On Specialization Constraints over Complex Objects*

Minoru Ito
Department of Information and Computer Science
Faculty of Engineering Science
Osaka University
Toyonaka, Osaka 560, Japan

Grant E. Weddell and Neil Coburn
Department of Computer Science
University of Waterloo
Waterloo, Ontario, Canada, N2L 3G1

Technical Report CS-91-62

*This research was supported in part by the Ministry of Education, Science and Culture of Japan, by the Natural Sciences and Engineering Research Council of Canada, by Bell-Northern Research Ltd., Ottawa, and by ITRC: Information Technology Research Center.

Abstract

Most semantic data models and object-oriented data models allow entity and object classes to be organized according to a generalization taxonomy. In addition, range restrictions (or property typing) may be specified not only on properties associated with a given class, but also on properties inherited from superclasses. In this paper, we consider a more general form of *specialization constraint* in which range restrictions are associated with property value paths, instead of with the properties themselves. One consequence is that the constraints enable a form of *molecular abstraction*, in which the internals of more complicated objects can be defined in terms of a collection of more primitive classes.

We consider the problem for two models. The first imposes no constraints on class membership for an object beyond those implied by subclassing constraints. In this case, we present a sound and complete axiomatization for arbitrary specialization constraints, and efficient decision procedures for the corresponding membership problems.

The second model is more typical and requires that each object is created with respect to a particular class. Membership problems in this case are shown to be NP-hard, and NP-complete if class schema include a "bottom" class. We exhibit polynomial-time decision procedures when a bottom class does exist and antecedent specialization constraints satisfy a *bounded path length* condition.

We also consider a case concerning the second model in which class schema satisfy a *lower semi-lattice* condition. A sound and complete axiomatization for *well-formed* specialization constraints is presented, together with efficient decision procedures for the membership problem for well-formed constraints, and for determining if an arbitrary constraint is well-formed. We prove that the membership problem for arbitrary specialization constraints remains NP-complete, however, even for class schema satisfying the lower semi-lattice condition.

Key Words: complex objects, dependency theory, logical database design, object-oriented databases

Contents

1. Introduction

Most semantic or object-oriented data models assume that entities or objects have an identity separate from any of their parts, and allow users to define complex object types in which part values may be any other objects [1, 2, 9, 13, 14, 15]. Such types are usually called *classes*, and can be organized in a generalization taxonomy by allowing a class definition to mention at least one superclass—more than one if the data model supports so-called *multiple inheritance*. In addition, range restrictions (or property typing) may be specified, not only on properties associated with a given class, but also on properties inherited from superclasses. In this paper, we consider a more general abstraction, called *specialization constraints*, in which range restrictions are associated with *path descriptions*. Specialization constraints can be used to assert property typing, since one kind of path description is an individual property name. Since another kind of path description, denoted `Id`, allows one to refer to property value paths of zero length, specialization constraints also abstract superclass relationships.

To concentrate on the essential ideas, we define a simple complex object model in the next subsection. An example class schema characterizing information about students and courses for a hypothetical UNIVERSITY application in terms of this model appears in Figure 1. Our diagrammatic convention is to represent each class by a labeled rectangular box, where the label mentions the class name together with a set of immediate properties; a * following a property name indicates that the property is set-valued. For example, an object in the `student` class has a set-valued `Takes` property, while each object in the `course` class has three single-valued properties: `Inst`, `In` and `Num`.

We represent specialization constraints as directed arcs between classes. The path description associated with an unlabeled arc is assumed to correspond to `Id`, and therefore asserts that the 'to' class is a superclass of the 'from' class. Thus, the unlabeled arc from `gradCourse` to `course` implies that each `gradCourse` object is also in the `course` class, and must therefore also have `Inst`, `In` and `Num` property values (although it is more common to say that `gradCourse` *inherits* these properties from `course`). The arc labeled `Num`, from `course` to `int[100-699]`, represents a property typing constraint which restricts the values of the `Num` property for `course` objects. Another arc, from `gradCourse` to `int[500-699]`, represents another property typing constraint which further restricts the values of the `Num` property for `course` objects that are also `gradCourse` objects.

Most semantic or object-oriented data models can express the organiza-

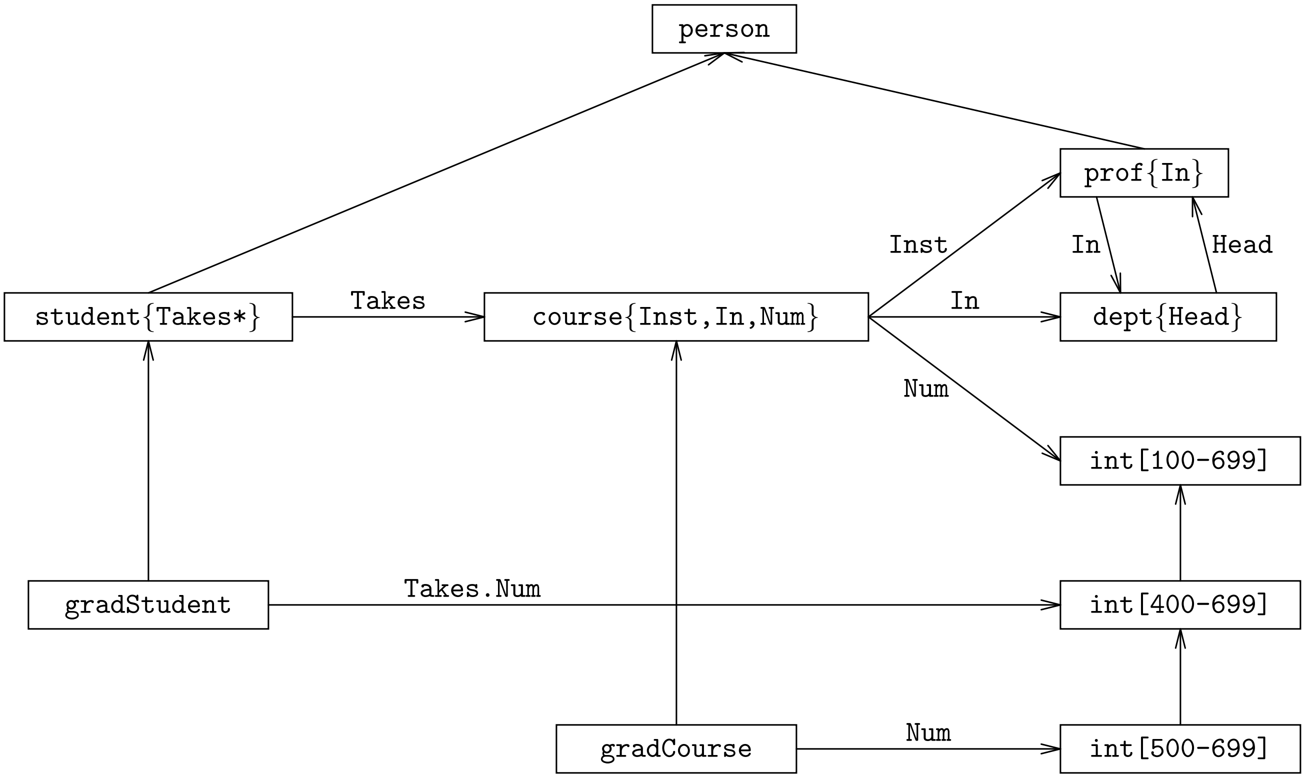

Figure 1: A UNIVERSITY Schema Diagram.

tion of information discussed so far in this example, without recourse to a general purpose constraint language containing arbitrary queries. However, so far as we are aware, the use of such a language would be necessary to express the specialization constraint represented by the arc labeled `Takes.Num`. This constraint also limits property values, but this time for a *complex* property of graduates, corresponding to the (set of) integers that are the course numbers of courses that they are taking. Since most models are unable to express such constraints, it is unlikely that existing query languages, or their parsers or optimizers, can benefit in any way from the use of such constraints.

To see how such constraints *can* be beneficial, consider a second schema, Figure 2. This schema characterizes (some of) the form of a parse tree for the relational algebra, and therefore how a query optimizer may access and update objects representing algebraic queries. First focus on the part of the schema *outside* of the area bounded by the dashed line. The four classes named `sel`, `proj`, `join` and `rName` represent four kinds of objects corresponding to selection, projection, and join operators, and relation names respectively. (For simplicity our discussion will ignore the project list and selection condition of the project and select operators, respectively. That is, we will only consider the expressions that these operators take as operands.) The `sel` and `proj` objects have single expressions as their `Arg`, while `join` objects have any number of expressions as their `Args`. The `rName` objects do not have properties. The part of the schema *inside* of the area bounded by the dashed line captures the additional structure of algebraic expressions in so-called *project-select-join* (PSJ) normal form. A PSJ query is a "projection of a selection of a join of a canonical expression," where a canonical expression may in turn be a relation name or PSJ query.

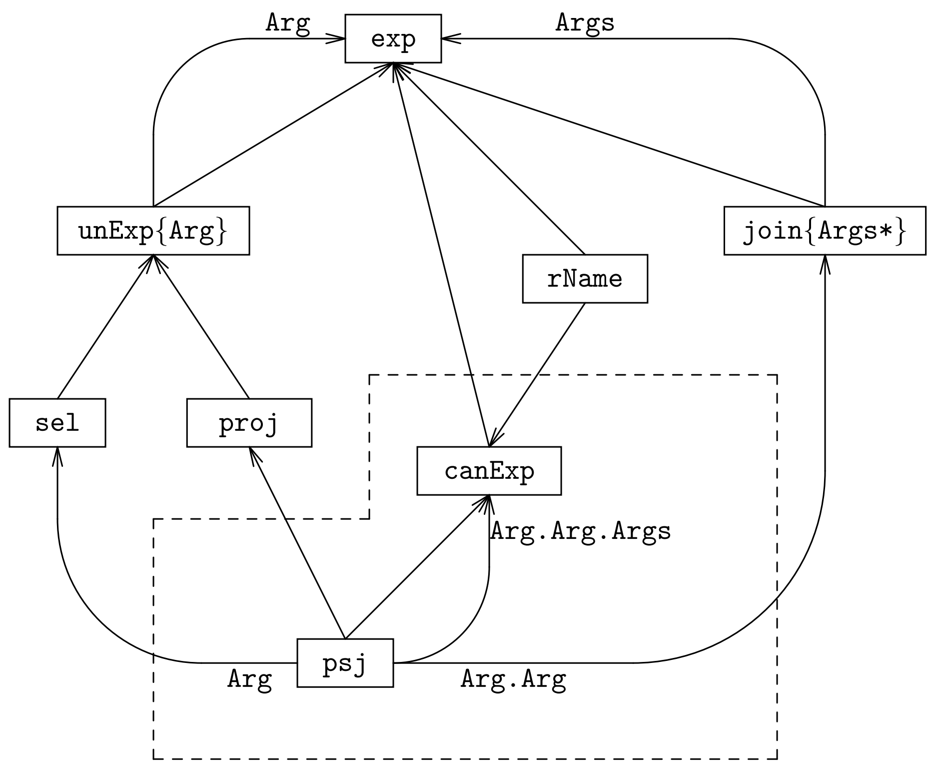

Figure 2: An ALGEBRA Schema Diagram.

