the superclasses. For example, assume that we add to the KB the following expressions:

\[
\begin{align*}
(\text{Suburban has (superclasses (Car)))} & \quad (2) \\
(\text{every Suburban has (has-engine} \quad & \text{(a Engine with (size (Large)))))} \quad (3)
\end{align*}
\]

Intuitively, these axioms state that Suburban is a subclass of Car, and every suburban has a large engine. Let us consider the situation when an instance \( s_1 \) of class Suburban is asserted. Using axiom (3), we conclude that \( s_1 \) has-engine an _Eng1 with size Large, and using axiom (1), we conclude that \( s_1 \) has-engine an _Eng2. This gives us a model of the KB in which \( s_1 \) has two engines: _Eng1 and _Eng2, which is counterintuitive for this particular example.

Several alternative approaches can solve this problem:

1. While writing (3), provide a mechanism using which a user can explicitly state that the engine in axiom (3) is a specialization of the engine introduced in (1). We refer to this approach as explicit coreference.

2. Add a cardinality constraint to the KB saying that cars have exactly one engine. We refer to this approach as the cardinality constraint.

3. Support a default reasoning mechanism that can draw intuitive conclusions with the axioms as they are currently written. We refer to this approach as the underspecified knowledge base.

The explicit coreference approach has the advantage that it leaves no ambiguity, but it has a major disadvantage: it breaks the modularity of class definitions in that while writing a class definition, we must refer to other classes. While creating the class Suburban, we must refer to its superclass Car, and explicitly say which specific engine value we are specializing. If after having created Car and Suburban, suppose we introduce a superclass Vehicle and we want this class to also have a value for has-engine, we must refer to its subclasses and make sure all of them now specialize the Engine value introduced in Vehicle. In case of multiple inheritance, for example, if a Car were to be a subclass of of Vehicle and Gas-driven objects each of which provide an Engine, the knowledge base author must resolve how the multiple inheritance must be handled while defining Car. The approach breaks down completely when we need to answer a novel question about an
object that must be an instance of multiple classes, and the KB does not have knowledge about all the coreferences. The extra work of fully specifying coreferences starts to become onerous and is unnecessary for a large number of practical situations in which each class provides either one value of a certain type or multiple values of the same type but which could either be distinguished based on their slot values or the difference between multiple values does not matter. This disadvantage is especially a limiting factor when the knowledge base is to be authored by a domain expert who is not well-versed in logical knowledge representation and the knowledge base must evolve over a period of time.

The cardinality constraint approach allows us to conclude the equality of $Eng_1$ and $Eng_2$. It can work for many situations, but in some cases it is incorrect to use such constraints. For example, race cars may have more than one engine, and it is too strong to add a constraint that every car has exactly one engine. The KM system supports such reasoning using cardinality constraints when the constraints are available, but in situations where it is incorrect to add constraints such reasoning cannot be used.

In an underspecified knowledge base approach, we assume that the inherited and locally defined engines must be the same unless there is a reason to believe otherwise. Thus, the class definitions for Car, Suburban, Vehicle, and Gas-driven objects could be written without making any reference to each other. At the time of answering questions, the reasoning system is able to resolve the underspecified reference between the Engine values, and as a default inference, assumes that they must be the same. If the KB contained knowledge to the contrary, for example, for a Race Car, we introduce a new engine such as Turbo Diesel Engine, and we had a statement in the KB saying that the class Turbo Engine is disjoint from Gas Engine, then the system will assume that an inherited Gas-driven engine could not be the same as a Turbo Engine. By supporting such reasoning with an underspecified knowledge base, we retain the modularity of class definitions. The users can write their knowledge base in a modular fashion, and be confident that any obvious missing details will be filled in by the reasoning mechanism at the time of answering questions. The KM system implements heuristic unification to do such default reasoning with an underspecified knowledge base.

The focus of this paper is to define a default reasoning mechanism called unification mapping or UMAP. UMAP is motivated by the heuristic unification in KM. UMAP has a declarative specification in Answer Set Programming (ASP) [16, 18], a declarative problem solving approach using logic programming under answer set semantics [10]. The use of ASP also enables reasoning with disjunction and negation for which KM is not well suited. UMAP has the same behavior as heuristic unification in KM for a number of practical examples. Thus, UMAP puts an empirically useful and well-tested behavior in a rigorous formal framework.

We will start with a review of logic programming with answer set semantics. We then define a simple description language suitable for the encoding of underspecified knowledge bases (KB) in logic programs. The KB will contain specifications about classes, individuals, and relationships between individuals. It might not contain all explicit specializations or lack constraints. Then we discuss the four principles for the unification mapping between terms in reasoning with an underspecified knowledge base. First is the well-known specificity principle in default reasoning and the second one is specific to this application, called the specialization principle, which dictates that the specificity principle should be applied in a controlled manner. The third principle aims at removing redundant specification, and the fourth principle ensures that unification between terms is consistently applied across all specifications. We then present an ASP program that implements
these principles. Finally, we relate our formalization to that of the KM system and discuss some potential uses of the newly developed ASP program.

2 Logic Programming and Answer Sets

A logic program $\Pi$ is a set of rules of the form

$$c_1 \mid \ldots \mid c_k \leftarrow a_1, \ldots, a_m, \text{not } a_{m+1}, \ldots, \text{not } a_n$$

(4)

where $0 \leq m \leq n$, $0 \leq k$, each $a_i$ or $c_j$ is a literal of a propositional language\(^1\) and not represents negation-as-failure. When $n = 0$, the rule is called a fact. When $k = 0$, the rule is called a constraint. A negation as failure literal (or naf-literal) is of the form not $a$ where $a$ is a literal.

For a rule $r$ of the form (4), $\text{head}(r)$ denotes the set $\{c_1, \ldots, c_k\}$; $\text{pos}(r)$ and $\text{neg}(r)$ denote $\{a_1, \ldots, a_m\}$ and $\{a_{m+1}, \ldots, a_n\}$, respectively. For a program $P$, $\text{lit}(P)$ denotes the set of literals occurring in $P$.

Consider a set of ground literals $X$. $X$ is consistent if there exists no atom $a$ such that both $a$ and $\neg a$ belong to $X$. The body of a rule $r$ of the form (4) is satisfied by $X$ if $\text{neg}(r) \cap X = \emptyset$ and $\text{pos}(r) \subseteq X$. A rule of the form (4) with nonempty head is satisfied by $X$ if either its body is not satisfied by $X$ or $\text{head}(r) \cap X \neq \emptyset$. A constraint is satisfied by $X$ if its body is not satisfied by $X$.

For a consistent set of ground literals $S$ and a program $\Pi$, the reduct of $\Pi$ w.r.t. $S$, denoted by $\Pi^S$, is the program obtained from the set of all rules of $\Pi$ by deleting (i) each rule that has a naf-literal not $a$ in its body with $a \in S$, and (ii) all naf-literals in the bodies of the remaining rules.

$S$ is an answer set (or a stable model) of $\Pi$ \[10\] if it satisfies the following conditions: (i) If $\Pi$ does not contain any naf-literal (i.e., $m = n$ in every rule of $\Pi$) then $S$ is a minimal consistent set of literals that satisfies all the rules in $\Pi$; and (ii) If $\Pi$ does contain some naf-literal ($m < n$ in some rule of $\Pi$), then $S$ is an answer set of $\Pi$ if $S$ is the answer set of $\Pi^S$.

Several new extensions have been introduced to enhance the modeling capability of logic programming. In this paper, we will make use of cardinality constraint atom \[19\] of the following form:

$$l \{b_1, \ldots, b_k\} u$$

(5)

where $b_j$’s are literals and $l$ and $u$ are two integers, $l \leq u$. An atom of the form (5) is true in a set of literals $X$ if at least $l$ and at most $u$ literals of the set $\{b_1, \ldots, b_k\}$ are true in $X$. Cardinality constraint atoms can be used anywhere a literal can be used. Using this type of atom, one can greatly reduce the number of rules of programs in answer set programming.

ASP has been successfully applied to several applications such as bioinformatics \[9\]; diagnosis for the space shuttle \[2\]; linear temporal logic (LTL) model checking \[12\]; planning \[20\]; security protocols verification \[1\]; and so on. The existence of answer set solvers, whose performance is comparable to state-of-the-art SAT solvers in many domains (e.g., CLASP \[8\]), the theoretical building block results of ASP, and its recent extensions (e.g., aggregates) make ASP an appealing language for modeling of and reasoning with knowledge-intensive applications.

---

\(^1\)Rules with variables are viewed as a shorthand for the set of its ground instances.
3 Underspecified Knowledge Bases

In this paper, we consider underspecified knowledge bases that are built over sorted signatures containing at least the three sorts class, slot, and individual. A sorted signature $\sigma$ is a collection of constants, predicate symbols, and function symbols and is represented by a tuple $\langle C, P, F \rangle$ where $C$, $P$, and $F$ are pairwise disjoint and

- $C$ is a collection of sorted constants, including class, slot, constants, and other constants;
- $P$ is a set of sorted first-order predicates that contains at least the following predicates: class, subclass_of, instance_of, and slot whose arity is class, $\text{class} \times \text{class}$, $\text{individual} \times \text{class}$, and $\text{slot} \times \text{individual} \times \text{class}$, respectively; and
- $F$ is a set of sorted first-order functions that contains at least a set of Skolem function mapping individuals to classes.

We denote the set of class, slot, individual, and other constants by $C_c$, $C_s$, $C_i$, and $C_o$; the set of Skolem functions by $F_s$. We define the following types of axioms.

- A class axiom is an atom of the form
  \[
  \text{class}(c)
  \]
  for $c \in C_c$. Intuitively, this axiom states that the constant $c$ denotes a class. For example, if $\text{car} \in C_c$ then $\text{class}(\text{car})$ is a class atom stating that car encodes the class of cars.
- A constant axiom is an axiom of the form
  \[
  \text{constant}(o)
  \]
  for $o \in C_i \cup C_o$. This axiom indicates that $o$ is a constant.
- A subclass axiom is an atom of the form
  \[
  \text{subclass}\_of(c,c')
  \]
  for $c, c' \in C_c, c \neq c'$. This axiom states that $c$ is a subclass of $c'$. E.g., the atom $\text{subclass}\_of(\text{suburban}, \text{car})$ states that the class of Suburbans, denoted by $\text{suburban}$, is a subclass of the class of cars, denoted by $\text{car}$.
- An instance axiom is a rule of the form
  \[
  \text{instance}\_of(i,c)
  \]
  for some $i \in C_i$, and $c \in C_c$. This axiom states that the constant $i$ is an individual of the class $c$. For example, to state that $s_1$ is a $\text{suburban}$, we write
  \[
  \text{instance}\_of(s_1, \text{suburban}).
  \]

\[\text{From the knowledge representation perspective, allowing } c = c' \text{ in the specification results in redundancy. A simple rule } \text{subclass}\_of(X,X) \leftarrow \text{class}(X) \text{ can be used to enforce reflexivity of the } \text{subclass}\_of \text{ relation.}\]
• A **descriptive axiom** is a rule of the form

\[
2 \left\{ \text{slot}(s, X, f_1(X)), \\
    \text{instance}_o f(f_1(X), c_1) \right\} \quad 2 \leftarrow \text{instance}_o f(X, c) \tag{10}
\]

or

\[
3 \left\{ \text{slot}(s, f_1(X), f_2(X)), \\
    \text{instance}_o f(f_1(X), c_1), \\
    \text{instance}_o f(f_2(X), c_2) \right\} \quad 3 \leftarrow \text{instance}_o f(X, c) \tag{11}
\]

where \(f_1, f_2\) are Skolem functions in \(F_s\), \(c, c_1, c_2 \in C_c\) and \(s \in C_s\), and \(X\) is a variable and \(c \not\in \{c_1, c_2\}\).

Intuitively, a descriptive axiom of the form (10) or (11) describes a property of individuals belonging to class \(c\), encoded by the slot-atom \(\text{slot}(s, X, f_1(X))\) or \(\text{slot}(s, f_1(X), f_2(X))\) respectively; (10) states for each individual \(X\) in \(c\), \(X\) is related to \(f_1(X)\)—an individual in class \(c_1\)—via the slot \(s\); (11) states that for each individual \(X\) in \(c\), \(f_1(X)\)—an individual in class \(c_1\)—is related to \(f_2(X)\)—an individual in the class \(c_2\)—via the slot \(s\). We observe that descriptive axioms help us capture rules of form (1), and more generally, the knowledge that one would find in KM prototypes.

• A **value axiom** is a rule of the form

\[
\text{slot}(s, X, v) \leftarrow \text{instance}_o f(X, c) \tag{12}
\]

or

\[
2 \left\{ \text{slot}(s, f(X), v), \\
    \text{instance}_o f(f(X), c') \right\} \quad 2 \leftarrow \text{instance}_o f(X, c) \tag{13}
\]

where \(c, c' \in C_c\), \(c \neq c'\), \(f \in F_s\), and \(v \in C_v\). Intuitively, (12)-(13) allow for the specification of specific value of a slot.

In the following we will assume that for each Skolem function \(f \in F_s\) there exists at most one \(d \in C_c\) such that \(\text{instance}_o f(f(X), d)\) appears in the axioms (10)–(13) since the Skolemization of a formula creates different Skolem functions for different variables.

We define the notion of a domain description as follows.

**Definition 1.** A domain description \(D(\sigma)\) over a signature \(\sigma\) is a logic program \(D(\sigma) = D_b \cup D_e\) where \(D_b \cap D_e = \emptyset\) and

• \(D_b\) is a set of axioms of the form (6)-(13); and

• \(D_e\) is set of ASP rules defined over the signature \(\sigma\).

We call \(D_b\) and \(D_e\) the basic and extended part of \(D\). We say that \(D\) is basic if \(D_e = \emptyset\).

For simplicity of the presentation, we delay the consideration of constraints in domain description to the later part of the paper.