Access to the components of a PSJ object can now be expressed more easily. For example, suppose it is known (through type checking) that a variable `V` references a `psj` object. Then, by following the generalization hierarchy (i.e. unlabeled arcs) to `unExp`, we find that `V.Arg` references an `exp` object. However, the arc from `psj`, labeled `Arg`, indicates that, for this subclass of `unExp`, `Arg` is actually a `sel` object. Thus `V.Arg` references a `sel` object; a subclass of `exp`. Similarly, access to the join of `V`'s selection can be expressed simply as `V.Arg.Arg`.

It is also possible to create more sophisticated object indices. Assume that the `sel` and `join` classes have additional real-valued `Selectivity` and `Size` properties, respectively. Then one can create an index of PSJ objects sorted in order of the value of

$$(\texttt{Arg.Selectivity})*(\texttt{Arg.Arg.Size}).$$

Such an index would be useful, for example, in performing join order selection.

In summary, specialization constraints also support a form of *molecular abstraction* useful to components of software development environments, which often require the manipulation of canonical forms. Although space prevents us from elaborating further on this, they are also useful to applications in computer aided design; they can be used to model typing information relating to the interconnection of internal components of complicated objects [4]. Finally, as the previous example illustrates, they enable more convenient object access as well as more sophisticated object encoding.

The remainder of this section is organized as follows. A formal definition of specialization constraints and the data model outlined above is presented next. Note that our definition of logical consequence will be based on a recent trend to view databases, hereafter referred to as *interpretations*, as labeled directed graphs [5, 6, 10, 18, 17]. We conclude with an overview of related work, and an outline of our results in the remaining sections.

1.1 Definitions

A database schema is an ordered pair $\langle S, \Sigma \rangle$ consisting of a class schema and a finite set of constraints. The class schema S consists of a finite set of declarations of the form

$$C\{P_1, ..., P_n\}$$

in which C is a class name, and the P_i are its immediate properties, written $\mathit{Props}(C)$. The set of class names in S is denoted $\mathit{Classes}(S)$, and the domain of a property P, written $\mathit{Dom}(P)$, is defined as $\{C \in \mathit{Classes}(S) \mid P \in \mathit{Props}(C)\}$. (Our usual convention when presenting examples will be to begin class names in lowercase and property names in uppercase.)

An *interpretation* for S is a (possibly infinite) labeled directed graph $G(V, A)$. The label for a vertex v is denoted $\mathit{Cl}(v)$, and corresponds to a subset of $\mathit{Classes}(S)$. The label for an arc corresponds to a property name. $G(V, A)$ must also satisfy the following condition.

A property value integrity condition: If $u \xrightarrow{P} v \in A$, then $\mathit{Cl}(u) \cap \mathit{Dom}(P) \neq \emptyset$.

The condition requires any object with one or more values for property P to be in at least one class for which P is an immediate property.

Example 1: The declarations for the ALGEBRA class schema, Figure 2, are listed in the first column of Table 1. An example interpretation for the ALGEBRA class schema appears in Figure 3. There are a total of five objects (we take the words "object" and "vertex" to be synonymous): a projection, a selection, a join and two relation names. The interpretation corresponds to (part of) the parse tree that might be produced by a query optimizer when applied to the input expression $\pi_X(\sigma_\rho(R_1 \bowtie R_2))$. □

Table 1: An ALGEBRA Schema.

S	Σ	
`exp{}`		
`join{Args}`	`join(Id:exp)`	`join(Args:exp)`
`rName{}`	`rName(Id:canExp)`	
`unExp{Arg}`	`unExp(Id:exp)`	`unExp(Arg:exp)`
		`FUNC(Arg)`
`sel{}`	`sel(Id:unExp)`	
`proj{}`	`proj(Id:unExp)`	
`canExp{}`	`canExp(Id:exp)`	
`psj{}`	`psj(Id:proj)`	`psj(Arg:sel)`
	`psj(Id:canExp)`	`psj(Arg.Arg:join)`
		`psj(Arg.Arg.Args:canExp)`

A *path description*, *pd*, is either `Id` (short for *identity*), or a sequence of property names separated by dots. Their composition and length are defined as follows:

$$pd_1 \circ pd_2 \equiv \begin{cases} pd_1 & \text{if } pd_2 \text{ is } \mathtt{Id}, \\ pd_2 & \text{if } pd_1 \text{ is } \mathtt{Id}, \\ pd_1.pd_2 & \text{otherwise.} \end{cases}$$

$$len(pd) \equiv \begin{cases} 0 & \text{if } pd \text{ is } \mathtt{Id}, \\ 1 + len(pd_1) & \text{otherwise, where } pd \text{ has the form } P \circ pd_1, \\ & \text{where } P \text{ is a property.} \end{cases}$$

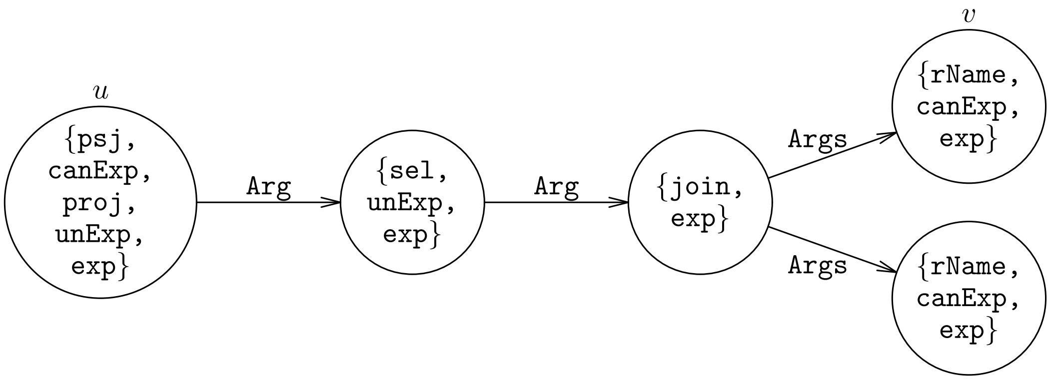

Figure 3: An interpretation of the ALGEBRA schema.

Note that composition is associative, that is, $pd_1 \circ (pd_2 \circ pd_3) = (pd_1 \circ pd_2) \circ pd_3$. The following is also a straightforward consequence of our definitions.

$$len(pd_1 \circ pd_2) = len(pd_1) + len(pd_2)$$

Given an interpretation $G(V, A)$, we say that a path in $G(V, A)$ is *described* by a path description *pd* if and only if the path is of zero length (i.e. consists of a single vertex) and *pd* is `Id`, or *pd* corresponds to the sequence of property labels on the path.

Example 2: In Figure 3, the single path from the vertex u to the vertex v is described by the path description `Arg.Arg.Args`. □

We consider two kinds of constraints in Σ. The first, called a *property functionality constraint* (`FUNC`), allows us to characterize properties that are single-valued and total on their domain classes. Each is declared with the form `FUNC`(P), and is *satisfied* by an interpretation $G(V, A)$ for S if and only if $G(V, A)$ satisfies the following two additional conditions.

1. (*property functionality*) If $u \xrightarrow{P} v$ and $u \xrightarrow{P} w$ are in A, then $v = w$.
2. (*property value completeness*) If $Cl(u) \cap Dom(P) \neq \emptyset$, then there is an arc $u \xrightarrow{P} v \in A$.

The second kind of constraint is called a *specialization constraint* (SC), and is our main concern in this paper. An SC (over S) is used to assert a *type* condition on a path description, and has the form $C_1(pd : C_2)$. Assuming that S is the schema over which the constraint is expressed, then $C_1(pd : C_2)$

is satisfied by an interpretation $G(V, A)$ for S if and only if whenever there exists a path in $G(V, A)$ from a vertex u to a vertex v described by pd, then $C_1 \in Cl(u)$ implies $C_2 \in Cl(v)$.

Example 3: The third entry in the third column of Table 1 is the only FUNC needed for the ALGEBRA schema. Note that this constraint is satisfied by the interpretation in Figure 3. The remaining entries in the second and third columns of Table 1 are the SCs needed for the ALGEBRA database. It is a simple exercise to confirm that each is satisfied by the interpretation for ALGEBRA given in Figure 3. Recall from our introductory comments that the entries in the second column, with the form $C_1(\mathtt{Id} : C_2)$, represent the superclass relationships. This should be clear from the above definitions, since such constraints distinguish interpretations in which a vertex v has C_1 in its class label, but not C_2. □

Let Σ be a finite set of constraints (SCs and FUNCs). By $\Sigma_{\mathtt{FUNC}}$ and $\Sigma_{\mathtt{SC}}$, we mean the sets of FUNCs and SCs in Σ, respectively. An instance of either kind of constraint σ is a *logical consequence* of Σ, written $\Sigma \models \sigma$, if any interpretation satisfying Σ must also satisfy σ.

As discussed, specialization constraints using Id are our means of establishing a generalization taxonomy among classes. It is therefore worthwhile to distinguish database schema which fail to satisfy the usual requirement that such taxonomies be acyclic. Accordingly, we shall say that S is a *generalization taxonomy* with respect to Σ if and only if there are no two distinct classes $C_1, C_2 \in \mathit{Classes}(S)$ such that

$$\Sigma \models C_1(\mathtt{Id} : C_2) \text{ and } \Sigma \models C_2(\mathtt{Id} : C_1).$$

However, note that we continue to allow *multiple inheritance*; there may exist $C_1, C_2, C_3 \in \mathit{Classes}(S)$ such that $\Sigma \models C_1(\mathtt{Id} : C_2)$ and $\Sigma \models C_1(\mathtt{Id} : C_3)$, but where $\Sigma \not\models C_2(\mathtt{Id} : C_3)$ and $\Sigma \not\models C_3(\mathtt{Id} : C_2)$. The three ALGEBRA classes `psj`, `proj` and `canExp` are an example of this.

The notion of a specialization constraint so far presented is very general, allowing us to express such constraints as

$$\mathtt{exp}(\mathtt{Arg} : \mathtt{exp}) \tag{1.1}$$

or even

$$\mathtt{rName}(\mathtt{Arg} : \mathtt{exp}). \tag{1.2}$$

Concerning the first example, the diagram of the ALGEBRA schema in Figure 2, together with our intuition about the information it describes, suggests that not all `exp` object can meaningly have an `Arg` property value, such as those that are also `join` objects. The second example is more extreme; we might expect that there will never be *any* `rName` object with an `Arg` property value, which implies that 1.2 is *vacuously* satisfied. These examples illustrate the need to be able to distinguish some kinds of path descriptions—in particular, those that correspond to single or set-valued functions which are *total* with respect to some class.

We begin by first defining the more restricted notion of *well-formed path function* [17]. A path description *pd* is a well-formed path function with respect to $\langle S, \Sigma \rangle$ and a class $C \in \mathit{Classes}(S)$ if and only if for any interpretation $G(V, A)$ satisfying Σ, whenever there is a vertex $u \in V$ such that $C \in \mathit{Cl}(u)$, then there must be a *unique* path in $G(V, A)$ from u described by *pd*. The set of well-formed path functions with respect to $\langle S, \Sigma \rangle$ and C is denoted $\mathit{PathFuncs}(C)$, where $\langle S, \Sigma \rangle$ is assumed to be understood from context.

A path description *pd* is *well-formed* with respect to $\langle S, \Sigma \rangle$ and a class $C \in \mathit{Classes}(S)$ if and only if it is a well-formed path function with respect to $\langle S, \Sigma \cup S_{\mathtt{FUNC}} \rangle$ and C, where

$$S_{\mathtt{FUNC}} = \{\, \mathtt{FUNC}(P) \mid \ P \text{ is a property in } S \}.$$

As above, the set of path descriptions that are well-formed with respect to $\langle S, \Sigma \rangle$ and C is denoted $\mathit{PathDescs}(C)$. Note that that $\mathit{PathDescs}(C)$ (and $\mathit{PathFuncs}(C)$) can still be countably infinite for some classes. For example, when $S = \{\, \mathtt{a}\{\mathtt{A}\} \,\}$ and $\Sigma = \{\, \mathtt{a}(\mathtt{A} : \mathtt{a}) \,\}$, then $\mathit{PathDescs}(\mathtt{a})$ consists of `Id`, `A`, `A.A`, `A.A.A`, and so on.

We extend the concept of well-formedness to SCs. An SC $C_1(pd : C_2)$ is well-formed with respect to $\langle S, \Sigma \rangle$ if and only if $pd \in \mathit{PathDescs}(C_1)$. Finally, a finite set of constraints Σ_1 is well-formed with respect to $\langle S, \Sigma \rangle$ if and only if every SC in Σ_1 is well-formed with respect to $\langle S, \Sigma \rangle$.

Example 4: For the ALGEBRA schema in Table 1, $\mathit{PathFuncs}(\mathtt{psj})$ and $\mathit{PathDescs}(\mathtt{psj})$ denote

$$\{\mathtt{Id,\ Arg,\ Arg.Arg}\}$$

and

$$\{\mathtt{Id,\ Arg,\ Arg.Arg,\ Arg.Arg.Args}\}$$

respectively. Also, both SCs 1.1 and 1.2 above are not well-formed with respect to the ALGEBRA schema, since `Arg` $\notin$ *PathDescs*(`RName`) $\cup$ *PathDescs*(`exp`). □

1.2 Review and outline

Our form of specialization constraint is more general than the combination of *subsetting* and *typing* constraints considered by Di Battista and Lenzerini [11], although other forms of constraints are included in their theory which are not expressible in our own. Arisawa and Miura [3] consider richer forms of subclass constraints, such as $C_1 * C_2 * C_3 < C_4 + C_5$, which states that the intersection of (the extensions of) classes C_1, C_2 and C_3 is a subset of the union of classes C_4 and C_5. They also outline polynomial time decision algorithms for cases in which either unions or intersections appear exclusively. Specialization and subsetting constraints have also been considered in the context of the relational model. Such constraints are called *inclusion dependencies*, and have the form $R(A_1, ..., A_n) \subseteq S(B_1, .., B_n)$, where R and S are relation names, and $A_1, ..., B_n$ are attribute names. The constraint is satisfied by relations r and s if the projection of r over $A_1, ..., A_n$ is a subset of the projection of s over $B_1, ..., B_n$. The inference problem for inclusion dependencies is P-space complete in the general case [7], but can be solved in linear time if dependencies are unary (i.e. $n = 1$) [12]. Casanova et al. [7] also considered the interaction of inclusion dependencies with functional dependencies. A more general form of functional dependency, for a data model similar to the one in this paper, has been considered in Weddell [16, 17]; its interaction with specialization constraints is under study [8].

The concept of a specialization constraint, as we have defined it, was first presented in [19]. In the remainder of this paper, we expand on the results presented by this earlier work, focusing on the various membership problems for specialization constraints, including the problems of identifying well-formed path functions and path descriptions. In general, we consider these problems for two models. The first, the subject of Section 2 which follows, imposes no constraints on class membership for an object beyond those implied by subclassing constraints. An object may be a member of any set of classes as long as the superclass-subclass relationships are maintained. (An interpretation of the UNIVERSITY database schema, Figure 1, might therefore include objects in both the `student` and `prof` classes, or indeed in all classes.) For this case, we present a sound and complete axiomatization for arbitrary specialization constraints, and efficient decision procedures for the corresponding membership problems.

The second model is more typical and requires that each object is created with respect to at most one class. If the set of classes to which an object belongs is nonempty, then the set must include one class for which all other classes in the set are superclasses. In Section 3, membership problems for this model are shown to be NP-hard, and NP-complete if class schema include a "bottom" class, denoted $\perp$. We exhibit polynomial-time decision procedures when $\perp$ does exist and the length of any path descriptions mentioned in antecedent specialization constraints is *bounded* by some constant.

In Section 4, we consider another special case for the second model, in which class schema also satisfy a *lower semi-lattice* condition. A sound and complete axiomatization for *well-formed* specialization constraints is presented, together with efficient decision procedures for the membership problem for well-formed constraints, and for determining if an arbitrary constraint is well-formed. We prove that the membership problem for arbitrary specialization constraints remains NP-complete, however, even for class schema satisfying the lower semi-lattice condition.

Table 2 summarizes our complexity results for the various membership problems, and indicates those cases for which a complete axiomatization is also presented. The number in parenthesis is the subsection in which the result is derived. Further summary comments appear in Section 5.

2. Implication Problems for Specialization Constraints

2.1 On finite implications

Let Σ be a finite set of constraints and let σ be an additional SC. In this section, it will be shown that σ is a logical consequence of Σ if and only if σ is a *finite* logical consequence of Σ. Formally, σ is a finite logical consequence of Σ, written $\Sigma \models^{\text{finite}} \sigma$, if any *finite* interpretation satisfying Σ must also satisfy σ. By definition, $\Sigma \models \sigma$ implies $\Sigma \models^{\text{finite}} \sigma$. In the following, we will prove its opposite direction; that is, $\Sigma \not\models \sigma$ implies $\Sigma \not\models^{\text{finite}} \sigma$. Let us denote σ by $C(pd : C')$.

For a path description *pd* $(= P_1.P_2.\cdots.P_n)$, a *pd-List* (for S) is a directed graph

$$v_0 \xrightarrow{P_1} v_1 \xrightarrow{P_2} \cdots \xrightarrow{P_n} v_n,$$

where each v_i has a label $Cl(v_i)$ that is a subset of $\mathit{Classes}(S)$. If $pd = \texttt{Id}$, then the *pd*-List consists of a single vertex v_0 with no arcs. Note that the *pd*-List is not necessarily an interpretation, since it may not satisfy a property value integrity condition.

Table 2: Complexity results for membership problems.

problem	(general case)	MSC/$\perp$	MSC/$\perp$ (bounded)	MSC/LSL
$pd \notin PathDescs(C)$	P-time (2.3)	NP-complete (3.1)	P-time (3.2)	P-time (4.2)
$pd \notin PathFuncs(C)$	P-time (2.3)	NP-complete (3.1)	P-time (3.2)	P-time (4.2)
$\Sigma \not\models C(pd : C')$ (where $pd \in PathDescs(C)$)	P-time(*) (2.3)	NP-complete (3.1)	P-time (3.2)	P-time(*) (4.2)
$\Sigma \not\models C(pd : C')$	P-time(*) (2.3)	NP-complete (3.1)	P-time (3.2)	NP-complete (4.3)
	(*) – complete axiomatization			

Lemma 1: If $\Sigma \not\models C(pd : C')$, then there is a pd-List $v_0 \xrightarrow{P_1} v_1 \xrightarrow{P_2} \cdots \xrightarrow{P_n} v_n$ that satisfies the following three conditions, where $pd = P_1.P_2. \cdots .P_n$.

PDL 1: The pd-List satisfies a property value integrity condition.

PDL 2: The pd-List satisfies Σ_{SC}.

PDL 3: The pd-List violates $C(pd : C')$; that is, $C \in Cl(v_0)$ and $C' \notin Cl(v_n)$.

Proof. Assume that $\Sigma \not\models C(pd : C')$. Then there is an interpretation $G(V, A)$ that satisfies Σ but violates $C(pd : C')$. That is, there is a path in $G(V, A)$ from a vertex u to a vertex v described by pd such that $C \in Cl(u)$ but $C' \notin Cl(v)$. The path can be denoted $u_0 \xrightarrow{P_1} u_1 \xrightarrow{P_2} \cdots \xrightarrow{P_n} u_n$, where $u_0 = u$ and $u_n = v$.[1] Let $v_0 \xrightarrow{P_1} v_1 \xrightarrow{P_2} \cdots \xrightarrow{P_n} v_n$ be a pd-List such that $Cl(v_i) = Cl(u_i)$ for $0 \leq i \leq n$. We will prove that the pd-List satisfies PDLs 1 to 3.

Since $G(V, A)$ is an interpretation, it satisfies a property value integrity condition. Thus it holds that $Cl(u_{i-1}) \cap Dom(P_i) \neq \emptyset$ for $1 \leq i \leq n$. Since $Cl(v_{i-1}) = Cl(u_{i-1})$ by definition, the pd-List also satisfies a property value integrity condition; that is, PDL 1 holds.

[1] A vertex u_i may coincide with another u_j, since $G(V, A)$ may contain cycles.

Since $G(V, A)$ satisfies $\Sigma_{\mathtt{SC}}$ ($\subseteq \Sigma$) and the path $u_0 \xrightarrow{P_1} u_1 \xrightarrow{P_2} \cdots \xrightarrow{P_n} u_n$ is a subgraph of $G(V, A)$, the path must satisfy $\Sigma_{\mathtt{SC}}$. Thus the *pd*-List also satisfies $\Sigma_{\mathtt{SC}}$; that is, PDL 2 holds.

Since $C \in Cl(u_0) = Cl(v_0)$ and $C' \notin Cl(u_n) = Cl(v_n)$, the *pd*-List violates $C(pd : C')$; that is, PDL 3 holds. This completes proving Lemma 1. □

Example 5: Let S be a class schema defined as follows:

$\mathtt{a}_1\{\mathtt{C}\}$, $\mathtt{a}_2\{\ \}$, $\mathtt{a}_3\{\mathtt{B}\}$, $\mathtt{b}\{\mathtt{A,D}\}$, $\mathtt{c}_1\{\ \}$, $\mathtt{c}_2\{\ \}$, $\mathtt{e}\{\mathtt{C}\}$,

where

$Dom(\mathtt{A}) = \{\mathtt{b}\}$, $Dom(\mathtt{B}) = \{\mathtt{a}_3\}$, $Dom(\mathtt{C}) = \{\mathtt{a}_1, \mathtt{e}\}$, $Dom(\mathtt{D}) = \{\mathtt{b}\}$.