The next example shows that domain descriptions are sufficiently expressive for the representation of knowledge encoded by KM’s expressions.
Example 1. Let us consider an extended version of the car domain discussed in the introduction. The signature $\sigma_0$ consists of $C_c = \{\text{car, engine, tank, suburban}\}$, $C_i = \{s_1\}$, $C_s = \{\text{has engine}, \text{has tank}, \text{connected}\}$, $F_s = \{\_\text{Eng}, \_\text{Tank}\}$. $D(\sigma_0)$ is given by the following axioms:

\[
\begin{align*}
\text{class}(X) & \quad \text{for } X \in C_c \\
\text{instance}_o f(s_1, \text{suburban}) \\
\text{subclass}_o f(\text{suburban, car})
\end{align*}
\]

and the following descriptive axioms (the third axiom is simplified by omitting the two instance axioms, which have already been included in the first two):

\[
\begin{align*}
2 \{ & \text{slot(\text{has engine}, X, \_Eng(X))}, \\
& \text{instance}_o f(\_\text{Eng(X)}, \text{engine}) \} \quad \text{2 } \leftarrow \text{instance}_o f(X, \text{car}) \\
2 \{ & \text{slot(\_T\text{ank(X)}, \text{tank})}, \\
& \text{instance}_o f(\text{\_Tank(X)}, \text{tank}) \} \quad \text{2 } \leftarrow \text{instance}_o f(X, \text{car}) \\
\text{slot(connected, \_Eng(X), \_Tank(X))} & \leftarrow \text{instance}_o f(X, \text{car}) \\
\end{align*}
\]

(14)

Observe that a domain description $D(\sigma)$ is a logic program that could be used to reason about properties of individuals encoded in $D(\sigma)$ under the answer set semantics represented by the slot-atoms. This reasoning will need to take into consideration the inheritance information encoded in $D(\sigma)$ in axioms of the form (6), (8), and (9). This can be achieved by the set $\Pi_I$, which allows for reasoning about class membership and subclass relationship in inheritance reasoning encoded in the following rules:

\[
\begin{align*}
\text{subclass}_o f(C_1, C_3) & \leftarrow \text{subclass}_o f(C_1, C_2), \text{subclass}_o f(C_2, C_3). \\
\text{instance}_o f(X, C_1) & \leftarrow \text{instance}_o f(X, C_2), \text{subclass}_o f(C_2, C_1).
\end{align*}
\]

(15) (16)

To use domain descriptions in reasoning about its individuals, we need to take $\Pi_I$ into consideration. We define the notion of an underspecified knowledge base as follows.

Definition 2. An underspecified knowledge base (or knowledge base) defined over a signature $\sigma$ is the program $D(\sigma) \cup \Pi_I$ where $D(\sigma)$ is a domain description over $\sigma$.

A knowledge base, under the answer set semantics, will entail various slot-atoms about its individuals. A feasible but naive approach is to use these atoms to characterize the properties of an individual as shown in the next example.

Example 2. Let $KB_0 = D(\sigma_0) \cup \Pi_I$. It is easy to see that $KB_0$ has a unique answer set containing the following slot-atoms: $\text{slot(connected, \_Eng(s_1), \_Tank(s_1))}$, $\text{slot(has engine, s_1, \_Eng(s_1))}$, and $\text{slot(has tank, s_1, \_Tank(s_1))}$. These atoms belong to the answer set because of the axioms in the group (14) and the fact that $s_1$ is an instance of suburban and thus is also a member of car (due to $\Pi_I$).

The result in Ex. 2 is intuitive and suggests that slot-atoms can be used for reasoning about individuals in an underspecified knowledge base. The next example shows that the approach is too weak to deal with inheritance.
Example 3. Let $\sigma_1$ be the signature $\sigma_0$ extended with the class constants \{lEngine, lTank\} and the Skolem functions \{lEng, lTank\} (the prefix ‘l’ stands for ‘large’). Let $D_1(\sigma_1)$ be $D_0(\sigma_0)$ extended with the following axioms:

\[
\begin{align*}
  &\text{class}(\text{lEngine}). \quad \text{subclass_of}(\text{lEngine}, \text{engine}). \\
  &\text{class}(\text{lTank}). \quad \text{subclass_of}(\text{lTank}, \text{tank}). \\
\end{align*}
\]

and the descriptive axioms

\[
\begin{align*}
  2 \{ \text{instance_of}(\text{lEng}(X), \text{lEngine}), \\
  &\text{slot}(\text{has_engine}, X, \text{lEng}(X)) \} \leftarrow \text{instance_of}(X, \text{sururban}). & (17) \\
  2 \{ \text{instance_of}(\text{lTank}(X), \text{lTank}), \\
  &\text{slot}(\text{has_tank}, X, \text{lTank}(X)) \} \leftarrow \text{instance_of}(X, \text{suburban}). & (18)
\end{align*}
\]

Let $KB_1 = D_1(\sigma_1) \cup \Pi_I$. Again, we can show that it has a unique answer set containing the slot-atoms in Fig. 1.

\[
\begin{align*}
  &\text{slot}(\text{has_engine}, s_1, \text{lEng}(s_1)) \\
  &\text{slot}(\text{has_tank}, s_1, \text{lTank}(s_1)) \\
  &\text{slot}(\text{connected}, \text{lEng}(s_1), \text{lTank}(s_1)) \\
  &\text{slot}(\text{has_engine}, s_1, \text{lEng}(s_1)) \\
  &\text{slot}(\text{has_tank}, s_1, \text{lTank}(s_1))
\end{align*}
\]

Figure 1: Slot atoms entailed by $KB_1$

The presence of $\text{slot}(\text{has_engine}, s_1, \text{lEng}(s_1))$ and $\text{slot}(\text{has_tank}, s_1, \text{lTank}(sa))$ is due to the newly introduced descriptive axioms.

Observe that $D_1(\sigma_1)$ contains several descriptive axioms about the slot $\text{has_engine}$ of the individual $s_1$. These axioms are related via the subclass relationship between classes. Consider the first descriptive axioms in (14) and (17)-(18). Both describe the slot $\text{has_engine}$ of individuals in the class $\text{suburban}$; the former via inheritance and the latter a direct specification. This is the reason for the presence of different slot-atoms (e.g., $\text{slot}(\text{has_engine}, s_1, \text{lEng}(s_1))$ and $\text{slot}(\text{has_engine}, s_1, \text{lEng}(s_1))$)—indicating that $s_1$ has two engines denoted by $\text{lEng}(s_1)$ and $\text{lEng}(s_1)$—in the answer set of $KB_1$. However, because $\text{lEngine}$ is a subclass of $\text{Engine}$, it is intuitive to conclude that $s_1$ has only one engine $\text{lEng}(s_1)$, i.e., to unify it with $\text{lEng}(s_1)$. Similarly, we have that $\text{lTank}(s_1)$ should be unified with $\text{lTank}(s_1)$. Our goal in the next section is to develop a declarative formalization of this type of unification.

4 UMAP-Atoms

We develop a set of logic programming rules defining a special type of atom, called umap-atom, to characterize unification mapping in underspecified knowledge bases. Each umap-atom is of the form $\text{umap}(N, X, Y)$ and has the same meaning as that of $\text{slot}(N, X, Y)$, except that a umap-atom might encode a set of slot-atoms of an individual. Ideally, the set of umap-atoms should represent the intuitive answer that a user would expect from that user’s knowledge base. For example, given $KB_1$ (Ex. 3), the slot-atoms in Fig. 1 should be represented by three umap-atoms: $\text{umap}(\text{connected}, \text{lEng}(s_1), \text{lTank}(s_1))$, $\text{umap}(\text{has_engine}, s_1, \text{lEng}(s_1))$, and $\text{umap}(\text{has_tank}, s_1, \text{lTank}(s_1))$. We begin with a discussion about the principles that will be used for UMAP.
4.1 Principles for Unification Mapping

Intuitively, a umap-atom unifies a set of slot-atoms, e.g., umap(has_engine, s1, lEng(s1)) represents the set consisting of slot(has_engine, s1, lEng(s1)) and slot(has_engine, s1, Eng(s1)). Therefore, the key questions in defining the umap-atoms are

1. How can such a set of slot-atoms be identified and when should the atoms be unified?
2. What are the consequences of a unification, if it is executed?

In the following, we will answer these questions by developing principles that should be applied in the unification process. First, let us define some additional notions. Let \( D \) be a domain description. In the following, by an atom, we mean a grounded atom occurring in the knowledge base \( D \cup \Pi_f \).

We refer to the second and third argument of a slot-atom occurring in \( D_b \) as a slot-term of \( D \) (or term, for short), i.e., if \( \text{slot}(S, X, Y) \) occurs in \( D_b \) then \( X \) and \( Y \) are terms. It is easy to see that by the definition of domain descriptions, a term is of the form \( X \) (or \( f(X) \)) for an instance \( X \) (or a Skolem function \( f \)) of some class \( c \); furthermore, if \( f(X) \) is a term then \( D \) will contain a rule stating that it is a member of some class. Due to the subclass relationship, a term can be a member of several classes. A class \( c \) is a most specific class of a term \( X \) if \( c \) is a minimal element (with respect to the ordering defined by the subclass relation) among classes having \( X \) as an instance. Two terms \( f(X) \) and \( g(X) \) are compatible if it can be proved that they belong to the same class. In Ex. 3, for any instance \( X \) of the class \textit{suburban}, we have that \( _{\text{iEng}}(X) \) and \( _{\text{Eng}}(X) \) are compatible terms since they are an instance of the class \textit{engine}; the most specific class of \( _{\text{iEng}}(X) \) and \( _{\text{Eng}}(X) \) is \textit{iEngine} and \textit{engine}, respectively. A term \( X \) is more specific than a term \( Y \) if \( X \) and \( Y \) are compatible and the most specific class of \( X \) is more specific than the most specific class of \( Y \).

The first principle that we would like to enforce in the definition of the umap-atoms is the well-known specificity principle in inheritance reasoning. In the context of this paper, it is as follows.

**P1** Specificity principle: in selecting terms for the construction of umap-atoms, more specific terms should be preferred over less specific ones.

Applying this principle to the construction of the umap-atoms implies that if umap\((N, X, Y)\) is entailed by the knowledge base then

- It should be obtained from an atom \( \text{slot}(N, X_1, Y_1) \) by substituting \( X_1 \) and \( Y_1 \) with compatible terms \( X \) and \( Y \), respectively; and
- \( X \) and \( Y \) are more specific than or at least as specific as, according to the inheritance principle, \( X_1 \) and \( Y_1 \), respectively.

According to (P1), \( \text{slot}(\text{has_engine}, s_1, _{\text{iEng}}(s_1)) \) and \( \text{slot}(\text{has_engine}, s_1, _{\text{Eng}}(s_1)) \) should be unified into a single umap-atom umap\((\text{has_engine}, s_1, _{\text{iEng}}(s_1)) \) since \( _{\text{iEng}}(s_1) \) is compatible to and more specific than \( _{\text{Eng}}(s_1) \). This is indeed what we expect given \( D_1(\sigma_1) \). However, the use of the more specific principle is in some situations too strong, as illustrated in the next example.

**Example 4.** Let \( \sigma_2 \) be the signature \( \sigma_1 \) (Ex. 3) extended with a Skolem function \( _{\text{eEng}} \) (‘\textit{eEng}’ stands for ‘extra engine’) and \( D_2(\sigma_2) \) be \( D_1(\sigma_1) \) extended with the axiom

\[
2 \left\{ \begin{array}{l}
  \text{slot}(\text{has_engine}, X, _{\text{eEng}}(X)), \\
  \text{instance_of}(_{\text{eEng}}(X), \text{engine})
\end{array} \right\} \quad 2 \leftarrow \text{instance_of}(X, \text{car}) \quad (19)
\]
Intuitively, (19) and (14) represent two different constraints on the slot \textit{has engine} of individuals in \textit{car}, which state that each car has two engines.

Consider an instance $s_0$ of \textit{car}. Intuitively, we have that $eEng(s_0)$ and $Eng(s_0)$ are compatible but neither is preferred over the other. As such, the two atoms $\text{slot(has engine, } s_0, eEng(s_0))$ and $\text{slot(has engine, } s_0, Eng(s_0))$ will yield two $\text{umap}$-atoms $\text{umap(has engine, } s_0, eEng(s_0))$ and $\text{umap(has engine, } s_0, Eng(s_0))$. This is intuitive.

Now, consider again the instance $s_1$ of \textit{suburban}. Observe that the term $lEng(s_1)$ is compatible to, and more specific than, both terms $Eng(s_1)$ and $eEng(s_1)$. Thus, applying the principle (P1) results in a single $\text{umap}$-atom $\text{umap(has engine, } s_1, lEng(s_1))$, i.e., $s_1$ has a single engine. This is counter intuitive since $s_1$, being an instance of \textit{car}, should have two engines.

Example 4 shows that the use of the most preferred term could lead to the loss of constraints on an individual with set valued slots. We therefore introduce the next principle in the definition of $\text{umap}$-axioms.

\textbf{(P2) Specialization Principle}: Given a slot $s$ and a class $c$, the application of the specificity principle should be limited to at most one possible value of $s$ at $c$. Furthermore, if the application of (P1) does not violate (P2) then (P1) should be applied.

Observe that while (P2) limits the application of (P1), it also guarantees that (P1) is applied on as many terms that have a more specific term as possible. This can be seen in the next example.

\textbf{Example 5}. Returning to Ex. 4, (P2) should yield two possible outcomes:

- $lEng(s_1)$ overrides $eEng(s_1)$ which gives $\text{umap(has engine, } s_1, lEng(s_1))$ and $\text{umap(has engine, } s_1, eEng(s_1))$; and

- $lEng(s_1)$ overrides $eEng(s_1)$ which yields $\text{umap(has engine, } s_1, lEng(s_1))$ and $\text{umap(has engine, } s_1, Eng(s_1))$.

Our next principle deals with the redundancy of specifications, which often occurs when knowledge bases are combined or merged. A slot can have multiple specifications and some might be redundant. Consider $D_1(\sigma_1)$ with the additional descriptive axiom:

$$2 \left\{ \text{slot(has engine, } X, \text{Eng1(X))}, \text{instance of(Eng1(X), engine)} \right\} 2 \leftarrow \text{instance of(X, suburban)}$$  \hspace{1cm} (20)

This axiom says that every suburban has an engine. Intuitively, this axiom is redundant because the conclusion that a suburban has an engine follows from the fact that it is a car and every car has an engine. An alternative view is that (20) is a more specific specification of the slot \textit{has engine} in (14) and thus should override the specification in (14). We call this the \textit{redundancy principle} as follows.

\textbf{(P3) Redundancy Principle}: In the presence of multiple specifications of a slot for an individual, the most-specific slot specification overrides less-specific ones.
(P1)-(P3) are used to decide whether a term $x$ should be unified with a term $y$ of the same slot. They do not relate different slots of the same classes. For example, $Eng(s_1)$ occurs in both slot(has_engine, $s_1$, $Eng(s_1)$) and slot(connected, $Eng(s_1)$, $Tank(s_1)$). Clearly, it is intuitive to require that if $Eng(s_1)$ is unified with a term $t$ then $t$ should be used in both umap-atoms derived from these two slot-atoms. This is stated in the next principle.

(P4) Consistency Principle: If a unification between $x$ and $y$ takes place at class $c$ then it should be applied in every slot of class $c$.

4.2 Computing UMAP-Atoms using ASP

We will now specify a program, denoted by $\Pi_R$, for defining the umap-axioms. $\Pi_R$ enforces the principles (P1)–(P3). It consists of rules for reasoning about specificity and defining the predicate umap. We next describe the rules of $\Pi_R$, dividing them into the set of domain-dependent rules $\Pi^d_R$ and the set of domain-independent rules $\Pi^i_R$. Assume that $KB(\sigma)$ is a knowledge base over $\sigma$.

- **Domain-dependent rules**: Rules in this group declare the compatibility between terms constructed from Skolem functions over individuals. For each pair of $f_1, f_2 \in F_s$ appearing in an axiom of the form (10) or (11), $\Pi^d_R$ contains the rule

$$
\text{compatible}(f_1(X), f_2(X)) \leftarrow \text{constant}(X), \text{instance}_of(f_1(X), D), \text{class}(D), \text{instance}_of(f_2(X), D).
$$

Furthermore, for each descriptive axiom of the form (10) $\Pi^d_R$ contains the rules

$$
\text{range}(f_i(X), c_i) \leftarrow \text{constant}(X) \quad (22)
$$

$$
\text{domain}(f_i(X), c) \leftarrow \text{constant}(X) \quad (23)
$$

Atoms of the form range($Z, c$) and domain($Z, c$) encode the range and domain of the Skolem function occurring in $Z$.

- **Domain-independent rules**: The set of independent rules $\Pi^i_R$ defines various predicates related to the compatibility between constants and terms of the form $f(X), f \in F_s$, and the preference between compatible terms under the specificity principle.

  - **Terms, compatibility between terms, and type of slots**: This group includes the following rules.

$$
2\{\text{term}(X), \text{term}(Y)\} \leftarrow \text{slot}(S, X, Y). \quad (24)
$$

$$
\text{term}\_\text{value}(Y, S, C) \leftarrow \text{slot}(S, X, Y), \text{range}(Y, C). \quad (25)
$$

$$
\text{compatible}(X, X) \leftarrow \text{constant}(X). \quad (26)
$$

Rule (24) defines terms that will be used in defining umap-atoms. Rule (25) associates a value (a term) with a slot at a class for use in UMAP since UMAP is restricted to values for the same slot at the same class. For example, terms referring to instances of the class engine and used in the slot has_engine can be unified with each other; they should not be unified with terms referring to the class tank. Rule (26), with (21), defines the compatibility between two terms $X$ and $Y$.
Specificity between terms: Rules in this group define a more specific relation between terms and the most specific term of a given term.

\[
\text{instance}_\text{by}_\text{inheritance}(X,C) \leftarrow \text{instance}_\text{of}(X,C_1), C_1 \neq C, \\
\quad \text{subclass}_\text{of}(C_1,C).
\]

\[
\text{most}_\text{specific}_\text{class}(X,C) \leftarrow \text{instance}_\text{of}(X,C), \\
\quad \text{not instance}_\text{by}_\text{inheritance}(X,C).
\]

\[
\text{more}_\text{specific}_\text{term}(X,Y) \leftarrow \text{compatible}(X,Y), \text{subclass}_\text{of}(C_1,C_2), \\
\quad C_1 \neq C_2, \text{most}_\text{specific}_\text{class}(X,C_1), \\
\quad \text{most}_\text{specific}_\text{class}(Y,C_2).
\]

\[
\text{has}_\text{more}_\text{specific}_\text{term}(Y) \leftarrow \text{more}_\text{specific}_\text{term}(X,Y), \text{compatible}(X,Y).
\]

\[
\text{most}_\text{specific}_\text{term}(X,Y,S,C) \leftarrow \text{compatible}(X,Y), \text{term}_\text{value}(X, S, C), \\
\quad \text{term}_\text{value}(Y, S, C), \text{more}_\text{specific}_\text{term}(X,Y), \\
\quad \text{not has}_\text{more}_\text{specific}_\text{term}(X).
\]

\[
\text{most}_\text{specific}_\text{term}(X,Y,S,C) \leftarrow \text{compatible}(X,Y), \text{subclass}_\text{of}(C,C_1), \\
\quad \text{term}_\text{value}(X, S, C), \text{term}_\text{value}(Y, S, C_1), \\
\quad \text{more}_\text{specific}_\text{term}(X,Y), \\
\quad \text{not has}_\text{more}_\text{specific}_\text{term}(X).
\]

Rule (27) says that \(X\) is an instance of a class \(C\) by inheritance (\text{instance}_\text{by}_\text{inheritance}(X,C)) if it is an instance of a class \(C_1\) that is a proper subclass of \(C\). Rule (28) identifies the most specific class of a term \(X\). Rule (29) defines the more specific relation between compatible terms: \(X\) is more specific than \(Y\) if \(X\) and \(Y\) are compatible terms and the most specific class of \(X\) is a subclass of the most specific class of \(Y\). Rule (30) identifies a term that has a compatible and more specific term. \(\text{most}_\text{specific}_\text{term}(X,Y,S,C)\), defined in (31)-(32), encodes that \(X\) is a most specific term of \(Y\) and can be used in defining \text{umap}-atoms for the slot \(S\) at the class \(C\). The rules state that \(X\) is a most specific term of \(Y\) if they are compatible, \(X\) is more specific than \(Y\), and there exists no term that is more specific than \(X\).

Computing Redundancy: Rules in this group identify redundant specifications and enforce the principle (P3).

\[
2 \left\{ \begin{array}{l}
\text{replace}(Y_1,Y), \\
\text{redundant}(S,X,Y)
\end{array} \right\} 2 \leftarrow \text{class}(C), \text{slot}(S,X,Y), \text{slot}(S,X,Y_1),
\]

\[
\quad \text{range}(Y,C), \text{range}(Y_1,C), \\
\quad \text{domain}(Y,D), \text{domain}(Y_1,D_1), \\
\quad \text{subclass}_\text{of}(D_1,D).
\]

\[
\text{replace}(Y_2,Y) \leftarrow \text{replace}(Y_2,Y_1), \text{replace}(Y_1,Y).
\]

\[
\text{selected}_\text{to}_\text{replace}(Y) \leftarrow \text{replace}(Y_1,Y).
\]

(33) declares \text{redundant}(S,X,Y) which encodes that \text{slot}(S,X,Y) is redundant if there exists another specification \text{slot}(S,X,Y_1) for the same class at a more specific class. In
this case, Y should be unified with \( Y_1 \), denoted by \( \text{replace}(Y_1, Y) \). (34) says that \( \text{replace} \)
is transitive and (35) says that \( Y \) will be replaced by some term if it is unified with some other term.

- **Unification rules**: Rules in this group create a mapping between terms for the unification mapping according to the principles \( (\mathbf{P1}) \), \( (\mathbf{P2}) \), and \( (\mathbf{P4}) \)

\[
0 \left\{ \begin{array}{l}
\text{pick\_more\_specific}(X,Y,S,C) : \\
\text{most\_specific\_term}(X,Y,S,C)
\end{array} \right\} 1 \leftarrow \text{term}(Y). \tag{36}
\]

Intuitively, \( \text{pick\_more\_specific}(X,Y,S,C) \) states that the most specific term \( X \) of \( Y \), w.r.t. the slot \( S \) at the class \( C \), will be unified with \( Y \). Rule (36) creates a mapping from the set of terms into the set of their most specific terms. To ensure that the two principles are enforced, we need the following rules.

\[
\text{override}(Y,S,C) \leftarrow \text{pick\_more\_specific}(X,Y,S,C). \tag{37}
\]

\[
\text{selected\_override}(Y) \leftarrow \text{pick\_more\_specific}(X,Y,S,C). \tag{38}
\]

\[
\text{used\_for\_override}(X,S,C) \leftarrow \text{pick\_more\_specific}(X,Y,S,C). \tag{39}
\]

These rules keep track of the terms that have been selected for unification mapping by projecting different elements of \( \text{pick}(X,Y,S,C) \). (37) and (38) define \( \text{override}(Y,S,C) \) and \( \text{selected\_override}(X) \), respectively, indicating that the value \( Y \) of slot \( S \) at class \( C \) has been selected to be unified with some other term. These atoms are used in the enforcement of \( (\mathbf{P1}) \) and \( (\mathbf{P2}) \) by the following constraints.

\[
\leftarrow \text{term\_value}(X,S,C), \text{term\_value}(Y,S,C), \text{term}(Z), X! = Y; \tag{40}
\]

\[
\text{pick\_more\_specific}(Z,X,S,C), \text{pick\_more\_specific}(Z,Y,S,C).
\]

\[
\leftarrow \text{most\_specific\_term}(X,Y,S,C), \tag{41}
\]

\[
\not \text{override}(Y,S,C), \not \text{used\_for\_override}(X,S,C).
\]

(40) guarantees that one term is not unified with two different and less specific terms. (41) ensures that the specificity principle is applied whenever it is possible. The combination of these two rules enforces \( (\mathbf{P2}) \).

The next rules finalize the mapping for the unification process.

\[
\text{unify\_with}(X,Y) \leftarrow \text{pick\_more\_specific}(X,Y,S,C). \tag{42}
\]

\[
\text{unify\_with}(X,Y) \leftarrow \text{replace}(X,Y), \not \text{selected\_to\_replace}(X). \tag{43}
\]

\[
\text{unify\_with}(X,X) \leftarrow \text{term}(X), \not \text{selected\_override}(X), \not \text{selected\_to\_replace}(X). \tag{44}
\]

Rules (42)-(44) enforce \( (\mathbf{P4}) \) by defining the predicate \( \text{unify\_with}(X,Y) \). If \( \text{unify\_with}(X,Y) \) holds, then the term \( X \) should replace the term \( Y \). (42) states that if \( X \) has been selected to replace \( Y \) via \( \text{pick\_more\_specific}(X,Y,S,C) \), then \( Y \) should be identified with \( X \). (43) states that if \( X \) has been selected to be replaced by any other term \( Y \), then \( Y \) should be identified with \( X \). (44) indicates that if \( X \) has not been selected to be replaced by any other term then, \( X \) is identified by itself.
Creating umap-atoms: Rule (45) defines the predicate $\text{umap}(S, X, V)$. It states that $\text{umap}(S, X, V)$ should be obtained from some slot-atom $\text{slot}(S, X_1, V_1)$ by applying the specificity and specialization principles on the terms $X_1$ and $V_1$ simultaneously.

\[ \text{umap}(S, X, V) \leftarrow \text{slot}(S, X_1, V_1), \text{not redundant}(S, X, V), \text{unify}_{\text{with}}(X, X_1), \text{unify}_{\text{with}}(V, V_1). \] (45)

Given an underspecified knowledge base $KB = D(\sigma) \cup \Pi_I$, by $KB^*$ we denote the program $KB \cup \Pi_R$.

**Definition 3.** Let $KB$ be an underspecified knowledge base over $\sigma$ and $i$ be an individual of the class $c$ in $KB$. The description of $i$ by $KB$ is defined as the set of atoms of the form $\text{umap}(S, X, V)$ such that $X$ and $V$ are ground terms of the form $i$ or $f(i)$, $f \in \mathcal{F}_s$, entailed by the program $KB^*$.

In the next examples, we illustrate the use of $KB^*$ in the different knowledge bases.

**Example 6.** Let us consider the KBs from Examples 2-3. We can check that $KB_0^*$ has an answer set $M$ that contains the umap-atoms in Fig. 2.

\[
\begin{align*}
\text{umap}(\text{has\_engine}, s_1, t_{\text{Eng}}(s_1)) \\
\text{umap}(\text{has\_tank}, s_1, t_{\text{Tank}}(s_1)) \\
\text{umap}(\text{connected}, t_{\text{Eng}}(s_1), t_{\text{Tank}}(s_1))
\end{align*}
\]

Figure 2: Description of $s_1$ by $KB_0$.

Each of the umap-atoms corresponds to one slot-atom (Ex. 2) because none of the terms in the set $\{s_1, t_{\text{Eng}}(s_1), t_{\text{Tank}}(s_1)\}$ has a more specific term. Hence, there exists no atom of the form $\text{more\_specific\_term}(V_1, V)$ in $M$. This implies that no atom of the form $\text{selected\_override}(V)$, $\text{selected\_to\_replace}(V)$, or $\text{redundant}(S, X, Y)$ exists in $M$. Under this condition, rule (44) identifies each term by itself. Rule (45) indicates that for each slot$(N, X, V)$ in $M$, we have that $\text{umap}(N, X, Y)$ belongs to $M$.