Let $\Sigma_{\mathtt{SC}}$ consist of the following seven SCs:

$\mathtt{a}_1(\mathtt{Id} : \mathtt{a}_2)$, $\mathtt{a}_1(\mathtt{C} : \mathtt{c}_2)$, $\mathtt{a}_2(\mathtt{Id} : \mathtt{a}_3)$, $\mathtt{a}_3(\mathtt{B} : \mathtt{b})$, $\mathtt{b}(\mathtt{A} : \mathtt{a}_2)$, $\mathtt{c}_1(\mathtt{Id} : \mathtt{c}_2)$, $\mathtt{e}(\mathtt{C} : \mathtt{c}_2)$.

Assume that $\Sigma_{\mathtt{FUNC}} = \emptyset$. The database schema $\langle S, \Sigma \rangle$ can be illustrated as in Figure 4.

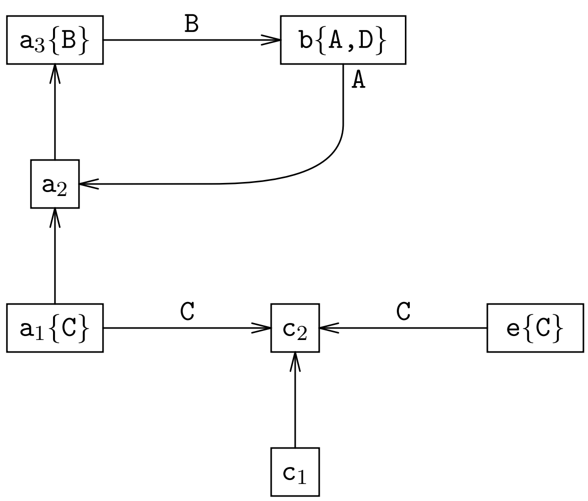

Figure 4: A database schema $\langle S, \Sigma \rangle$.

Let us consider whether or not $\Sigma \models \mathtt{a}_2(\mathtt{B.A.C} : \mathtt{c}_1)$. To say the conclusion first, it holds that $\Sigma \not\models \mathtt{a}_2(\mathtt{B.A.C} : \mathtt{c}_1)$. In fact, for the SC $\mathtt{a}_2(\mathtt{B.A.C} : \mathtt{c}_1)$, there is a '$\mathtt{B.A.C}$'-List $v_0 \xrightarrow{\mathtt{B}} v_1 \xrightarrow{\mathtt{A}} v_2 \xrightarrow{\mathtt{C}} v_3$ satisfying PDLs 1 to 3, as is given in Figure 5. □

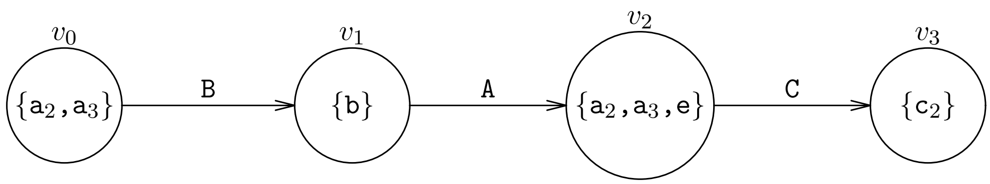

Figure 5: A 'B.A.C'-List.

The *pd*-List obtained in Lemma 1 does not necessarily satisfy $\Sigma_{\texttt{FUNC}}$, though it satisfies $\Sigma_{\texttt{SC}}$. Note that if the *pd*-List does not satisfy $\Sigma_{\texttt{FUNC}}$, then there is a vertex v_i that violates property value completeness for some $\texttt{FUNC}(P) \in \Sigma_{\texttt{FUNC}}$; that is, $Cl(v_i) \cap Dom(P) \neq \emptyset$ but v_i has no out-arc labeled P.[2] By adding one vertex and a number of arcs to the *pd*-List, however, we can construct an interpretation $G(V, A)$ that satisfies $\Sigma \cup S_{\texttt{FUNC}}$.[3] Let us first consider how to construct such an interpretation.

Let $v_0 \xrightarrow{P_1} v_1 \xrightarrow{P_2} \cdots \xrightarrow{P_n} v_n$ be a *pd*-List, where $pd = P_1.P_2.\cdots.P_n$. The *augmented graph* of the *pd*-List with respect to S is a directed graph $G(V, A)$ obtained by the following procedure.

Procedure 1: (Computing the augmented graph $G(V, A)$ of a *pd*-List with respect to S.)
input: a database schema $\langle S, \Sigma \rangle$ and a *pd*-List $v_0 \xrightarrow{P_1} v_1 \xrightarrow{P_2} \cdots \xrightarrow{P_n} v_n$.

1. Assign the *pd*-List to $G(V, A)$.
2. Add a new vertex u to V, where $Cl(u) = Classes(S)$.
3. For each property P in S and each v_i, where $0 \leq i \leq n$, if $Cl(v_i) \cap Dom(P) \neq \emptyset$ and v_i has no out-arc labeled P, then add an arc $v_i \xrightarrow{P} u$ to A.
4. For each property P in S, add an arc $u \xrightarrow{P} u$ to A. □

Example 6: Let us consider how to construct the augmented graph $G(V, A)$ of the 'B.A.C'-List $v_0 \xrightarrow{\texttt{B}} v_1 \xrightarrow{\texttt{A}} v_2 \xrightarrow{\texttt{C}} v_3$ in Figure 5 with respect to S.

In Step 1, a vertex u with $Cl(u) = Classes(S)$ is added to V. Since $\texttt{b} \in Cl(v_1) \cap Dom(\texttt{D})$, an arc $v_2 \xrightarrow{\texttt{B}} u$ is added to A in Step 3. Similarly, since $\texttt{a}_3 \in Cl(v_2) \cap Dom(\texttt{B})$, an arc $v_2 \xrightarrow{\texttt{B}} u$ is also added to A in Step 3. Since

[2]Since each vertex in the *pd*-List has at most one out-arc, it satisfies property functionality.

[3]Note that $\Sigma_{\texttt{FUNC}} \subseteq S_{\texttt{FUNC}}$ by definition.

S consists of four properties A, B, C, D, four arcs (such as $u \xrightarrow{\texttt{A}} u$) is added to A in Step 4. As a result, the augmented graph $G(V, A)$ is constructed as in Figure 6. It is important to note that $G(V, A)$ is a finite interpretation satisfying $\Sigma \cup S_{\texttt{FUNC}}$ but violating $\texttt{a}_2\texttt{(B.A.C : c}_1\texttt{)}$, where $S_{\texttt{FUNC}} = \{\, \texttt{FUNC(A)}, \texttt{FUNC(B)}, \texttt{FUNC(C)}, \texttt{FUNC(D)} \,\}$. Thus it turns out that $\Sigma \not\models^{\text{finite}} \texttt{a}_2\texttt{(B.A.C : c}_1\texttt{)}$. Note that there is an infinite interpretation for S satisfying Σ, because of a cycle $\texttt{a}_2\texttt{(Id : a}_3\texttt{)}$, $\texttt{a}_3\texttt{(B : b)}$, $\texttt{b(A : a}_2\texttt{)}$. □

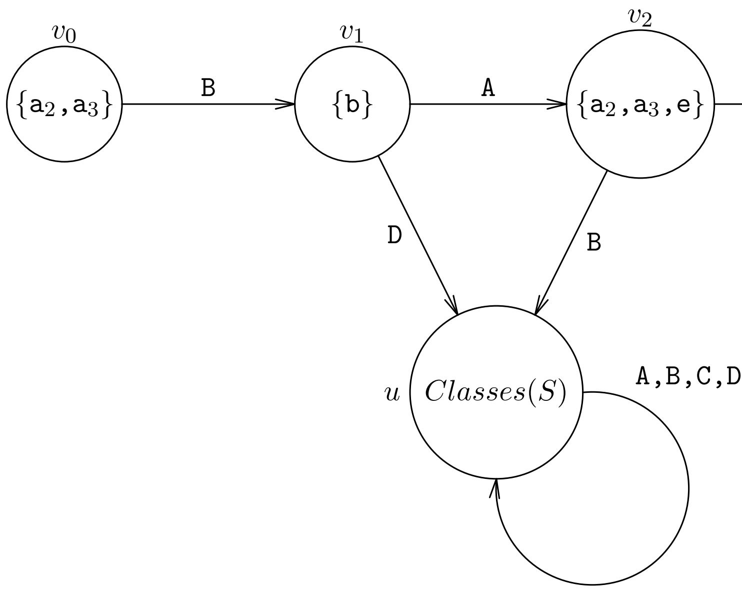

Figure 6: The augmented graph of the 'B.A.C'-List in Figure 5.

Lemma 2: If the *pd*-List $v_0 \xrightarrow{P_1} v_1 \xrightarrow{P_2} \cdots \xrightarrow{P_n} v_n$ satisfies PDLs 1 to 3, then its augmented graph $G(V, A)$ with respect to S is a finite interpretation satisfying $\Sigma \cup S_{\texttt{FUNC}}$ but violating $C(pd : C')$.

Proof. Clearly, $G(V, A)$ is finite. In order to prove that $G(V, A)$ is an interpretation, it suffices to show that $G(V, A)$ satisfies a property value integrity condition. This is true as explained below: For each $v_{i-1} \xrightarrow{P_i} v_i \in A$, it holds that $Cl(v_{i-1}) \cap Dom(P_i) \neq \emptyset$, since the *pd*-List satisfies a property value integrity condition by PDL 1. For each $v_i \xrightarrow{P} u$ added to A in Step 3, it follows from the if condition in Step 3 that $Cl(v_i) \cap Dom(P) \neq \emptyset$. Furthermore, for each $u \xrightarrow{P} u$ added to A in Step 4, it also follows that $Cl(u) \cap Dom(P) \neq \emptyset$, since $Cl(u) = Classes(S)$. Thus $G(V, A)$ satisfies a property value integrity condition.

We next prove that $G(V, A)$ satisfies $\Sigma \cup S_{\mathtt{FUNC}}$. By construction, it is clear that $G(V, A)$ satisfies $\mathtt{FUNC}(P)$ for every property P in S. It remains to show that $G(V, A)$ satisfies $\Sigma_{\mathtt{SC}}$. For an SC $C_1(pd' : C_2) \in \Sigma_{\mathtt{SC}}$, assume that there is a path in $G(V, A)$ from a vertex w to a vertex w' described by pd' such that $C_1 \in Cl(w)$. It suffices to show that $C_2 \in Cl(w')$. If $w' = u$, then this is true, since $Cl(u) = Classes(S)$ by definition. Assume that $w' \neq u$. Since the destination of every arc added in Steps 3 and 4 is the new vertex u, the assumption $w' \neq u$ implies that the path from w to w' must be on the pd-List. Since the pd-List satisfies $C_1(pd' : C_2) \in \Sigma_{\mathtt{SC}}$ by PDL 2, $C_1 \in Cl(w)$ implies $C_2 \in Cl(w')$.