Now consider $KB_1$ from Ex. 3. We can show that the program $KB_1^*$ has a unique answer set $M$ that contains the umap-atoms in Fig. 3.

\[
\begin{align*}
\text{umap}(\text{has\_engine}, s_1, t_{\text{Eng}}(s_1)) \\
\text{umap}(\text{has\_tank}, s_1, t_{\text{Tank}}(s_1)) \\
\text{umap}(\text{connected}, t_{\text{Eng}}(s_1), t_{\text{Tank}}(s_1))
\end{align*}
\]

Figure 3: Description of $s_1$ by $KB_1$.

Let us detail the reasons for the presence of these atoms in $M$. We have that $\text{instance\_of}(s_1, \text{suburban})$ belongs to $M$ because it is a fact of $KB_1^*$. This, together with rule (16), implies that $\text{instance\_of}(s_1, \text{car})$ must belong to $M$. The latter, with the descriptive axioms in $D_1(\sigma_1)$ (Ex. 3), implies that $\text{instance\_of}(t_{\text{Eng}}(s_1), \text{engine})$ and $\text{instance\_of}(t_{\text{Tank}}(s_1), \text{tank})$ belong to $M$. Similarly, we can conclude that $\text{instance\_of}(t_{\text{Eng}}(s_1), \text{Engine})$ and $\text{instance\_of}(t_{\text{Tank}}(s_1), \text{Tank})$ belong to $M$ because $s_1$ is an instance of suburban.

It is easy to see that the atoms of the form $\text{instance\_of}(X, Y)$ in $M$ imply the presence of the five atoms of the form $\text{slot}(S, X, V)$ in Fig. 1 in $M$.

Since there exists no class that is a proper subclass of $\text{tEng}$ or $\text{tTank}$, the most specific class of $\text{tEng}(s_1)$ and $\text{tTank}(s_1)$ is $\text{tEngine}$ and $\text{tTank}$, respectively (by (27)-(28)); this implies that the most specific term of $\text{tEng}(s_1)$ (resp. $\text{tTank}(s_1)$) is itself (by (31)-(32)). On the other hand,
Eng(s₁) (resp. Tank(s₁)) has a more specific term, Eng(s₁) (resp. Tank(s₁)), as both are instances of engine (resp. tank), but the former is more specific than the latter. This results in the following atoms are in M:

\[
\text{most\_specific\_term(Eng(s₁), Eng(s₁)) and} \\
\text{most\_specific\_term(Tank(s₁), Tank(s₁))}
\]

Since none of the terms Eng(s₁), Tank(s₁), and Tank(s₁) appears more than once in atoms of the form most\_specific\_term, rules (36)-(41) imply that pick\_more\_specific(Eng(s₁), Eng(s₁)) and pick\_more\_specific(Tank(s₁), Tank(s₁)) belong to M. So, unify\_with(Eng(s₁), Eng(s₁)) belongs to M because of (40). Similarly, we can show that unify\_with(Tank(s₁), Tank(s₁)) belongs to M. This implies that the application of rule (45) yields the following conclusions:

- Its application on slot(has\_engine, s₁, Eng(s₁)) and slot(has\_engine, s₁, Eng(s₁)) results in the single atom umap(has\_engine, s₁, Eng(s₁));
- Its application on slot(has\_tank, s₁, Tank(s₁)) and slot(has\_tank, s₁, Tank(s₁)) creates umap(has\_tank, s₁, Tank(s₁));
- Its application on slot(connected, Eng(s₁), Tank(s₁)) results in the atom umap(connected, Eng(s₁), Tank(s₁)).

We observe that the principle (P2) might create different specializations of a slot and thus different sets of umap-atoms for an individual. It is easy to see that for \(D_2(σ_2)\) (Ex. 4) and \(KB_2 = D_2(σ_2) \cup Π_I, KB_2^r\) has two answer sets, each corresponding to one outcome discussed in Ex. 5.

4.3 Properties of the Implementation

We now discuss some properties of the program developed in the previous subsection. Proofs of theorems are deferred to Appendix 1. In general, we say that a program P satisfies a principle (Pi) \((i = 1, \ldots, 4)\) if none of the answer sets of P violates (P1).

**Theorem 1.** For a basic domain description \(D(σ)\) over the signature \(σ\), the program \(KB^r = D(σ) \cup Π_I \cup Π_R\) satisfies the principles (P1)-(P4).

To discuss the next property, we need an additional definition. Given a domain description \(D(σ)\), a slot s is deterministic at a class c in \(D(σ)\) if

- there exists at most one Skolem function \(f_c\) such that slot(s, X, f_c(X)) appears in an axiom of the form (10) in \(D(σ)\) whose right side is instance\_of(X, c); and
- there exists at most one pair of Skolem functions \(f_1, f_2\) such that slot(s, f_1(X), f_2(X)) appears in \(D(σ)\) whose right side is instance\_of(X, c).
A slot \( s \) is \textit{deterministic} if it is deterministic at every class in \( D(\sigma) \). Otherwise, \( s \) is \textit{nondeterministic}. We should emphasize that the fact that a slot \( s \) is deterministic does not imply that the slot is associated with a single value as commonly called \textit{single-valued} slot in the literature. For example, the slot \textit{has engine} of \( D_1(\sigma_1) \) (Ex. 3) is indeed a deterministic slot since for each class appearing in \( D_1(\sigma_1) \), we can check that the above conditions are satisfied.

A domain description \( D(\sigma) \) is deterministic if all slots in \( D(\sigma) \) are deterministic. A knowledge base \( KB \) is deterministic if its underlying domain description is deterministic. We can prove the following theorem.

**Theorem 2.** For a basic and deterministic domain description \( D(\sigma) \), the program \( KB^r = D(\sigma) \cup \Pi_I \cup \Pi_R \) has a unique answer set.

## 5 Related Work and Discussions

We first comment on our choice of using ASP for this work as opposed to a description logic formalism. Recall that the axiom (1) is the most frequently occurring axiom pattern in KM prototypes. The axiom (1) violates the tree model property. The axioms of this form can only be represented in description graphs, but that requires separating the slots that participate in such graphs from the slots in the rest of the KB [17]. For this reason, we chose to first do this formalization using ASP. We do, however, believe that it is possible to express aspects of reasoning in an underspecified knowledge base in a description logic framework, and we are investigating that in our current research.

Our formalization of umap-atoms is inspired by the heuristic unification implemented in the KM system [5]. There are several differences between unification mapping as specified in this paper and the heuristic unification used in KM. First, KM’s approach is procedural and is explained mostly using examples while our formalization is declarative. Second, KM computes only one possible unification while our approach computes all possible unifications. A single default choice for unification suffices in many practical situations, but in situations where the default choice goes wrong, it is important to give the user an option of choosing amongst different alternatives. Finally, the heuristic unification in KM is destructive, i.e., when two individuals are unified, one is replaced with the other in the KB. In contrast, our approach is non-destructive. The unification decisions made by UMAP are truly non-monotonic in the sense that if additional information is added to the KB, the individuals that were unified in the initial version of the KB may no longer be unified in the new version of the KB. Due to destructive nature of unification in KM, it is not capable of such non-monotonic behavior. KM allows a user to make explicit unification assertions in the KB. We have not yet considered that aspect of heuristic unification in our formalization.

We note that even though an ability to write class definitions in a modular manner is a desirable property, it is still useful to provide a facility when a change in a class could be automatically reflected in its subclasses. The SHAKEN and AURA systems have supported a knowledge propagation mechanism for this purpose. The current paper ignores the aspect of updating an under-specified knowledge base.

### 5.1 Dealing with Multiple Inheritance

Multiple inheritance occurs when two values of a slot at a class are represented by Skolem functions defined in two different classes.
Example 7. Consider the following statements about cars: (i) A powerful car is a car that has an engine with lots of power; (ii) A big car is a car that has a large engine; (iii) Suburbs are powerful cars and are big cars; and (iv) $s_1$ is a suburban. This information can be easily encoded as a domain description over the signature $\sigma_{\text{car}}$ containing $C_i = \{ s_1 \}$; $C_c = \{ \text{engine}, \text{big car}, \text{powerful car}, \text{suburban} \}$; $C_o = \{ \text{large}, \text{lots} \}$; $C_s = \{ \text{has engine}, \text{size}, \text{power} \}$; and $C_f = \{ \_\text{Eng}1, \_\text{Eng}2, \_\text{size}, \_\text{power} \}$. For brevity, we omit the class, instance, constant, and subclass axioms of $D(\sigma_{\text{car}})$. To state that a big car has a large engine, we use\(^3\)

\[
3 \begin{cases}
\text{instance_of(}_\text{Eng}1(X),\text{engine}), \\
\text{slot(size,}_\text{Eng}1(X),\text{large}), \\
\text{slot(has engine, X,}_\text{Eng}1(X))
\end{cases} \leftarrow \text{instance_of}(X,\text{big car}). \quad (46)
\]

That powerful car has a powerful engine is encoded as

\[
3 \begin{cases}
\text{slot(power,}_\text{Eng}2(X),\text{lots}) \\
\text{slot(has engine, X,}_\text{Eng}2(X)) \\
\text{instance_of(}_\text{Eng}2(X),\text{engine})
\end{cases} \leftarrow \text{instance_of}(X,\text{powerful car}) \quad (47)
\]

We can check that $D(\sigma_{\text{car}}) \cup \Pi_I \cup \Pi_R$ has a single answer set containing $\text{umap(has engine, s}_1,\_\text{Eng}1(s}_1))$, $\text{umap(size,}_\text{Eng}1(s}_1,\text{large})$, $\text{umap(power,}_\text{Eng}2(s}_1,\text{lots})$, $\text{umap(has engine, s}_1,\_\text{Eng}2(s}_1))$. \(\Box\)

The answer given by $\text{KB(}\sigma_{\text{car}})^r$ is only partially satisfactory in that it indicates that $s_1$ has an engine that is large and an engine that is powerful. It is not fully satisfactory since, intuitively, $\_\text{Eng}1(s}_1$ and $\_\text{Eng}2(s}_1$ should have been unified. Observe that because there is no subclass relation between $\text{big car}$ and $\text{powerful car}$, there is no preference among the slot specification for $\text{has engine}$ in (46) and (47). Nevertheless, it is intuitive that they should be unified. When should this unification be executed? To answer this question, we observe that a slightly different situation occurs in Ex. 4 where the two slot-atoms should not be unified. We observe that the specifications of the $\text{has engine}$ slot in Ex. 4 have the same domain while those in Ex. 7 come from unrelated domains. So, we can resolve the coreferential problem by (i) determining the set $S(\nu, s, c)$ of coreferential slot-atoms given each triple of a term $\nu$, a class $c$, and a slot $s$; and (ii)

\(^3\)For brevity, we combine two descriptive axioms into one.
unify the set $S(v, s, c)$. The first task can be realized using the following ASP rule:

$$
\text{coreference}(X, Y, S, C) \leftarrow \text{compatible}(X, Y), X \neq Y,
$$

$$
\text{term}_{\text{value}}(X, S, C), \text{term}_{\text{value}}(Y, S, C),
$$

$$
\text{not has more specific term}(X),
$$

$$
\text{not has more specific term}(Y),
$$

$$
\text{domain}(X, C_1), \text{domain}(Y, C_2), C_1 \neq C_2,
$$

$$
\text{notsubclass of}(C_1, C_2), \text{notsubclass of}(C_2, C_1).
$$

The above rule states that two compatible terms $X$ and $Y$, which are possible values of the term $S$ at class $C$, are coreferential if their domains are unrelated and neither has a more specific term. The rules (49)-(55) are similar to the unification rules (36)–(44) to accomplish task (ii).

5.2 Dealing with Constraints

We will now develop a set of rules for dealing with cardinality constraints on the set of slot values. First, we extend our domain description language to allow the representation of cardinality constraints on a set of slot values as follows. A constraint axiom is of the following form:

$$
\text{constraint}(s, c, l, u)
$$

where $s \in C_s$, $c \in C_c$, and $l$ and $u$ are integers with $1 \leq l \leq u$. Intuitively, this constraint states that slot $s$ has at least $l$ and at most $u$ values at class $c$. We will next discuss a set of rules for enforcing the constraints of the form (56). First, we need to guarantee that the set $\{\text{umap}(s, X, Y) \mid X \text{ is an instance of } c\}$ has at least $l$ elements. This can be achieved by the following constraint:

$$
\left\{ \text{pick to unify}(X, Y, S, C) : \text{coreference}(X, Y, S, C) \right\} 1 \leftarrow \text{term}(X).
$$

$$
\text{selected to use}(X) \leftarrow \text{pick to unify}(X, Y, S, C).
$$

$$
\text{selected to replace}(Y) \leftarrow \text{selected to use}(X), \text{coreference}(X, Y, S, C).
$$

$$
\text{selected to replace}(Y) \leftarrow \text{pick to unify}(X, Y, S, C), \text{not selected to replace}(X), \text{not selected to replace}(Y).
$$

$$
\text{not selected to replace}(X) \leftarrow \text{coreference}(X, Y, S, C), \text{selected to use}(X), \text{selected to use}(Y).
$$

$$
\text{identify with}(X, Y) \leftarrow \text{selected to use}(X), \text{coreference}(X, Y, S, C).
$$

The above rule states that two compatible terms $X$ and $Y$, which are possible values of the term $S$ at class $C$, are coreferential if their domains are unrelated and neither has a more specific term. The rules (49)-(55) are similar to the unification rules (36)–(44) to accomplish task (ii).
We will need to add rules, similar to rules (36)–(44), to the program to make sure that, for each constraint \((s,c,l,u)\), if the set \(\{\text{slot}(s,X,Y) : \text{instance}(X,c)\}\) has more than \(u\) elements then unification needs to be executed. Furthermore, we also need to propagate this constraint to the subclasses of \(c\). The rules for this purpose are:

\[
\begin{align*}
\text{constraint}(S,C_1,L,U) & \leftarrow \text{constraint}(S,C,L,U), \text{subclass}(C_1,C). \quad (59) \\
\text{need\_unification}(S,C) & \leftarrow U + 1\{\text{slot}(S,X,Y) : \text{instance}(X,C)\}, \quad (60) \\
\text{constraint}(S,C,L,U) & .
\end{align*}
\]

\[
\begin{align*}
1\{\text{pick\_for\_constraint}(S,X,Y), \\
\text{not\_pick\_for\_constraint}(S,X,Y)\} & \leftarrow \text{need\_unification}(S,C), \text{instance}(X,C), \quad (61) \\
\text{slot}(S,X,Y), \text{not\_redundant}(S,X,Y) & .
\end{align*}
\]

\[
\begin{align*}
1\{\text{overridden}(Y_1,Y)\} & \leftarrow \text{pick\_for\_constraint}(S,X,Y), \\
\text{need\_unification}(S,C), \text{instance}(X,C), \\
\text{not\_pick\_for\_constraint}(S,X,Y_1), Y \neq Y_1 & .
\end{align*}
\]

\[
\begin{align*}
\text{selected\_to\_replace}(Y) & \leftarrow \text{pick\_for\_constraint}(S,X,Y). \quad (63)
\end{align*}
\]

The first rule propagates the constraint to the subclasses. The second rule identifies the pairs of slots and classes whose cardinality constraint is violated. Rule (61) is a choice rule that decides whether or not a slot value should be unified with some other value in order to maintain the constraint. Rule (62) indicates that if some slot value has been selected to be unified with some other value, then this should be overridden by some value. Rule (63) does the housekeeping job, indicating that the value \(Y\) has been selected to be replaced by some other value.