Finally, $G(V, A)$ violates $C(pd : C')$, since the pd-List is a subgraph of $G(V, A)$ and violates the SC by PDL 3. This completes proving Lemma 2. □

We are now ready to prove the following theorem.

Theorem 1: The following three statements are equivalent.

1. $\Sigma \not\models C(pd : C')$.
2. $\Sigma \not\models^{\text{finite}} C(pd : C')$.
3. There is a pd-List satisfying PDLs 1 to 3.

Proof. By definition, (2) implies (1). By Lemma 1, (1) implies (3). We prove that (3) implies (2). Assume that there is a pd-List satisfying PDLs 1 to 3. Let $G(V, A)$ be the augmented graph of the pd-List with respect to S. Since $\Sigma \subseteq \Sigma \cup S_{\mathtt{FUNC}}$, it follows from Lemma 2 that $G(V, A)$ is a finite interpretation satisfying Σ but violating $C(pd : C')$. Hence $\Sigma \not\models^{\text{finite}} C(pd : C')$. □

2.2 Axioms for SCs

Let Σ be a finite set of constraints and let σ be an SC. For a finite set $\mathcal{A}$ of axioms, if σ is derivable from Σ using the axioms in $\mathcal{A}$, then we denote $\Sigma \vdash_{\mathcal{A}} \sigma$ (or $\Sigma \vdash \sigma$, if $\mathcal{A}$ is understood from context). In this section, we will show the following result.

The following axioms A1 to A3 are sound and complete for deciding whether or not $\Sigma \models \sigma$; that is, $\Sigma \models \sigma$ if and only if $\Sigma \vdash_{\{\text{A1,A2,A3}\}} \sigma$. In particular, if σ is well-formed with respect to $\langle S, \Sigma \rangle$, then only A1 and A2 are sound and complete.

A1: (*identity*) If $C \in Classes(S)$, then $C(\mathtt{Id} : C)$.

A2: (*composition*) If $C(pd : C')$ and $C'(pd' : C'')$, then $C(pd \circ pd' : C'')$.

A3: (*prefix augmentation*) For a property P, if $C_p(P \circ pd : C')$ for every $C_p \in Dom(P)$, then $C(pd' \circ P \circ pd : C')$ for every $C \in Classes(S)$ and every path description pd'.

Lemma 3: Axioms A1 to A3 are sound; that is, $\Sigma \vdash \sigma$ implies $\Sigma \models \sigma$.

Proof. Clearly, A1 and A2 are sound. Consider A3. Assume that an interpretation $G(V, A)$ satisfies $C_p(P \circ pd : C')$ for every $C_p \in Dom(P)$. We must show that $G(V, A)$ satisfies $C(pd' \circ P \circ pd : C')$. Assume that there is a path in $G(V, A)$ from a vertex u to a vertex v described by $pd' \circ P \circ pd$ such that $C \in Cl(u)$. It suffices to show that $C' \in Cl(v)$. The path can be divided into two parts: one path from u to a vertex w described by pd' and the other from w to v described by $P \circ pd$. Since (1) $G(V, A)$ satisfies a property value integrity condition and (2) w has an out-arc labeled P, it holds that $Cl(w) \cap Dom(P) \neq \emptyset$. Let $C_p \in Cl(w) \cap Dom(P)$. Then $G(V, A)$ satisfies $C_p(P \circ pd : C')$ by assumption. Furthermore, since there is a path in $G(V, A)$ from w to v described by $P \circ pd$, $C_p \in Cl(w)$ implies $C' \in Cl(v)$. □

Example 7: Consider the database schema $\langle S, \Sigma \rangle$ of Example 5. For property `C`, where $Dom(\texttt{C}) = \{\texttt{a}_1, \texttt{e}\}$, both $\texttt{a}_1(\texttt{C} : \texttt{c}_2)$ and $\texttt{e}(\texttt{C} : \texttt{c}_2)$ are in Σ. Thus by axiom A3, an SC $\texttt{b}(\texttt{D.C} : \texttt{c}_2)$ is derived from Σ. Similarly, an SC $\texttt{a}_2(\texttt{B.A.C} : \texttt{c}_2)$ is also derived from Σ by axiom A3. □

In the following we will prove completeness of the axioms; that is, $\Sigma \not\vdash \sigma$ implies $\Sigma \not\models \sigma$. Let us denote σ by $C(pd : C')$. The proof will be done by showing that if $\Sigma \not\vdash C(pd : C')$, then there is a pd-List satisfying PDLs 1 to 3, and thus $\Sigma \not\models C(pd : C')$ by Theorem 1. In order to construct such a pd-List, we will introduce the concept of *chase*, which is a pd-List satisfying $\Sigma_{\texttt{SC}}$ for a class $C \in Classes(S)$ and a path description pd.

The chase of C and pd under $\Sigma_{\texttt{SC}}$, written $Chase_{\Sigma_{\texttt{SC}}}(C, pd)$, is a pd-List $v_0 \xrightarrow{P_1} v_1 \xrightarrow{P_2} \cdots \xrightarrow{P_n} v_n$ obtained by the following procedure, where $pd = P_1.P_2.\cdots.P_n$.

Procedure 2: (Computing $Chase_{\Sigma_{\texttt{SC}}}(C, pd)$.)
input: a database schema $\langle S, \Sigma \rangle$, a class $C \in Classes(S)$, and a path description pd $(= P_1.P_2.\cdots.P_n)$.

1. Construct a *pd*-List $v_0 \xrightarrow{P_1} v_1 \xrightarrow{P_2} \cdots \xrightarrow{P_n} v_n$ such that $Cl(v_0) = \{ C \}$ and $Cl(v_i) = \emptyset$ for $1 \leq i \leq n$.

2. Apply the following rule to the *pd*-List exhaustively:

   SC-rule: For an SC $C_a(pd' : C_b) \in \Sigma_{\texttt{SC}}$, if there are two vertices v_i, v_j such that $C_a \in Cl(v_i)$, $C_b \notin Cl(v_j)$, and $pd' = P_{i+1}.P_{i+2}. \cdots .P_j$, then add C_b to $Cl(v_j)$.[4] □

Example 8: For the database schema $\langle S, \Sigma \rangle$ in Example 5, let us compute $Chase_{\Sigma_{\texttt{SC}}}(\texttt{a}_2, \texttt{B.A.C})$.

In Step 1, a '$\texttt{B.A.C}$'-List $v_0 \xrightarrow{\texttt{B}} v_1 \xrightarrow{\texttt{A}} v_2 \xrightarrow{\texttt{C}} v_3$ is constructed, where $Cl(v_0) = \{ \texttt{a}_2 \}$ and $Cl(v_i) = \emptyset$ for $1 \leq i \leq 3$.

For SC $\texttt{a}_2(\texttt{Id} : \texttt{a}_3) \in \Sigma_{\texttt{SC}}$, since $\texttt{a}_2 \in Cl(v_0)$ and $\texttt{a}_3 \notin Cl(v_0)$, class $\texttt{a}_3$ is added to $Cl(v_0)$ by applying the SC-rule for $\texttt{a}_2(\texttt{Id} : \texttt{a}_3)$. Then by applying the SC-rule for $\texttt{a}_3(\texttt{B} : \texttt{b}) \in \Sigma_{\texttt{SC}}$, class $\texttt{b}$ is added to $Cl(v_1)$, since $\texttt{a}_3 \in Cl(v_0)$ and $\texttt{b} \notin Cl(v_1)$. Finally, we obtain the '$\texttt{B.A.C}$'-List given in Figure 7 as $Chase_{\Sigma_{\texttt{SC}}}(\texttt{a}_2, \texttt{B.A.C})$. Note that $Chase_{\Sigma_{\texttt{SC}}}(\texttt{a}_2, \texttt{B.A.C})$ satisfies $\Sigma_{\texttt{SC}}$. □

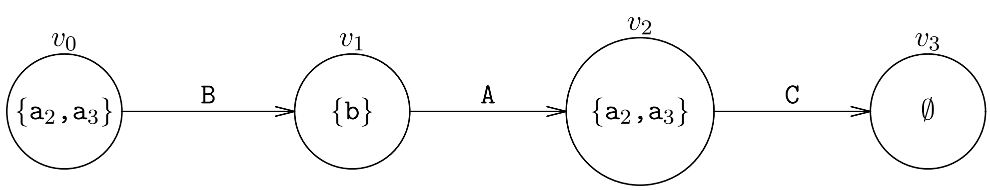

Figure 7: $Chase_{\Sigma_{\texttt{SC}}}(\texttt{a}_2, \texttt{B.A.C})$.

Note that the result of Procedure 2 is independent of the order of applying SC-rules in Step 2.

Let us first consider the case that $C(pd : C')$ is well-formed with respect to $\langle S, \Sigma \rangle$; that is, $pd \in PathDescs(C)$. We want to show that if $\Sigma \not\vdash_{\{A1,A2\}} C(pd : C')$, then $Chase_{\Sigma_{\texttt{SC}}}(C, pd)$ satisfies PDLs 1 to 3. We will present three lemmas, which will be used for the chase to satisfy the three conditions.

Assume in the rest of this section that $pd = P_1.P_2. \cdots .P_n$ and that the result of $Chase_{\Sigma_{\texttt{SC}}}(C, pd)$ is $v_0 \xrightarrow{P_1} v_1 \xrightarrow{P_2} \cdots \xrightarrow{P_n} v_n$, unless stated otherwise. Furthermore, let us denote a prefix $P_1.P_2. \cdots .P_i$ of pd by pd_i, where $0 \leq i \leq n$.[5]

[4] If $pd' = \texttt{Id}$, then let $i = j$.
[5] Note that $pd_0 = \texttt{Id}$ and $pd_n = pd$.

Lemma 4: (a) The pd_i-List $v_0 \xrightarrow{P_1} v_1 \xrightarrow{P_2} \cdots \xrightarrow{P_i} v_i$ satisfies Σ_{SC}, where $0 \leq i \leq n$. In particular, $Chase_{\Sigma_{\mathsf{SC}}}(C, pd)$ satisfies Σ_{SC}.

(b) $Chase_{\Sigma_{\mathsf{SC}}}(C, pd_i)$ coincides with the pd_i-List $v_0 \xrightarrow{P_1} v_1 \xrightarrow{P_2} \cdots \xrightarrow{P_i} v_i$, where $0 \leq i \leq n$.

Proof. By definition, if a pd-List does not satisfy an SC in Σ_{SC}, then the SC-rule for the SC must apply to the pd-List. In order to compute $Chase_{\Sigma_{\mathsf{SC}}}(C, pd)$, SC-rules for Σ_{SC} have been exhaustively applied, and thus $Chase_{\Sigma_{\mathsf{SC}}}(C, pd)$ satisfies Σ_{SC}. Hence the pd_i-List also satisfies Σ_{SC}, since it is a subgraph of $Chase_{\Sigma_{\mathsf{SC}}}(C, pd)$. That is, Lemma 4(a) holds.

We next prove Lemma 4(b). Since the order of applying SC-rules in Step 2 does not affect the final result, Procedure 2 can yield, as an intermediate result, a pd-List $v'_0 \xrightarrow{P_1} v'_1 \xrightarrow{P_2} \cdots \xrightarrow{P_n} v'_n$ such that (1) $Cl(v'_j) = \emptyset$ for $i+1 \leq j \leq n$ and (2) the pd_i-List $v'_0 \xrightarrow{P_1} v'_1 \xrightarrow{P_2} \cdots \xrightarrow{P_i} v'_i$ satisfies Σ_{SC}. Clearly, the pd_i-List coincides with $Chase_{\Sigma_{\mathsf{SC}}}(C, pd_i)$. On the other hand, by the definition of SC-rule, once $v_0 \xrightarrow{P_1} v_1 \xrightarrow{P_2} \cdots \xrightarrow{P_i} v_i$ satisfies Σ_{SC} in Step 2, the pd_i-List remains unchanged afterward. Hence Lemma 4(b) holds. □

Lemma 5: $Cl(v_i) = \{C_i \in Classes(S) \mid \Sigma_{\mathsf{SC}} \vdash_{\{\text{A1,A2}\}} C(pd_i : C_i)\}$ for $0 \leq i \leq n$.

Proof. We first prove that if $\Sigma_{\mathsf{SC}} \vdash_{\{\text{A1,A2}\}} C(pd_i : C_i)$, then $C_i \in Cl(v_i)$. Assume that $\Sigma_{\mathsf{SC}} \vdash_{\{\text{A1,A2}\}} C(pd_i : C_i)$. Then $\Sigma_{\mathsf{SC}} \models C(pd_i : C_i)$ by Lemma 3. Thus $Chase_{\Sigma_{\mathsf{SC}}}(C, pd)$ satisfies $C(pd_i : C_i)$ by Lemma 4(a). Hence $C \in Cl(v_0)$ implies $C_i \in Cl(v_i)$.

We next prove that if $C_i \in Cl(v_i)$, then $\Sigma_{\mathsf{SC}} \vdash_{\{\text{A1,A2}\}} C(pd_i : C_i)$. Induction on the number of applying SC-rules in Step 2.

The basis follows from axiom A1, since $Cl(v_0) = \{ C \}$ and $Cl(v_j) = \emptyset$ for $1 \leq j \leq n$ in Step 1.

As an induction hypothesis, assume during an execution of Step 2 that $\Sigma_{\mathsf{SC}} \vdash_{\{\text{A1,A2}\}} C(pd_i : C_i)$ for every $C_i \in Cl(v_i)$. Assume that a class C_b should be added to $Cl(v_j)$ by applying an SC-rule for $C_a(pd' : C_b) \in \Sigma_{\mathsf{SC}}$. Then by the definition of SC-rule, it holds that $C_b \notin Cl(v_j)$, $C_a \in Cl(v_k)$, and $pd' = P_{k+1}.P_{k+2}. \cdots .P_j$ for some k. We must prove that $\Sigma_{\mathsf{SC}} \vdash_{\{\text{A1,A2}\}} C(pd_j : C_b)$. By the induction hypothesis, $C_a \in Cl(v_k)$ implies that $\Sigma_{\mathsf{SC}} \vdash_{\{\text{A1,A2}\}} C(pd_k : C_a)$. By axiom A2, $C(pd_k : C_a)$ and $C_a(pd' : C_b)$ imply $C(pd_k \circ pd' : C_b)$. Note that $pd_k \circ pd' = pd_j$. Hence $\Sigma_{\mathsf{SC}} \vdash_{\{\text{A1,A2}\}} C(pd_j : C_b)$. This completes proving Lemma 5. □

Lemma 6: $pd \in \mathit{PathDescs}(C)$ if and only if $\mathit{Chase}_{\Sigma_{\mathtt{SC}}}(C, pd)$ satisfies a property value integrity condition.

Proof. <u>*If part.*</u> Induction on $\mathit{len}(pd)$.

Basis. If $\mathit{len}(pd) = 0$, that is $pd = \mathtt{Id}$, then clearly $pd \in \mathit{PathDescs}(C)$. Hence the basis holds.

Induction. As an induction hypothesis, assume that if $\mathit{len}(pd) \le n - 1$ and $\mathit{Chase}_{\Sigma_{\mathtt{SC}}}(C, pd)$ satisfies a property value integrity condition, then $pd \in \mathit{PathDescs}(C)$, where $n \ge 1$.

Assume that $\mathit{len}(pd) = n$ and $\mathit{Chase}_{\Sigma_{\mathtt{SC}}}(C, pd)$ satisfies a property value integrity condition. Then it holds that $\mathit{Cl}(v_{i-1}) \cap \mathit{Dom}(P_i) \ne \emptyset$ for $1 \le i \le n$. In particular, there is a class $C_{n-1} \in \mathit{Classes}(S)$ such that

$$C_{n-1} \in \mathit{Cl}(v_{n-1}) \cap \mathit{Dom}(P_n). \tag{2.1}$$

Let $G(V, A)$ be an interpretation satisfying $\Sigma \cup S_{\mathtt{FUNC}}$. Let u be a vertex in V such that $C \in \mathit{Cl}(u)$. In order to prove that $pd \in \mathit{PathDescs}(C)$, it suffices to show that there is a path in $G(V, A)$ from u described by pd. We claim that

$$pd_{n-1} \in \mathit{PathDescs}(C). \tag{2.2}$$

Since $\mathit{len}(pd_{n-1}) = n - 1$, it follows from the induction hypothesis that if $\mathit{Chase}_{\Sigma_{\mathtt{SC}}}(C, pd_{n-1})$ satisfies a property value integrity condition, then 2.2 holds. Thus it suffices to show that $\mathit{Chase}_{\Sigma_{\mathtt{SC}}}(C, pd_{n-1})$ satisfies a property value integrity condition. By Lemma 4(b), $\mathit{Chase}_{\Sigma_{\mathtt{SC}}}(C, pd_{n-1})$ coincides with the pd_{n-1}-List $v_0 \xrightarrow{P_1} v_1 \xrightarrow{P_2} \cdots \xrightarrow{P_{n-1}} v_{n-1}$. Furthermore, the pd_{n-1}-List satisfies a property value integrity condition, since $\mathit{Chase}_{\Sigma_{\mathtt{SC}}}(C, pd)$ satisfies that condition by assumption and pd_{n-1} is a prefix of pd. Hence 2.2 holds.

By 2.2, there must be a path in $G(V, A)$ from u to a vertex w described by pd_{n-1}.[6] We next claim that

$$\mathit{Cl}(w) \cap \mathit{Dom}(P_n) \ne \emptyset. \tag{2.3}$$

By 2.1, it suffices to show that $C_{n-1} \in \mathit{Cl}(w)$. Since $C_{n-1} \in \mathit{Cl}(v_{n-1})$ by 2.1, it follows from Lemma 5 that $\Sigma \vdash_{\{\mathrm{A1,A2}\}} C(pd_{n-1} : C_{n-1})$, and thus $\Sigma \models C(pd_{n-1} : C_{n-1})$ by Lemma 3. This implies that $G(V, A)$ satisfies $C(pd_{n-1} : C_{n-1})$, since $G(V, A)$ satisfies Σ. Note that $C \in \mathit{Cl}(u)$ and there is a path in $G(V, A)$ from u to w described by pd_{n-1}. Hence it must hold that $C_{n-1} \in \mathit{Cl}(w)$, which implies 2.3.

[6]Since $G(V, A)$ satisfies $S_{\mathtt{FUNC}}$, the vertex w is unique.

Since $G(V, A)$ satisfies $\mathtt{FUNC}(P_n) \in S_{\mathtt{FUNC}}$, it follows from 2.3 and property value completeness for $\mathtt{FUNC}(P_n)$ that there must be an arc $w \xrightarrow{P_n} w' \in A$. Furthermore, since there is a path in $G(V, A)$ from u to w described by pd_{n-1}, there is also a path in $G(V, A)$ from u to w' described by $pd_{n-1} \circ P_n$ $(= pd)$. This completes the induction proof of the if part.

Only if part. Assume that $Chase_{\Sigma_{\mathtt{SC}}}(C, pd)$ does not satisfy a property value integrity condition. (We will prove that $pd \notin PathDescs(C)$.) Then there is an index i such that

$$Cl(v_i) \cap Dom(P_{i+1}) = \emptyset, \tag{2.4}$$

where $0 \leq i \leq n-1$. Let i be the smallest index satisfying 2.4. For the pd_i-List $v_0 \xrightarrow{P_1} v_1 \xrightarrow{P_2} \cdots \xrightarrow{P_i} v_i$, construct the augmented graph $G(V, A)$ with respect to S. We will prove that $G(V, A)$ is an example showing that $pd \notin PathDescs(C)$. It suffices to show that (1) $G(V, A)$ is an interpretation satisfying $\Sigma \cup S_{\mathtt{FUNC}}$ and (2) there is no path in $G(V, A)$ from v_0 described by pd. Note that $C \in Cl(v_0)$.

By the minimality of i, it holds that $Cl(v_j) \cap Dom(P_{j+1}) \neq \emptyset$ for $0 \leq j \leq i-1$. Thus the pd_i-List satisfies a property value integrity condition. Furthermore, since the pd_i-List satisfies $\Sigma_{\mathtt{SC}}$ by Lemma 4(a), it follows from Lemma 2 that $G(V, A)$ is an interpretation satisfying $\Sigma \cup S_{\mathtt{FUNC}}$.

Clearly, there is a path in $G(V, A)$ from v_0 to v_i described by pd_i. Note that the path is unique, since $G(V, A)$ satisfies $S_{\mathtt{FUNC}}$. By Step 3 of Procedure 1, if there is an arc $v_i \xrightarrow{P} u \in A$, then $Cl(v_i) \cap Dom(P) \neq \emptyset$. Hence it follows from 2.4 that there is no arc $v_i \xrightarrow{P_{i+1}} u \in A$. That is, there is no path in $G(V, A)$ from v_0 described by $pd_i \circ P_{i+1}$. Since $pd_i \circ P_{i+1}$ is a prefix of pd, there is no path in $G(V, A)$ from v_0 described by pd, either. This completes proving the only if part. Hence Lemma 6 holds. □

Example 9: Consider $Chase_{\Sigma_{\mathtt{SC}}}(\mathtt{a}_2, \mathtt{B.A.C})$, given in Figure 7, for the database schema $\langle S, \Sigma \rangle$ of Example 5. Then $Chase_{\Sigma_{\mathtt{SC}}}(\mathtt{a}_2, \mathtt{B.A.C})$ does not satisfy a property value integrity condition, since $Cl(v_2) \cap Dom(\mathtt{C}) = \emptyset$. Thus $\mathtt{B.A.C} \notin PathDescs(\mathtt{a}_2)$ by Lemma 6. On the other hand, $Chase_{\Sigma_{\mathtt{SC}}}(\mathtt{a}_2, \mathtt{B.A})$ satisfies a property value integrity condition, since $\mathtt{a}_3 \in Cl(v_0) \cap Dom(\mathtt{B})$ and $\mathtt{a}_2 \in Cl(v_1) \cap Dom(\mathtt{A})$. Thus $\mathtt{B.A} \in PathDescs(\mathtt{a}_2)$ by Lemma 6. □

We are now ready to prove the following theorem.