6 Conclusions and Future Work

In this paper, we considered the problem of reasoning in an under-specified knowledge base. Specifically, we considered two forms of underspecification: some of the cardinality constraints are omitted from the KB and some values are specialized across a class hierarchy but the explicit references to which values are specialized are omitted. Such underspecification is very useful in achieving modularity in a large complex KB. We presented an approach called UMAP or unification mapping to do inheritance reasoning in such an under-specified KB. UMAP is inspired by a similar reasoning mechanism called heuristic unification that is implemented in KM and has proven to be empirically useful in enabling knowledge base construction by biologists with little background in formal knowledge representation. While we have used ASP as a formal framework to present our approach, we believe that the basic ideas are general and applicable to other reasoning frameworks.

Our immediate goal in the near future is to do the performance evaluation of the proposed framework using the biology knowledge base developed as part of Project Halo [11]. Our focus will be on using the program in answering three types of questions: (i) What is a \(X\)? The set of UMAP-atoms represents a complete description about an individual and thus can serve as the answer for this question; (ii) What are the relationships between \(X\) and \(Y\)? In this type of question, we will focus on relationships that can be described by paths connecting two individuals \(X\) and \(Y\). We expect that the efficient solvers that have been developed for ASP will help us compute relationships that cannot be computed by the current KM system. (iii) A third possible class
of questions involves process interruption reasoning. Specifically, we plan to enhance the current formalization to include a modular action language similar to what has been done in [13].

Acknowledgment: This work has been funded by Vulcan Inc. and SRI International. This work has greatly benefited from substantive discussions with Pete Clark, Richard Fikes, Benjamin Grosof, Michael Kifer and Michael Wessel. We are grateful to them for their input.

7 Appendix 1: Proofs

Theorem 1. For a basic domain description $D(\sigma)$ over the signature $\sigma$, the program $KB^r = D(\sigma) \cup \Pi_I \cup \Pi_R$ satisfies the principles (P1)-(P4).

Proof. Let $M$ be an answer set of $KB^r$. We will show that $M$ satisfies the (P1)-(P4).

1. (P1): Observe that a term $x$ is unified with a term $y, x \neq y$, iff one of the following conditions is satisfied:
   
   (a) There exists some $pick\_more\_specific(x, y, s, c) \in M$ (rule: (42)) which implies that which $x$ is more specific than $y$.
   
   (b) $replace(x, y) \in M$ and $selected\_to\_replace(x) \notin M$ (rule (43)). This implies that there exists no $replace(u, x) \in M$ (because of (35)). Furthermore, we can show that if $replace(x, y) \in M$ then there exist no $s$ and $c$ such that $most\_specific\_term(y, x, s, c) \in M$. That means that if $y$ is not preferred to $x$.

   The above shows that $M$ does not violate (P1).

2. (P2) Let $s$ be a slot and $c$ be a class. The rule (40) guarantees that a term $x$, which is a slot value of $s$ at $c$ (because $term\_value(s, s, c) \in M$) cannot be used to override two terms $y$ and $z$ with $y \neq z$. The rule (41) indicates that for each $most\_specific\_term(x, y, s, c) \in M$, either $override(y, s, c) \in M$ which indicates that $y$ has been overridden by some term; or $used\_for\_override(x, s, c) \in M$ which states that $x$ has been used to override some term. This shows that $M$ does not violate (P2).

3. (P3) Let $s$ be a slot with multiple specifications $slot(s, x_1, c_1), slot(s, x_2, c_2), \ldots, slot(s, x_n, c_n)$, such that $subclass\_of(c_i, c_{i+1})$ for $1 \leq i \leq n - 1$. It is easy to see that $replace(x_{i+1}, x_i) \in M$ for $1 \leq i \leq n - 1$ (because (33)-(35)). Furthermore, the specification $slot(s, x_n, c_n)$ is the most specific one among the specifications of $s$. This also shows that $M$ satisfies (P3).

4. (P4) This is trivial because of the rule (45).

\[\square\]

Theorem 2. For a basic and deterministic knowledge base $KB = D(\sigma) \cup \Pi_I$ over the signature $\sigma$, the program $KB^r$ has an unique answer set.

Proof. We will use the splitting sequence theorem [15] to prove this theorem. We define the following set of literals in $KB^r$: 

\[\square\]
- $L_1$ is the set of atoms of the form `class(X)`, `constant(X)`, `instance_of(X,Y)`, `subclass_of(X,Y)`, and `slot(X,Y,Z)`;

- $L_2$ is the set of atoms of the form `compatible(X,Y)`, `range(X,Y)`, `domain(X,C)`, `term(X)`, `term_value(X,S,C)`, and `instance_by_inheritance(X,C)`;

- $L_3$ is the set of atoms of the form `most_specific_class(X,C)`;

- $L_4$ is the set of atoms of the form `more_specific_term(X,Y)`;

- $L_5$ is the set of atoms of the form `most_specific_term(X,Y,S,C)`;

- $L_6$ is the set of atoms of the form `replace(X)`, `selected_to_replace(X)`, or `redundant(S,X,Y)`;

- $L_7 = \text{lit}(KB^r) \setminus \bigcup_{i=1}^6 L_i$ where `\text{lit}(KB^r)` is the set of all atoms occurring in $KB^r$.

We can check that $X_i = \bigcup_{i=1}^6 L_i$ for $i = 1, \ldots, 6$ is a splitting sequence of $KB^r$.

Observe that $X_1$ is the set of literals of the program $D(\sigma) \cup \Pi_f$, which is the bottom of $KB^r$ relative to $X_1$, denoted by $b_{X_1}(KB^r)$. Because $b_{X_1}(KB^r)$ is a positive program, it has a unique answer set. Let us denote this answer set by $S_0$.

The partial evaluation of $KB^r$ by $S_0$ with respect to $X_2$, denoted by $e_{X_2}(b_{X_2}(KB^r) \setminus b_{X_1}(KB^r), S_0)$, is a set $S_1$ of ground atoms of the form `compatible(X,Y)`, `range(X,Y)`, `term_value(X,S,C)`, `term(X)`, and `instance_by_inheritance(X,C)` satisfying the rules defining them. For instance, for each constant $c \in C_0$ such that `constant(c) \in M`, `compatible(c,c)` belongs to $S_1$; for every instance $a$ of a class $c$ and two different Skolem functions $f, f' \in F_s$, if `instance_of(f(a),c)` and `instance_of(f'(a),c)` belong to $M$, then `compatible(f(a),f'(a))` belongs to $S_1$; etc. Clearly, $S_1$ is the unique answer set of $e_{X_2}(b_{X_2}(KB^r) \setminus b_{X_1}(KB^r), S_0)$.

The partial evaluation of $KB^r$ by $S_0 \cup S_1$ with respect to $X_3$, denoted by $e_{X_3}(b_{X_3}(KB^r) \setminus b_{X_1}(KB^r), S_0 \cup S_1)$, is a set $S_2$ of ground atoms of the form `most_specific_class(X,C)` such that if `most_specific_class(x,c) \in S_2` then `instance_of(x,c) \in S_0` and `instance_by_inheritance(x,c) \not \in S_1`. Furthermore, it is easy to see that, due to the rules (15)-(16), we have that

(*) for term $x$, `term(x) \in S_1`, there exists a unique $c$ such that `class(c) \in S_0` and `most_specific_class(x,c) \in S_2`.

Similar arguments with respect to the splitting set $X_4$ allow us to conclude that the partial evaluation of $KB^r$ by $S_0 \cup S_1 \cup S_2$ with respect to $X_4$ has a unique answer set $S_3$ consisting of atoms of the form `more_specific_term(X,Y)`.

The partial evaluation of $KB^r$ by $S_0 \cup S_1 \cup S_2 \cup S_3$ with respect to $X_5$ has a unique answer set $S_4$ consisting of atoms of the form `most_specific_term(X,Y,S,C)`. Due to the determinicity of $D(\sigma)$, we can conclude the following

(**) for each tuple $(x,s,c)$ satisfying that `term(x) \in S_1`, `class(c) \in S_0`, and $s$ appears in some atom of the form `slot(s,t,t')` in $S_0$, there exists at most one term $y$ such that `term(y) \in S_1` and `most_specific_term(y,x,s,c) \in S_4`. 

21
(*** for each tuple \((y, s, c)\) satisfying that \(\text{term}(y) \in S_1\), \(\text{class}(c) \in S_0\), and \(s\) appears in some atom of the form \(\text{slot}(s, t, t')\) in \(S_0\), there exists at most one term \(x\) such that \(\text{term}(x) \in S_1\) and \(\text{most specific term}(y, x, s, c) \in S_4\).

The partial evaluation of \(KB^r\) by \(\bigcup_{i=0}^{\alpha} S_i\) with respect to \(X_5\) is a positive program and contains the following rules:

\[
2 \{\text{replace}(Y_1, Y), \text{redundant}(S, X, Y)\} \rightarrow \text{replace}(Y_2, Y, \text{replace}(Y_2, Y_1), \text{replace}(Y_1, Y)).
\]

\[
\text{selected to replace}(Y) \rightarrow \text{replace}(Y_1, Y).
\]

Obviously, this program has a unique answer set \(S_5\).

The partial evaluation of \(KB^r\) by \(S = \bigcup_{i=0}^{\beta} S_i\) with respect to \(X_6\) denoted by \(P\), contains the following rules:

\[
0\left\{\begin{array}{ll}
\text{pick more specific}(v_1, v, s, c) & \text{if term}(v) \in S \\
\text{most specific term}(v_1, v, s, c) & \text{override}(v, s, c) \rightarrow \text{pick more specific}(v_1, v, s, c).
\end{array}\right.
\]

\[
\text{override}(v, s, c) \rightarrow \text{pick more specific}(v_1, v, s, c).
\]

\[
\text{used for override}(v_1, s, c) \rightarrow \text{pick more specific}(v_1, v, s, c).
\]

\[
\text{pick more specific}(z, x, s, c).
\]

\[
\text{pick more specific}(z, y, s, c).
\]

\[
\text{not override}(v, s, c),
\]

\[
\text{not used for override}(v_1, s, c).
\]

\[
\text{unify with}(x, y) \rightarrow \text{pick more specific}(x, y, s, c).
\]

\[
\text{unify with}(x, x) \rightarrow \text{not selected override}(x).
\]

\[
\text{unmap}(s, x, v) \rightarrow \text{unify with}(x, x_1),
\]

\[
\text{unify with}(v, v_1).
\]

where, in all of the above rules, \(s \in F_s\), \(c \in C_c\) such that \(\text{class}(c) \in M\) and \(s\) appears in one of the atoms of the form \(\text{slot}(s, x, y)\) in \(M\), \(x, y, v\) and \(v_1\) are terms appearing the second or third argument of atoms of the form \(\text{slot}(s, x, y)\) in \(M\).

It follows from the splitting theorem that each answer set of \(KB^r\) is of the form \(S \cup U\) where \(U\) is an answer set of the program \(P\). As such, to prove the theorem, it suffices to show that \(P\) has a unique answer set.

Let \(Q\) be the set of rules of the form (67) belonging to \(P\), \(X \subseteq S_4\), and

\[
Y(X) = \{\text{pick more specific}(v_1, v, s, c) \mid \text{most specific term}(v_1, v, s, c) \in X\}
\]

then the program \(P \setminus Q \cup Y(X)\) has at most one answer set. This is because \(Y(X)\) is a splitting set of the program \(P \setminus Q\) and the evaluation of \(P \setminus Q\) relative to \(Y(X)\) is a positive program.
Let us consider the case $X = S_4$. Due to (***) and (**), we have that there exists no most_specific_term$(v_1, v, s, c) \in S$ such that $v_1 \neq v_2$ and there exists no most_specific_term$(v_1, v', s, c) \in S$ such that $v \neq v'$. Thus, the constraints (71)-(72) are trivially satisfied. This means that $P \setminus Q \cup Y(S_4)$ has an answer set. Let us denote the answer set of this program with $Z$. By the splitting sequence theorem, we have that $U = Z \cup Y(S_4)$ is an answer set of $P$ and $S \cup U$ is an answer set of $KB^r$.

It remains to be shown that there exists no $X \subseteq S_4$ such that $X \neq S_4$ and $P \setminus Q \cup Y(X)$ has an answer set. Assume the contrary, there exists $X_0 \subseteq S_4$ such that $X_0 \neq S_4$ and $P \setminus Q \cup Y(X_0)$ has an answer set $Z_0$. Consider an atom most_specific_term$(v_1, v, s, c) \in S_4 \setminus X_0$. Because of (**), we have there exists no most_specific_term$(v_2, v, s, c) \in S_4$ such that $v_1 \neq v_2$. As such, the program $P \setminus Q \cup Y(X_0)$ contains no rule of the form (68) whose right hand side is satisfied and whose left hand side is $\text{override}(v, s, c)$, i.e., $\text{override}(v, s, c) \notin Z_0$. Likewise, we have that $\text{used_for_override}(v_1, s, c) \notin Z_0$ because there exists no other atom of the form most_specific_term$(v_1, v', s, c) \in S_4$ and $v \neq v'$ (due to (***)). The last two conclusions and the fact that most_specific_term$(v_1, v, s, c) \in S_4$ imply that the constraint (72) is violated. Thus, the program $P \setminus Q \cup Y(X_0)$ is inconsistent. So, we have proved that $X = S_4$ is the unique set such that $P \setminus Q \cup Y(X)$ is consistent. This concludes the proof of the theorem.

8 Appendix 2: Experimental Evaluation

This section includes some simple underspecified knowledge bases which have been used as case studies for the development of the principles and the ASP program for computing of the umap-atoms. To simplify the readings, we use a graphical representation, which is similar to a concept map in [11], to represent the domain description. The signatures are also implicitly given.

8.1 Case Study 1 (CS1)

The first part of CS1 is given in Fig. 4. Rectangles represent classes. Links between classes represent slots. The Skolem functions are of the form $Xn$ where $X$ is a class name.

![Figure 4: Case Study 1](image)

The graph in Fig. 4 represents the following domain description, which we will denote with $D_{cs1}$

```
class(classA).
class(classB).
class(classC).
```

23
instance_of(_classB2(X), classB) :- instance_of(X, classA).
instance_of(_classC3(X), classC) :- instance_of(X, classA).
slot(s1, X, _classB2(X)) :- instance_of(X, classA).
slot(s2, X, _classC3(X)) :- instance_of(X, classA).

For this case study, we investigate the following five sub-cases:

8.1.1 Sub-Case 1

The domain contains subclassSA, a subclass of classA, and two constants, a and sa which are individuals belonging to classA and subclassSA, correspondingly. This information is represented by the following graph

![Class SA](image)

Figure 5: Sub-Case 1

and encoded by the following program, denoted by $D_{s_1}$.

class(classSA).
subclass_of(classSA, classA).

instance_of(a, classA).
constant(a).

instance_of(sa, classSA).
constant(sa).

Let $KB_{cs1,1} = D_{cs1} \cup D_{s_1} \cup \Pi_I$. $KB_{cs1,1}^r$ has a unique answer set with the following $umap$-atoms:

$umap(s1, a, _classB2(a))$
$umap(s2, a, _classC3(a))$
$umap(s1, sa, _classB2(sa))$
$umap(s2, sa, _classC3(sa))$

Observe that a and sa are two different individuals and sa is a member of the class subclassSA which is a subclass of classA. As such, even though there is no descriptive axiom about elements in subclassSA, sa inherits all slots from the descriptive axioms about elements in classA.

Notice that for the computation of the answer set, we need to add the following domain-dependent rules:

object_of(_classB2(X), X):- constant(X).
object_of(_classC3(X), X):- constant(X).

codomain_class(_classB2(X), classB):- constant(X).
codomain_class(_classC3(X), classC):- constant(X).

The graph can be represented as a set of nodes and links and can be translated automatically into a domain description. The programs included in this section are generated in this way.
8.1.2 Sub-Case 2

In addition to the information in the Sub-Case 1, in this case, the domain also contains classSB, a subclass of classB, and one slot as given by the following graph

![Figure 6: Sub-Case 2](image)

The program $D_{s_2}$ is the union of $D_{s_1}$ and the set of the following rules:

```
class(classSB).
subclass_of(classSB, classB).

instance_of(_classSB5(X), classSB) :- instance_of(X, classSA).
slot(s1, X, _classSB5(X)) :- instance_of(X, classSA).
```

Let $KB_{cs1,2} = D_{cs1} \cup D_{s_2} \cup \Pi_I$. $KB_{cs1,2}^r$ has a unique answer set with the following $umap$-atoms related to $sa$:

```
umap(s1, sa, _classSB5(sa))
umap(s2, sa, _classC3(sa))
```

8.1.3 Sub-Case 3

In addition to the information in the Sub-Case 2, in this case, the domain also contains classSC, a subclass of classC, and two slots as given by the following graph

![Figure 7: Sub-Case 3](image)

The program $D_{s_3}$ is the union of $D_{s_2}$ and the set of the following rules:

```
class(classSC).
subclass_of(classSC, classC).

instance_of(_classSC6(X), classSC) :- instance_of(X, classSA).
slot(s2, X, _classSC6(X)) :- instance_of(X, classSA).
```

Let $KB_{cs1,3} = D_{cs1} \cup D_{s_3} \cup \Pi_I$. $KB_{cs1,3}^r$ has a unique answer set containing the following $umap$-atoms related to $sa$:

```
umap(s1, sa, _classSB5(sa))
umap(s2, sa, _classSC6(sa))
```
8.1.4 Sub-Case 4

In addition to the information in the Sub-Case 1, in this case, the domain also contains classD and a slot as given by the following graph

![Figure 8: Sub-Case 4](image)

The program $D_{s_4}$ is the union of $D_{s_1}$ and the set of the following rules:

```prolog
class(classD).

instance_of(_classD5(X), classD) :- instance_of(X, classSA).
slot(s1, X, _classD5(X)) :- instance_of(X, classSA).
```

Let $KB_{cs1,4} = D_{cs1} \cup D_{s_4} \cup \Pi_I$. $KB_{cs1,4}^r$ has a unique answer set containing the following $umap$-atoms related to $sa$:

- $umap(s1,sa,_classB2(sa))$
- $umap(s2,sa,_classC3(sa))$
- $umap(s1,sa,_classD5(sa))$

This case illustrates that our reasoning is additive with respect to slot values in different classes.

8.1.5 Sub-Case 5

In addition to the information in the Sub-Case 1, in this case, the domain also contains classD and classE and additional slots as given by the following graph

![Figure 9: Sub-Case 5](image)

The program $D_{s_5}$ is the union of $D_{s_1}$ and the set of the following rules:

```prolog
class(classD).
class(classE).

instance_of(_classD5(X), classD) :- instance_of(X, classSA).
slot(s3, X, _classD5(X)) :- instance_of(X, classSA).
instance_of(_classE6(X), classE) :- instance_of(X, classSA).
slot(s4, X, _classE6(X)) :- instance_of(X, classSA).
```

Let $KB_{cs1,5} = D_{cs1} \cup D_{s_5} \cup \Pi_I$. $KB_{cs1,5}^r$ has a unique answer set containing the following $umap$-atoms related to $sa$:


umap(s1,sa,_classB2(sa))
umap(s2,sa,_classC3(sa))
umap(s3,sa,_classD5(sa))
umap(s4,sa,_classE6(sa))

This case illustrates that our reasoning is additive with respect to slot values in different classes.

8.2 Case Study 2 (CS2)

The first part of CS2 is given in Fig. 10. The main different between this case and CS1 is the slot $s_3$ from classB to classC.

![Figure 10: Case 2](image)

The program, $D_{sc2}$, representing this graph is $D_{sc1}$ plus the following rule:

```prolog
slot(s3, _classB2(X), _classC3(X)) :- instance_of(X, classA).
```

We consider three sub-cases as in Figures 5-7.

8.2.1 Sub-Case 1

For $KB_{sc2,1} = D_{sc2} \cup D_{s_1} \cup \Pi_I$, $KB_{sc2,1}$ has one answer set containing the following $umap$-atoms related to $sa$:

- $umap(s1,sa,_classB2(sa))$
- $umap(s2,sa,_classC3(sa))$
- $umap(s3,_classB2(sa),_classC3(sa))$

8.2.2 Sub-Case 2

For $KB_{sc2,2} = D_{sc2} \cup D_{s_2} \cup \Pi_I$, $KB_{sc2,2}$ has one answer set containing the following $umap$-atoms related to $sa$:

- $umap(s1,sa,_classB5(sa))$
- $umap(s2,sa,_classC3(sa))$
- $umap(s3,_classB5(sa),_classC3(sa))$

8.2.3 Sub-Case 3

For $KB_{sc2,3} = D_{sc2} \cup D_{s_3} \cup \Pi_I$, $KB_{sc2,3}$ has one answer set containing the following $umap$-atoms related to $sa$:

- $umap(s1,sa,_classB5(sa))$
- $umap(s2,sa,_classC6(sa))$
- $umap(s3,_classB5(sa),_classC6(sa))$
8.3 Case Study 3 (CS3)

The first part of CS3 is given in Fig. 11.

Figure 11: Case 3

and encoded by the following program, denoted by $D_{sc3}$.

```prolog
class(classA).
class(classB).
class(classC).
class(classD).
subclass_of(classSA, classA).

instance_of(_classB2(X), classB) :- instance_of(X, classA).
instance_of(_classD3(X), classD) :- instance_of(X, classA).
instance_of(_classC4(X), classC) :- instance_of(X, classA).
slot(s1, X, _classB2(X)) :- instance_of(X, classA).
slot(s3, X, _classD3(X)) :- instance_of(X, classA).
slot(s2, _classB2(X), _classC4(X)) :- instance_of(X, classA).
slot(s4, _classD3(X), _classC4(X)) :- instance_of(X, classA).
```

8.3.1 Sub-Case 1

For $KB_{sc3,1} = D_{sc3} \cup D_{s1} \cup \Pi_I$, $KB_{sc3,1}^r$ has one answer set containing the following umap-atoms related to sa:

```prolog
umap(s1, sa, _classB2(sa))
umap(s3, sa, _classD3(sa))
umap(s2, _classB2(sa), _classC4(sa))
umap(s4, _classD3(sa), _classC4(sa))
```

8.3.2 Sub-Case 2

For $KB_{sc3,2} = D_{sc3} \cup D_{s2} \cup \Pi_I$, $KB_{sc3,2}^r$ has one answer set containing the following umap-atoms related to sa:

```prolog
umap(s1, sa, _classSB6(sa))
```
8.3.3 Sub-Case 3

The information for this sub-case is given in Fig. 12.

The program $D_{s3}^3$ is the set of the following rules:

```
subclass_of(classSA, classA).
subclass_of(classSB, classB).
subclass_of(classSD, classD).
instance_of(_classSB6(X), classSB) :- instance_of(X, classSA).
instance_of(_classSD7(X), classSD) :- instance_of(X, classSA).
slot(s1, X, _classSB6(X)) :- instance_of(X, classSA).
slot(s3, X, _classSD7(X)) :- instance_of(X, classSA).
```

Let $KB_{cs3,3} = D_{cs3} \cup D_{r3}^r \cup \Pi_I$. $KB_{cs3,3}$ has a unique answer set containing the following $umap$-atoms related to $sa$:

```
umap(s1, sa, _classSB6(sa))
umap(s3, sa, _classSD7(sa))
umap(s2, _classSB6(sa), _classC4(sa))
umap(s4, _classSD7(sa), _classC4(sa))
```

8.3.4 Sub-Case 4

The information for this sub-case is given in Fig. 13.

The program $D_{s4}^3$ is the set of the following rules:
class(classSA).
class(classSB).
class(classSD).
class(classSC).

subclass_of(classSA, classA).
subclass_of(classSB, classB).
subclass_of(classSD, classD).
subclass_of(classSC, classC).

instance_of(_classSB6(X), classSB) :- instance_of(X, classSA).
instance_of(_classSD7(X), classSD) :- instance_of(X, classSA).
instance_of(_classSC8(X), classSC) :- instance_of(X, classSA).
slot(s1, X, _classSB6(X)) :- instance_of(X, classSA).
slot(s3, X, _classSD7(X)) :- instance_of(X, classSA).
slot(s2, _classSB6(X), _classSC8(X)) :- instance_of(X, classSA).

Let $KB_{cs3,4} = D_{cs4} \cup D^r_{s4} \cup \Pi_I$. $KB^r_{cs3,4}$ has a unique answer set containing the following $umap$-atoms related to $sa$:

$umap(s1, sa, _classSB6(sa))$
$umap(s3, sa, _classSD7(sa))$
$umap(s2, _classSB6(sa), _classSC8(sa))$
$umap(s4, _classSD7(sa), _classSC8(sa))$

8.4 Case Study 4 (CS4)

The first part of CS4 is given in Fig. 14.

![Figure 14: Case 4](image_url)

and encoded by the following program, denoted by $D_{sc4}$.

class(classA).
class(classB).
class(classC).
class(classD).
instance_of(_classB2(X), classB) :- instance_of(X, classA).
instance_of(_classC3(X), classC) :- instance_of(X, classA).
instance_of(_classD4(X), classD) :- instance_of(X, classA).
slot(s1, X, _classB2(X)) :- instance_of(X, classA).
slot(s2, X, _classC3(X)) :- instance_of(X, classA).
slot(s3, _classD4(X), _classB2(X)) :- instance_of(X, classA).

8.4.1 Sub-Case 1

For $KB_{sc4,1} = D_{sc1} \cup D_{s1} \cup \Pi_f$, $KB_{sc4,1}^r$ has one answer set containing the following umap-atoms related to $sa$:

umap(s1,sa,_classB2(sa))
umap(s2,sa,_classC3(sa))
umap(s3,_classD4(sa),_classB2(sa))

8.4.2 Sub-Case 2

For $KB_{sc4,2} = D_{sc1} \cup D_{s2} \cup \Pi_f$, $KB_{sc4,2}^r$ has one answer set containing the following umap-atoms related to $sa$:

umap(s1,sa,_classSB6(sa))
umap(s2,sa,_classC3(sa))
umap(s3,_classD4(sa),_classSB6(sa))

8.5 Case Study 5 (CS5)

The domain description of CS5 is given in Fig. 15.