Theorem 2: If $pd \in \mathit{PathDescs}(C)$, then the following three statements are equivalent.[7]

1. $\Sigma \models C(pd : C')$.

2. $\Sigma \vdash_{\{\mathrm{A1,A2}\}} C(pd : C')$.

3. $C' \in \mathit{Cl}(v_n)$.

Proof. By Lemma 3, (2) implies (1). By Lemma 5, (3) implies (2). We prove that (1) implies (3); that is, if $C' \notin \mathit{Cl}(v_n)$, then $\Sigma \not\models C(pd : C')$. By Theorem 1, it suffices to show that if $C' \notin \mathit{Cl}(v_n)$, then $\mathit{Chase}_{\Sigma_{\mathsf{SC}}}(C, pd)$ satisfies PDLs 1 to 3.

Since $pd \in \mathit{PathDescs}(C)$, PDL 1 follows from Lemma 6. PDL 2 follows from Lemma 4(a). Finally, since $C \in \mathit{Cl}(v_0)$ by definition, if $C' \notin \mathit{Cl}(v_n)$, then $\mathit{Chase}_{\Sigma_{\mathsf{SC}}}(C, pd)$ violates $C(pd : C')$; that is, PDL 3 follows. □

Finally, let us consider the case that pd is not necessarily in $\mathit{PathDescs}(C)$. Then we have the following theorem, which is a generalization of Theorem 2.

Theorem 3: The following three statements are equivalent.

1. $\Sigma \models C(pd : C')$.

2. $\Sigma \vdash_{\{\mathrm{A1,A2,A3}\}} C(pd : C')$.

3. At least one of the following two conditions holds.

   Condition A: $\Sigma \vdash_{\{\mathrm{A1,A2}\}} C(pd : C')$.

   Condition B: For some i such that $0 \leq i \leq n-1$,
   $\Sigma \vdash_{\{\mathrm{A1,A2}\}} C_i(P_{i+1}.P_{i+2}.\cdots.P_n : C')$ for every $C_i \in \mathit{Dom}(P_{i+1})$.

Proof. By Lemma 3, (2) implies (1). We next prove that (3) implies (2).

Clearly, Condition A implies (2). Assume that Condition B holds. Then by axiom A3, for every $C'' \in \mathit{Classes}(S)$ and every path description pd',
$\Sigma \vdash_{\{\mathrm{A1,A2,A3}\}} C''(pd' \circ P_{i+1}.P_{i+2}.\cdots.P_n : C')$.
In particular, by letting $C'' = C$ and $pd' = pd_i$,
$\Sigma \vdash_{\{\mathrm{A1,A2,A3}\}} C(pd : C')$.
Hence Condition B also implies (2). As a result, (3) implies (2).

Finally, we prove that (1) implies (3) by induction on $\mathit{len}(pd)$.

[7] Σ is not necessarily well-formed with respect to $\langle S, \Sigma \rangle$.

Basis. If $len(pd) = 0$, that is $pd = \texttt{Id}$, then clearly $pd \in PathDescs(C)$. Thus (1) implies Condition A by Theorem 2. Hence the basis holds.

Induction. As an induction hypothesis, assume that for any SC $C(pd : C')$, if $len(pd) \leq n-1$, then (1) implies (3). Assume that $len(pd) = n$. In the following we will prove that if $C(pd : C')$ satisfies neither Condition A nor B, then $\Sigma \not\models C(pd : C')$. (Thus it will turn out that (1) implies (3).)

Assume that $C(pd : C')$ satisfies neither Condition A nor B. The negation of Condition A is that

$$\Sigma \not\vdash_{\{\mathrm{A1,A2}\}} C(pd : C'). \tag{2.5}$$

The negation of Condition B is that for every i such that $0 \leq i \leq n-1$,

$$\Sigma \not\vdash_{\{\mathrm{A1,A2}\}} C_i(P_{i+1}.P_{i+2}.\cdots.P_n : C') \text{ for some } C_i \in Dom(P_{i+1}). \tag{2.6}$$

There are two cases to be considered: $C \in Dom(P_1)$ and $C \notin Dom(P_1)$.

Case 1. Assume that $C \in Dom(P_1)$. For simplicity, let $pd' = P_2.P_3.\cdots.P_n$. By 2.6, there is a class $C_1 \in Dom(P_2)$ such that

$$\Sigma \not\vdash_{\{\mathrm{A1,A2}\}} C_1(pd' : C'). \tag{2.7}$$

We claim that

$$\Sigma \not\models C_1(pd' : C'). \tag{2.8}$$

Since $len(pd') = n-1$, it follows from the induction hypothesis that if $C_1(pd' : C')$ satisfies neither Condition A nor B, then 2.8 holds. Hence it suffices to show that $C_1(pd' : C')$ satisfies neither Condition A nor B. By 2.7, $C_1(pd' : C')$ does not satisfy Condition A. Furthermore, by ignoring the case of $i = 0$ in 2.6, we notice that $C_1(pd' : C')$ does not satisfy Condition B, either. This completes proving 2.8.

By 2.8 and Theorem 1, there is a pd'-List $u_1 \xrightarrow{P_2} u_2 \xrightarrow{P_3} \cdots \xrightarrow{P_n} u_n$ satisfying PDLs 1 to 3. Let $w_0 \xrightarrow{P_1} w_1 \xrightarrow{P_2} \cdots \xrightarrow{P_n} w_n$ be a pd-List such that $Cl(w_0) = Cl(v_0)$ and $Cl(w_i) = Cl(u_i) \cup Cl(v_i)$ for $1 \leq i \leq n$. By Theorem 1, in order to prove that $\Sigma \not\models C(pd : C')$, it suffices to show that the pd-List $w_0 \xrightarrow{P_1} w_1 \xrightarrow{P_2} \cdots \xrightarrow{P_n} w_n$ satisfies PDLs 1 to 3.

As for PDL 1, it suffices to show that $Cl(w_{i-1}) \cap Dom(P_i) \neq \emptyset$ for $1 \leq i \leq n$. If $i = 0$, then $Cl(w_0) \cap Dom(P_1) \neq \emptyset$, since $C \in Cl(v_0) = Cl(w_0)$ and $C \in Dom(P_1)$. Assume that $i \geq 1$. Since the pd'-List $u_1 \xrightarrow{P_2} u_2 \xrightarrow{P_3} \cdots \xrightarrow{P_n} u_n$ satisfies PDL 1, it holds that $Cl(u_i) \cap Dom(P_{i+1}) \neq \emptyset$. Since $Cl(u_i) \subseteq Cl(w_i)$, it holds that $Cl(w_i) \cap Dom(P_{i+1}) \neq \emptyset$. Hence PDL 1 holds.

As for PDL 2, assume that there is an SC $C_a(P_j.P_{j+1}.\cdots.P_k : C_b) \in \Sigma_{\mathsf{SC}}$ such that $C_a \in Cl(w_j)$, where $0 \leq j \leq k \leq n$. It suffices to show that $C_b \in Cl(w_k)$. If $C_a \in Cl(v_j)$, then $C_b \in Cl(v_k) \subseteq Cl(w_k)$, since $Chase_{\Sigma_{\mathsf{SC}}}(C, pd)$ satisfies $C_a(P_j.P_{j+1}.\cdots.P_k : C_b) \in \Sigma_{\mathsf{SC}}$ by Lemma 4(a). On the other hand, if $C_a \in Cl(u_j)$, then $C_b \in Cl(u_k) \subseteq Cl(w_k)$, since the pd'-List $u_1 \xrightarrow{P_2} u_2 \xrightarrow{P_3} \cdots \xrightarrow{P_n} u_n$ satisfies $C_a(P_j.P_{j+1}.\cdots.P_k : C_b) \in \Sigma_{\mathsf{SC}}$ by PDL 2. Hence PDL 2 holds.

Finally, since the pd'-List $u_1 \xrightarrow{P_2} u_2 \xrightarrow{P_3} \cdots \xrightarrow{P_n} u_n$ satisfies PDL 3, it holds that $C' \notin Cl(u_n)$. Furthermore, $C' \notin Cl(v_n)$ by 2.5 and Lemma 5. Since $Cl(w_n) = Cl(v_n) \cup Cl(u_n)$, it holds that $C' \notin Cl(w_n)$. On the other hand, $C \in Cl(v_0)$ implies $C \in Cl(w_0)$. Hence PDL 3 holds. This completes Case 1.

Case 2. Assume that $C \notin Dom(P_1)$. By 2.6, there is a class $C_0 \in Dom(P_1)$ such that $\Sigma \not\vdash_{\{\mathrm{A1,A2}\}} C_0(pd : C')$. Since $C_0 \in Dom(P_1)$, it follows from Case 1 above that

$$\Sigma \not\vdash_{\{\mathrm{A1,A2}\}} C_0(pd : C') \text{ implies } \Sigma \not\models C_0(pd : C').$$

Thus by Theorem 1, there is a pd-List $u_0 \xrightarrow{P_1} u_1 \xrightarrow{P_2} \cdots \xrightarrow{P_n} u_n$ satisfying PDLs 1 to 3. Let $w_0 \xrightarrow{P_1} w_1 \xrightarrow{P_2} \cdots \xrightarrow{P_n} w_n$ be a pd-List such that $Cl(w_i) = Cl(u_i) \cup Cl(v_i)$ for $0 \leq i \leq n$. It can be proved in the same way as in Case 1 that the latter pd-List satisfies PDLs 1 to 3. Hence $\Sigma \not\models C(pd : C')$ by Theorem 1. This completes the induction proof that (1) implies (3). Consequently, Theorem 3 holds. □

2.3 Decision Procedures

In this section, we will prove the following theorem.

Theorem 4: The following three decision problems can be solved in $O(D \cdot (len(pd) + 1))$ time, where D is the size of database schema $\langle S, \Sigma \rangle$.

a. $pd \in PathDescs(C)$?

b. $pd \in PathFuncs(C)$?

c. $\Sigma \models C(pd : C')$? □

To say the conclusion first, the time for computing $Chase_{\Sigma_{\mathsf{SC}}}(C, pd)$ dominates the time complexities of the three decision problems. Let us present a procedure for computing $Chase_{\Sigma_{\mathtt{SC}}}(C, pd)$, which is a refinement of Procedure 2, and then estimate its computation time. In order to construct $Chase_{\Sigma_{\mathtt{SC}}}(C, pd)$, it suffices to compute $Cl(v_0), Cl(v_1), \cdots, Cl(v_n)$. The following procedure will compute them in the order of $0, 1, \cdots, n$.