![Figure 15: Case 5](image)

and encoded by the following program, denoted by $D_{sc5}$.

class(classA).
class(classB).
instance_of(_classB2(X), classB) :- instance_of(X, classA).
instance_of(_classC3(X), classC) :- instance_of(X, classA).
instance_of(_classD4(X), classD) :- instance_of(X, classA).
instance_of(_classE5(X), classE) :- instance_of(X, classA).
slot(s1, X, _classB2(X)) :- instance_of(X, classA).
slot(s2, X, _classC3(X)) :- instance_of(X, classA).
slot(s3, _classD4(X), _classB2(X)) :- instance_of(X, classA).
slot(s4, _classE5(X), _classD4(X)) :- instance_of(X, classA).
instance_of(a, classA).
constant(a).

instance_of(sa, classSA).
constant(sa).

For $KB_{sc5} = D_{sc5} \cup \Pi_I$, $KB_{sc5}^r$ has one answer set containing the following $umap$-atoms related to $sa$:

$umap(s1, sa, _classB2(sa))$
$umap(s2, sa, _classC3(sa))$
$umap(s3, _classD4(sa), _classB2(sa))$
$umap(s4, _classE5(sa), _classD4(sa))$

8.6 Case Study 6 (CS6)

The domain description of CS6 is given in Fig. 16.

and encoded by the following program, denoted by $D_{sc6}$.

instance_of(_classB2(X), classB) :- instance_of(X, classA).
instance_of(_classC3(X), classC) :- instance_of(X, classA).
instance_of(_classD4(X), classD) :- instance_of(X, classA).
instance_of(_classE5(X), classE) :- instance_of(X, classA).
instance_of(_classF6(X), classF) :- instance_of(X, classA).
8.6.1 Sub-Case 1

For $KB_{sc6,1} = D_{sc6} \cup D_{s1} \cup \Pi_f$, $KB_{sc6,1}^r$ has one answer set containing the following $umap$-atoms related to $sa$:

- $umap(s1, sa, _\text{classB2}(sa))$
- $umap(s2, _\text{classB2}(sa), _\text{classC3}(sa))$
- $umap(s3, sa, _\text{classD4}(sa))$
- $umap(s4, _\text{classD4}(sa), _\text{classC3}(sa))$
- $umap(s5, sa, _\text{classE5}(sa))$
- $umap(s6, _\text{classE5}(sa), _\text{classF6}(sa))$
- $umap(s7, _\text{classF6}(sa), _\text{classC3}(sa))$

8.6.2 Sub-Case 2

For $KB_{sc6,2} = D_{sc6} \cup D_{s2} \cup \Pi_f$, $KB_{sc6,2}^r$ has one answer set containing the following $umap$-atoms related to $sa$:

- $umap(s1, sa, _\text{classSB8}(sa))$
umap(s2, _classSB8(sa), _classC3(sa))
umap(s3, sa, _classD4(sa))
umap(s4, _classD4(sa), _classC3(sa))
umap(s5, sa, _classE5(sa))
umap(s6, _classE5(sa), _classF6(sa))
umap(s7, _classF6(sa), _classC3(sa))

8.6.3 Sub-Case 3

The information in Fig. 17 is encoded by the following rules, $D_{s3}^6$

class(classSA).
class(classSB).
class(classSC).

subclass_of(classSA, classA).
subclass_of(classSB, classB).
subclass_of(classSC, classC).

instance_of(_classSB8(X), classSB) :- instance_of(X, classSA).
instance_of(_classSC9(X), classSC) :- instance_of(X, classSA).
slot(s1, X, _classSB8(X)) :- instance_of(X, classSA).
slot(s2, _classSB8(X), _classSC9(X)) :- instance_of(X, classSA).

For $KB_{sc6,3} = D_{sc6} \cup D_{s3}^6 \cup \Pi_f$, $KB_{sc6,3}^r$ has one answer set containing the following umap-atoms related to sa:

umap(s1, sa, _classSB8(sa))
umap(s2, _classSB8(sa), _classSC9(sa))
umap(s3, sa, _classD4(sa))
umap(s4, _classD4(sa), _classSC9(sa))
umap(s5, sa, _classE5(sa))
umap(s6, _classE5(sa), _classF6(sa))
umap(s7, _classF6(sa), _classC3(sa))

8.7 Case Study 7 (CS7)

The domain description of CS7 is given in Fig. 18.

and encoded by the following program, denoted by $D_{sc7}$.  

Figure 17: Sub-Case 3
8.7.1 Sub-Case 1

For $KB_{sc7,1} = D_{sc7} \cup D_{s1} \cup \Pi_I$, $KB'_{sc7,1}$ has one answer set containing the following $umap$-atoms related to $sa$:

- $umap(s1,sa,_\text{classB2}(sa))$
- $umap(s2, _\text{classB2}(sa), _\text{classC3}(sa))$
- $umap(s3, sa, _\text{classD4}(sa))$
- $umap(s4, _\text{classD4}(sa), _\text{classC3}(sa))$
- $umap(s5, sa, _\text{classE5}(sa))$
- $umap(s6, _\text{classE5}(sa), _\text{classF6}(sa))$
- $umap(s7, _\text{classF6}(sa), _\text{classC3}(sa))$
- $umap(s7, _\text{classG7}(sa), _\text{classC3}(sa))$

8.7.2 Sub-Case 2

For $KB_{sc7,2} = D_{sc7} \cup D_{s2} \cup \Pi_I$, $KB'_{sc7,2}$ has one answer set containing the following $umap$-atoms related to $sa$:

- $umap(s1,sa, _\text{classB9}(sa))$
- $umap(s2, _\text{classB9}(sa), _\text{classC3}(sa))$
- $umap(s3, sa, _\text{classD4}(sa))$
- $umap(s4, _\text{classD4}(sa), _\text{classC3}(sa))$
- $umap(s5, sa, _\text{classE5}(sa))$
8.7.3 Sub-Case 3
For $KB_{sc7,3} = D_{sc7} \cup D_{sa}^6 \cup \Pi_I$, $KB_{sc7,3}'$ has one answer set containing the following $umap$-atoms related to $sa$:

\begin{align*}
umap(s_6,\_classE5(sa),\_classF6(sa)) \\
umap(s_7,\_classF6(sa),\_classC3(sa)) \\
umap(s_7,\_classG7(sa),\_classC3(sa))
\end{align*}

8.8 Case Study 8 (CS8)
The domain description of CS8 is given in Fig. 19. and encoded by the following program, denoted by $D_{sc8}$.

```
class(classA).
class(classB).
class(classC).

\text{instance} \_of( \_classB2(X), classB) :- \text{instance} \_of(X, classA).
\text{instance} \_of( \_classB3(X), classB) :- \text{instance} \_of(X, classA).
\text{instance} \_of( \_classB4(X), classB) :- \text{instance} \_of(X, classA).
\text{instance} \_of( \_classC5(X), classC) :- \text{instance} \_of(X, classA).
\text{slot}(s_1, X, \_classB2(X)) :- \text{instance} \_of(X, classA).
```
slot(s1, X, _classB3(X)) :- instance_of(X, classA).
slot(s1, X, _classB4(X)) :- instance_of(X, classA).
slot(s2, X, _classC5(X)) :- instance_of(X, classA).

8.8.1 Sub-Case 1

For $KB_{s8,1} = D_{s8} \cup D_{s1} \cup \Pi_I$, $KB_{s8,1}^r$ has one answer set containing the following umap-atoms related to $sa$:

umap(s1,sa,_classB2(sa))
umap(s1,sa,_classB3(sa))
umap(s1,sa,_classB4(sa))
umap(s2,sa,_classC5(sa))

8.8.2 Sub-Case 2

For $KB_{s8,2} = D_{s8} \cup D_{s2} \cup \Pi_I$, $KB_{s8,2}^r$ has three answer sets containing the following umap-atoms related to $sa$:

- **Answer 1:**
  
  umap(s1,sa,_classB7(sa))
  umap(s1,sa,_classB3(sa))
  umap(s1,sa,_classB4(sa))
  umap(s2,sa,_classC5(sa))

- **Answer 2:**
  
  umap(s1,sa,_classB2(sa))
  umap(s1,sa,_classB7(sa))
  umap(s1,sa,_classB4(sa))
  umap(s2,sa,_classC5(sa))

- **Answer 3:**
  
  umap(s1,sa,_classB2(sa))
  umap(s1,sa,_classB3(sa))
  umap(s1,sa,_classB7(sa))
  umap(s2,sa,_classC5(sa))

8.9 Case Study 9 (CS9)

The domain description of CS9 is given in Fig. 20 and the sub-case 2 in Fig. 6.

The encoding of the domain description, denoted by $D_{sc9}$, is:

class(classA).
class(classB).
class(classC).
Figure 20: Case 9

class(classSA).
class(classSB).
class(classSC).

instance_of(sa, classSA).
constant(sa).

instance_of(a, classA).
constant(a).

subclass_of(classSA, classA).
subclass_of(classSB, classB).
subclass_of(classSC, classC).

instance_of(_classB1(X), classB) :- instance_of(X, classA).
slot(s1, X, _classB1(X)) :- instance_of(X, classA).

instance_of(_classB3(X), classB) :- instance_of(X, classA).
slot(s1, X, _classB3(X)) :- instance_of(X, classA).

instance_of(_classB4(X), classB) :- instance_of(X, classA).
slot(s4, X, _classB4(X)) :- instance_of(X, classA).

instance_of(_classB5(X), classB) :- instance_of(X, classA).
slot(s4, X, _classB5(X)) :- instance_of(X, classA).

instance_of(_classC2(X), classC) :- instance_of(X, classA).
slot(s2, X, _classC2(X)) :- instance_of(X, classA).

instance_of(_classC3(X), classC) :- instance_of(X, classA).
slot(s2, X, _classC3(X)) :- instance_of(X, classA).

instance_of(_classB1(X), _classC2(X)) :- instance_of(X, classA).
slot(s3, _classB1(X), _classC2(X)) :- instance_of(X, classA).

instance_of(_classSB3(X), classB) :- instance_of(X, classSA).
slot(s1, X, _classSB3(X)) :- instance_of(X, classSA).
instance_of( _classSC4(X), classSC) :- instance_of(X, classSA).
slot(s2, X, _classSC4(X)) :- instance_of(X, classSA).

For $KB_{sc9} = D_{sc9} \cup \Pi_I$, $KB_{sc9}^r$ has four answer sets containing the following $umap$-atoms related to $sa$:

- **Answer 1:**
  
  \[
  \text{umap}(s1, sa, _\text{classSB3}(sa)) \\
  \text{umap}(s1, sa, _\text{classB3}(sa)) \\
  \text{umap}(s4, sa, _\text{classB4}(sa)) \\
  \text{umap}(s4, sa, _\text{classB5}(sa)) \\
  \text{umap}(s2, sa, _\text{classSC4}(sa)) \\
  \text{umap}(s2, sa, _\text{classC3}(sa)) \\
  \text{umap}(s3, _\text{classSB3}(sa), _\text{classSC4}(sa))
  \]

- **Answer 2:**
  
  \[
  \text{umap}(s1, sa, _\text{classB1}(sa)) \\
  \text{umap}(s1, sa, _\text{classSB3}(sa)) \\
  \text{umap}(s4, sa, _\text{classB4}(sa)) \\
  \text{umap}(s4, sa, _\text{classB5}(sa)) \\
  \text{umap}(s2, sa, _\text{classSC4}(sa)) \\
  \text{umap}(s2, sa, _\text{classC3}(sa)) \\
  \text{umap}(s3, _\text{classB1}(sa), _\text{classSC4}(sa))
  \]

- **Answer 3:**
  
  \[
  \text{umap}(s1, sa, _\text{classSB3}(sa)) \\
  \text{umap}(s1, sa, _\text{classB3}(sa)) \\
  \text{umap}(s4, sa, _\text{classB4}(sa)) \\
  \text{umap}(s4, sa, _\text{classB5}(sa)) \\
  \text{umap}(s2, sa, _\text{classC2}(sa)) \\
  \text{umap}(s2, sa, _\text{classSC4}(sa)) \\
  \text{umap}(s3, _\text{classSB3}(sa), _\text{classC2}(sa))
  \]

- **Answer 4:**
  
  \[
  \text{umap}(s1, sa, _\text{classB1}(sa)) \\
  \text{umap}(s1, sa, _\text{classSB3}(sa)) \\
  \text{umap}(s4, sa, _\text{classB4}(sa)) \\
  \text{umap}(s4, sa, _\text{classB5}(sa)) \\
  \text{umap}(s2, sa, _\text{classC2}(sa)) \\
  \text{umap}(s2, sa, _\text{classSC4}(sa)) \\
  \text{umap}(s3, _\text{classB1}(sa), _\text{classC2}(sa))
  \]
Figure 21: Case 10
8.10 Case Study 10 (CS10)

The domain description of CS10 is given in Fig. 21.

The encoding of the domain description, denoted by $D_{sc10}$, is:

class(classA).
class(classB).
class(classC).
class(classSA).
class(classSB).
class(classSC).

instance_of(sa, classSA).
canonical(sa).

instance_of(a, classA).
canonical(a).

subclass_of(classSA, classA).
subclass_of(classSB, classB).
subclass_of(classSC, classC).

instance_of( _classB1(X), classB) :- instance_of(X, classA).
slot(s1, X, _classB1(X)) :- instance_of(X, classA).
instance_of( _classB3(X), classB) :- instance_of(X, classA).
slot(s1, X, _classB3(X)) :- instance_of(X, classA).
instance_of( _classC2(X), classC) :- instance_of(X, classA).
slot(s2, X, _classC2(X)) :- instance_of(X, classA).
slot(s3, _classB1(X), _classC2(X)) :- instance_of(X, classA).
instance_of( _classB3(X), classB) :- instance_of(X, classSA).
slot(s1, X, _classB3(X)) :- instance_of(X, classSA).
instance_of( _classB4(X), classB) :- instance_of(X, classSA).
slot(s5, X, _classB4(X)) :- instance_of(X, classSA).
instance_of( _classSC4(X), classSC) :- instance_of(X, classSA).
slot(s2, X, _classSC4(X)) :- instance_of(X, classSA).

The program has two answer sets:

- **Answer set 1:**
  
  umap(s1,sa,_classSB3(sa))
  umap(s1,sa,_classSB3(sa))
  umap(s2,sa,_classSC4(sa))
  umap(s3,_classSB3(sa),_classSC4(sa))
  umap(s5,sa,_classSB4(sa))

- **Answer set 2:**
8.11 Case Study 11 (CS11)

The domain description of CS11 is given in Fig. 22.

The encoding of the domain description, denoted by $D_{sc11}$, is:

\begin{align*}
\text{class(classA).} \\
\text{class(classB).} \\
\text{class(classC).} \\
\text{class(classSA).} \\
\text{class(classSB).} \\
\text{class(classSC).} \\
\text{instance_of(sa, classSA).} \\
\text{constant(sa).} \\
\text{instance_of(a, classA).} \\
\text{constant(a).}
\end{align*}
subclass_of(classSA, classA).
subclass_of(classSB, classB).
subclass_of(classSC, classC).

definition

instance_of(_classB1(X), classB) :- instance_of(X, classA).
slot(s1, X, _classB1(X)) :- instance_of(X, classA).

instance_of(_classB3(X), classB) :- instance_of(X, classA).
slot(s1, X, _classB3(X)) :- instance_of(X, classA).

instance_of(_classC2(X), classC) :- instance_of(X, classA).
slot(s2, X, _classC2(X)) :- instance_of(X, classA).

instance_of(_classB1(X), _classC2(X)) :- instance_of(X, classA).

instance_of(_classB3(X), classB) :- instance_of(X, classA).

instance_of(_classSB3(X), classSB) :- instance_of(X, classSA).

instance_of(_classSB3(X)) :- instance_of(X, classSA).

instance_of(_classSB4(X), classSB) :- instance_of(X, classSA).

instance_of(_classSB4(X)) :- instance_of(X, classSA).

instance_of(_classSC4(X), classSC) :- instance_of(X, classSA).

instance_of(_classSC4(X)) :- instance_of(X, classSA).


The program has two answer sets:

- **Answer set 1:**
  
  - `umap(s1,sa,_classSB4(sa))`
  - `umap(s1,sa,_classSB3(sa))`
  - `umap(s2,sa,_classSC4(sa))`
  - `umap(s3,_classSB4(sa),_classSC4(sa))`

- **Answer set 2:**
  
  - `umap(s1,sa,_classSB4(sa))`
  - `umap(s1,sa,_classSB3(sa))`
  - `umap(s2,sa,_classSC4(sa))`
  - `umap(s3,_classSB3(sa),_classSC4(sa))`

8.12 Case Study 12 (CS12)

The domain description of CS12 is given in Fig. 23.

The encoding of the domain description, denoted by $D_{sc12}$, is:

class(classC).
class(classD).
class(classA).
class(classE).
class(classSA).
class(classSE).
class(classSC).
class(classSD).

instance_of(sc, classSC).
constant(sc).

instance_of(c, classC).
constant(c).

subclass_of(classSA, classA).
subclass_of(classSE, classE).
subclass_of(classSC, classC).
subclass_of(classSD, classD).

instance_of(_classA1(X), classA) :- instance_of(X, classC).
slot(s1, X, _classA1(X)) :- instance_of(X, classC).
instance_of(_classE1(X), classE) :- instance_of(X, classC).
slot(s2, _classA1(X), _classE1(X)) :- instance_of(X, classC).
instance_of(_classD2(X), classD) :- instance_of(X, classC).
slot(s3, _classA1(X), _classD2(X)) :- instance_of(X, classC).
instance_of(_classSA3(X), classSA) :- instance_of(X, classSC).
slot(s1, X, _classSA3(X)) :- instance_of(X, classSC).
instance_of(_classSE4(X), classSE) :- instance_of(X, classSC).
slot(s2, _classSA3(X), _classSE4(X)) :- instance_of(X, classSC).
instance_of(_classSD5(X), classSD) :- instance_of(X, classSC).
slot(s3, _classSA3(X), _classSD5(X)) :- instance_of(X, classSC).
This program has a unique answer set containing the following related to \( sc \):

\[ \text{umap}(s1,sc,\_\text{classSA3}(sc)) \]
\[ \text{umap}(s2,\_\text{classSA3}(sc),\_\text{classSE4}(sc)) \]
\[ \text{umap}(s3,\_\text{classSA3}(sc),\_\text{classSD5}(sc)) \]

8.13 Case Study 13 (CS13)

The domain description of CS13 is given in Fig. 24.

The encoding of the domain description, denoted by \( D_{sc13} \), is:

\[ \text{class(classA)}. \]
\[ \text{class(classSA)}. \]
\[ \text{class(classSSA)}. \]
\[ \text{class(classB)}. \]
\[ \text{class(classSB)}. \]
\[ \text{class(classSSB)}. \]

Figure 24: Case 13

The encoding of the domain description, denoted by \( D_{sc13} \), is:
instance_of(ssa, classSSA).
constant(ssa).

instance_of(sa, classSA).
constant(sa).

instance_of(a, classA).
constant(a).

subclass_of(classSA, classA).
subclass_of(classSSA, classSA).
subclass_of(classSSB, classSB).
subclass_of(classSB, classB).

\[
\text{instance_of( } _\text{classB1(}X\text{)} , \text{classB}) :- \text{instance_of(}X , \text{classA}) .
\]
\[
\text{slot(}s1 , X , _\text{classB1(}X\text{)}) :- \text{instance_of(}X , \text{classA}) .
\]
\[
\text{instance_of( } _\text{classB2(}X\text{)} , \text{classB}) :- \text{instance_of(}X , \text{classA}) .
\]
\[
\text{slot(}s2 , X , _\text{classB2(}X\text{)}) :- \text{instance_of(}X , \text{classA}) .
\]
\[
\text{instance_of( } _\text{classB3(}X\text{)} , \text{classB}) :- \text{instance_of(}X , \text{classA}) .
\]
\[
\text{slot(}s2 , X , _\text{classB3(}X\text{)}) :- \text{instance_of(}X , \text{classA}) .
\]
\[
\text{instance_of( } _\text{classSB1(}X\text{)} , \text{classSB}) :- \text{instance_of(}X , \text{classSA}) .
\]
\[
\text{slot(}s1 , X , _\text{classSB1(}X\text{)}) :- \text{instance_of(}X , \text{classSA}) .
\]
\[
\text{instance_of( } _\text{classSB2(}X\text{)} , \text{classSB}) :- \text{instance_of(}X , \text{classSA}) .
\]
\[
\text{slot(}s2 , X , _\text{classSB2(}X\text{)}) :- \text{instance_of(}X , \text{classSA}) .
\]
\[
\text{instance_of( } _\text{classSSB1(}X\text{)} , \text{classSSB}) :- \text{instance_of(}X , \text{classSSA}) .
\]
\[
\text{slot(}s1 , X , _\text{classSSB1(}X\text{)}) :- \text{instance_of(}X , \text{classSSA}) .
\]
\[
\text{instance_of( } _\text{classSSB2(}X\text{)} , \text{classSSB}) :- \text{instance_of(}X , \text{classSSA}) .
\]
\[
\text{slot(}s2 , X , _\text{classSSB2(}X\text{)}) :- \text{instance_of(}X , \text{classSSA}) .
\]

The program has four answer sets.

- **Answer:** 1\n  \[\text{umap(s1,ssa, classSSB1(ssa))}\]
  \[\text{umap(s2,ssa, classSSB2(ssa))}\]
  \[\text{umap(s2,ssa, classB3(ssa))}\]
  \[\text{umap(s1,sa, classSB1(sa))}\]
  \[\text{umap(s2,sa, classSB2(sa))}\]
  \[\text{umap(s2,sa, classB3(sa))}\]
  \[\text{umap(s1,a, classB1(a))}\]
  \[\text{umap(s2,a, classB2(a))}\]
  \[\text{umap(s2,a, classB3(a))}\]

- **Answer:** 2\n  \[\text{umap(s1,ssa, classSSB1(ssa))}\]
  \[\text{umap(s2,ssa, classSSB2(ssa))}\]
  \[\text{umap(s2,ssa, classB3(ssa))}\]
  \[\text{umap(s1,sa, classSB1(sa))}\]
8.14 Case Study 14 (CS14)

The domain description of CS14 is given in Fig. 25.

\[ D_{sc14} \]

\begin{center}
\begin{tikzpicture}
\node [class] at (0,0) (class_a) {Class A};
\node [class] at (2,0) (class_b) {Class B};
\node [class] at (1,-2) (class_c) {Class C};
\node [class] at (-1,-2) (class_sa) {Class SA};
\path
  (class_a) edge [->] node [above] {$S_1$} (class_b)
  (class_a) edge [->] node [left] {$S_2$} (class_c)
  (class_sa) edge [->] node [left] {$S_1$} (class_b);
\end{tikzpicture}
\end{center}

Figure 25: Case 14

The encoding of the domain description, denoted by $D_{sc14}$, is:
class(classA).
class(classB).
class(classC).
class(classSA).

subclass_of(classSA, classA).

constant(a).
instance_of(a, classA).

constant(sa).
instance_of(sa, classSA).

instance_of(_classB2(X), classB) :- instance_of(X, classA).
instance_of(_classB3(X), classB) :- instance_of(X, classSA).
instance_of(_classC3(X), classC) :- instance_of(X, classA).
slot(s1, X, _classB2(X)) :- instance_of(X, classA).
slot(s1, X, _classB3(X)) :- instance_of(X, classSA).
slot(s2, X, _classC3(X)) :- instance_of(X, classA).

domain_class(_classB2(X), classA) :- constant(X).
domain_class(_classB3(X), classSA) :- constant(X).
domain_class(_classC3(X), classA) :- constant(X).

The program has two answer sets.

- **Answer: 1**
  umap(s1, sa, _classB2(sa))
  umap(s1, a, _classB2(a))
  umap(s1, sa, _classB3(sa))
  umap(s2, sa, _classC3(sa))
  umap(s2, a, _classC3(a))

- **Answer: 2**
  umap(s1, a, _classB2(a))
  umap(s1, sa, _classB3(sa))
  umap(s2, sa, _classC3(sa))
  umap(s2, a, _classC3(a))

8.15 Case Study 15 (CS15)
We encode an example from the KM manual (Page 36). The comments are self-explanatory.

class(animal).
class(mammal).
class(dog).

class(head).
class(body).
class(leg).
class(tail).
class(skin).
class(fur).

subclass_of(mammal, animal).
subclass_of(dog, mammal).

%(every Animal has (parts ((a Head)%(a Body with (covering (*Skin)))))

instance_of(_body1(X), body):- instance_of(X, animal).
slot(has_part, X, _body1(X)):- instance_of(X, animal).
instance_of(_skin1(X), skin):- instance_of(X, animal).
slot(covering, _body1(X), _skin1(X)):- instance_of(X, animal).
instance_of(_head2(X), head):- instance_of(X, animal).
slot(has_part, X, _head2(X)):- instance_of(X, animal).

%(Mammal has (superclasses (Animal)))
%(every Mammal has (parts ((a Leg) (a Leg) (a Leg) (a Leg))))

instance_of(_leg1(X), leg):- instance_of(X, mammal).
slot(has_part, X, _leg1(X)):- instance_of(X, mammal).
instance_of(_leg2(X), leg):- instance_of(X, mammal).
slot(has_part, X, _leg2(X)):- instance_of(X, mammal).
instance_of(_leg3(X), leg):- instance_of(X, mammal).
slot(has_part, X, _leg3(X)):- instance_of(X, mammal).
instance_of(_leg4(X), leg):- instance_of(X, mammal).
slot(has_part, X, _leg4(X)):- instance_of(X, mammal).

%(Dog has (superclasses (Mammal)))
%(every Dog has (parts ((a Tail) (a Body with (covering (*Skin *Fur)))))

instance_of(_tail1(X), tail):- instance_of(X, dog).
slot(has_part, X, _tail1(X)):- instance_of(X, dog).
instance_of(_body2(X), tail):- instance_of(X, dog).
slot(has_part, X, _body2(X)):- instance_of(X, dog).
instance_of(_skin2(X), skin):- instance_of(X, dog).
slot(covering, _body2(X), _skin2(X)):- instance_of(X, dog).
instance_of(_fur3(X), fur):- instance_of(X, dog).
slot(covering, _body2(X), _fur3(X)):- instance_of(X, dog).

object_of(_body1(X), X):- constant(X).
object_of(_head2(X), X):- constant(X).
Without the use of the redundant axioms, we will get the naive answer for the dog. In other words, the program will have the following umap-atoms related to the dog $d$:

\[
\begin{align*}
\text{umap(has_part, } & \text{d, _body1(d))} \\
\text{umap(covers, } & \text{body1(d), _skin1(d))} \\
\text{umap(has_part, } & \text{d, _head2(d))} \\
\text{umap(has_part, } & \text{d, _leg1(d))} \\
\text{umap(has_part, } & \text{d, _leg2(d))} \\
\end{align*}
\]
With the redundant axioms, the program has two answer sets. The first one does not enforce the redundancy and is the same as the above. The second one is

\begin{verbatim}
umap(has_part,d,_head2(d))
umap(has_part,d,_leg1(d))
umap(has_part,d,_leg2(d))
umap(has_part,d,_leg3(d))
umap(has_part,d,_leg4(d))
umap(has_part,d,_tail1(d))
umap(covering,_body2(d),_skin2(d))
umap(covering,_body2(d),_fur3(d))
\end{verbatim}

References