Procedure 3: (Computing $Cl(v_0), Cl(v_1), \cdots, Cl(v_n)$.)
input: a database schema $\langle S, \Sigma \rangle$, a class $C \in Classes(S)$, and a path description pd $(= P_1.P_2. \cdots .P_n)$.

1. Divide $\Sigma_{\mathtt{SC}}$ into two sets: $\Sigma_{\mathtt{Id}} = \{C_a(pd' : C_b) \in \Sigma_{\mathtt{SC}} \mid pd' = \mathtt{Id}\}$ and $\Sigma_{\neg\mathtt{Id}} = \Sigma_{\mathtt{SC}} - \Sigma_{\mathtt{Id}}$.
2. Let $CL_0 \leftarrow \{C_0 \in Classes(S) \mid \Sigma_{\mathtt{Id}} \vdash_{\{\mathrm{A1,A2}\}} C(\mathtt{Id} : C_0)\}$.
3. **for** $i \leftarrow 1$ **to** n

   do begin

   4. Let $CL \leftarrow \{C_b \in Classes(S) \mid$ there is an SC $C_a(pd' : C_b) \in \Sigma_{\neg\mathtt{Id}}$ such that $pd' = P_{j+1}.P_{j+2}. \cdots .P_i$ and $C_a \in CL_j$ for some j $\}$.
   5. Let $CL_i \leftarrow \bigcup_{C_b \in CL}\{C_i \in Classes(S) \mid \Sigma_{\mathtt{Id}} \vdash_{\{\mathrm{A1,A2}\}} C_b(\mathtt{Id} : C_i)\}$.

   end □

Example 10: Consider the database schema $\langle S, \Sigma \rangle$ in Example 5. Let us execute Procedure 3 for class $\mathtt{a}_2$ and path description $\mathtt{B.A.C}$. In Step 1, $\Sigma_{\mathtt{SC}}$ is divided as follows:

$\Sigma_{\mathtt{Id}} = \{\mathtt{a}_1(\mathtt{Id} : \mathtt{a}_2), \mathtt{a}_2(\mathtt{Id} : \mathtt{a}_3), \mathtt{c}_1(\mathtt{Id} : \mathtt{c}_2)\}$

$\Sigma_{\neg\mathtt{Id}} = \{\mathtt{a}_1(\mathtt{C} : \mathtt{c}_2), \mathtt{a}_3(\mathtt{B} : \mathtt{b}), \mathtt{b}(\mathtt{A} : \mathtt{a}_2), \mathtt{e}(\mathtt{C} : \mathtt{c}_2)\}$

In Step 2, $CL_0 = \{\mathtt{a}_2, \mathtt{a}_3\}$, since (1) $\Sigma_{\mathtt{Id}} \vdash_{\mathrm{A1}} \mathtt{a}_2(\mathtt{Id} : \mathtt{a}_2)$ and (2) $\mathtt{a}_2(\mathtt{Id} : \mathtt{a}_2)$ and $\mathtt{a}_2(\mathtt{Id} : \mathtt{a}_3) \in \Sigma_{\mathtt{Id}}$ imply $\mathtt{a}_2(\mathtt{Id} : \mathtt{a}_3)$ by axiom A2; that is, $\Sigma_{\mathtt{Id}} \vdash_{\{\mathrm{A1,A2}\}} \mathtt{a}_2(\mathtt{Id} : \mathtt{a}_3)$. No other classes are added to CL_0.

Consider the for loop in Step 3. Let $i = 1$. In Step 4, for each $C_a(pd' : C_b) \in \Sigma_{\neg\mathtt{Id}}$, it is examined whether or not C_b should be added to CL. In this case, only for $\mathtt{a}_3(\mathtt{B} : \mathtt{b})$, class $\mathtt{b}$ is added to CL, since $\mathtt{a}_3 \in CL_0$. That is, $CL = \{\mathtt{b}\}$. In Step 5, $CL_1 = \{\mathtt{b}\}$, since there is no SC in $\Sigma_{\mathtt{Id}}$ of the form $\mathtt{b}(\mathtt{Id} : C)$ for any $C \in Classes(S)$. Similarly, for $i = 2$, we have $CL = \{\mathtt{a}_2\}$ and $CL_2 = \{\mathtt{a}_2, \mathtt{a}_3\}$. For $i = 3$, we have $CL = CL_3 = \emptyset$.

Consider $Chase_{\Sigma_{\mathtt{SC}}}(\mathtt{a}_2, \mathtt{B.A.C})$ $(= v_0 \xrightarrow{\mathtt{B}} v_1 \xrightarrow{\mathtt{A}} v_2 \xrightarrow{\mathtt{C}} v_3)$, which was constructed in Example 8. Then it holds that $Cl(v_i) = CL_i$ for all i. □

The correctness of Procedure 3 follows from the following lemma.

Lemma 7: $CL_i = Cl(v_i)$ for $0 \leq i \leq n$.

Proof. Induction on i.

Basis. Consider the case that $i = 0$. Assume that a class C_b should be added to $Cl(v_0)$ by applying an SC-rule for $C_a(pd' : C_b) \in \Sigma_{\mathtt{SC}}$ in Step 2 of Procedure 2. Then clearly, $pd' = \mathtt{Id}$, that is, $C_a(pd' : C_b) \in \Sigma_{\mathtt{Id}}$. Thus $\Sigma_{\mathtt{SC}} \vdash_{\{A1,A2\}} C(\mathtt{Id} : C_0)$ if and only if $\Sigma_{\mathtt{Id}} \vdash_{\{A1,A2\}} C(\mathtt{Id} : C_0)$. Hence it follows from Lemma 5 that $CL_0 = Cl(v_0)$. That is, the basis holds.

Induction. As an induction hypothesis, assume that if $j \leq i - 1$, then $CL_j = Cl(v_j)$, where $i \geq 1$. By Lemma 5, it suffices to show that

$$CL_i = \{C_i \in Classes(S) \mid \Sigma_{\mathtt{SC}} \vdash_{\{A1,A2\}} C(pd_i : C_i)\}. \tag{2.1}$$

Proof of '$\subseteq$': We prove that if $C_i \in CL_i$, then $\Sigma_{\mathtt{SC}} \vdash_{\{A1,A2\}} C(pd_i : C_i)$. Let $C_i \in CL_i$. There are two cases to be considered: $C_i \in CL$ and $C_i \notin CL$.

Case 1. Assume that $C_i \in CL$. By definition, there is an SC $C_a(pd' : C_i) \in \Sigma_{\neg\mathtt{Id}}$ such that $pd' = P_{j+1}.P_{j+2}. \cdots .P_i$ and $C_a \in CL_j$ for some j. Since $pd' \neq \mathtt{Id}$ by the definition of $\Sigma_{\neg\mathtt{Id}}$, it holds that $j \leq i - 1$. Thus $C_a \in CL_j$ implies $C_a \in Cl(v_j)$ by the induction hypothesis. Hence $\Sigma_{\mathtt{SC}} \vdash_{\{A1,A2\}} C(pd_j : C_a)$ by Lemma 5. By axiom A2, $C(pd_j : C_a)$ and $C_a(pd' : C_i)$ imply $C(pd_i : C_i)$, where $pd_j \circ pd' = pd_i$. That is, $\Sigma_{\mathtt{SC}} \vdash_{\{A1,A2\}} C(pd_i : C_i)$.

Case 2. Assume that $C_i \notin CL$. By the definition of CL_i, the assumption implies that $\Sigma_{\mathtt{Id}} \vdash_{\{A1,A2\}} C_b(\mathtt{Id} : C_i)$ for some $C_b \in CL$. Since $C_b \in CL$, it follows from Case 1 above that $\Sigma \vdash_{\{A1,A2\}} C(pd_i : C_b)$. By axiom A2, $C(pd_i : C_b)$ and $C_b(\mathtt{Id} : C_i)$ imply $C(pd_i : C_i)$, since $pd_i \circ \mathtt{Id} = pd_i$. That is, $\Sigma_{\mathtt{SC}} \vdash_{\{A1,A2\}} C(pd_i : C_i)$. This completes the proof of '$\subseteq$'.

Proof of '$\supseteq$': We prove that if $\Sigma_{\mathtt{SC}} \vdash_{\{A1,A2\}} C(pd_i : C_i)$, then $C_i \in CL_i$. Assume that $\Sigma_{\mathtt{SC}} \vdash_{\{A1,A2\}} C(pd_i : C_i)$. Then $\Sigma_{\mathtt{SC}} \models C(pd_i : C_i)$ by Lemma 3. Let $u_0 \xrightarrow{P_1} u_1 \xrightarrow{P_2} \cdots \xrightarrow{P_i} u_i$ be a pd_i-List such that $Cl(u_j) = CL_j$ for $0 \leq j \leq i$. We will prove that the pd_i-List satisfies $\Sigma_{\mathtt{SC}}$. Since $C \in CL_0$, this will imply that $C_i \in CL_i$.

Since $CL_j = Cl(v_j)$ for $0 \leq j \leq i-1$ by the induction hypothesis, it follows from Lemma 4(a) that the pd_{i-1}-List $u_0 \xrightarrow{P_1} u_1 \xrightarrow{P_2} \cdots \xrightarrow{P_i} u_{i-1}$ satisfies $\Sigma_{\mathtt{SC}}$. By Step 4, for every SC $C_a(pd' : C_b) \in \Sigma_{\neg\mathtt{Id}}$, if $pd' = P_{j+1}.P_{j+2}. \cdots .P_i$ and $C_a \in CL_j$, then $C_b \in CL_i$. Thus the pd_i-List satisfies $\Sigma_{\neg\mathtt{Id}}$. Similarly, by Step 5, for every SC $C_a(\mathtt{Id} : C_b) \in \Sigma_{\mathtt{Id}}$, if $C_a \in CL_i$, then $C_b \in CL_i$. Thus the

pd_i-List also satisfies $\Sigma_{\mathtt{Id}}$. Since $\Sigma_{\mathtt{SC}} = \Sigma_{\neg\mathtt{Id}} \cup \Sigma_{\mathtt{Id}}$ by definition, the pd_i-List satisfies $\Sigma_{\mathtt{SC}}$. This completes the induction proof. Consequently, Lemma 7 holds. □

Now consider the time complexity of Procedure 3. Let *Classes*(S) consist of K classes. (Note that $K \leq D$.) We use bit arrays of size K to represent arbitrary subsets of *Classes*(S), such as those denoted by the variables CL or CL_i which are used in the procedure. Testing for class membership or inserting a class into a given subset can then be executed in constant time. Also, each of these variables can then be initialized to the empty set in $O(K)$ time.

Clearly, Step 1 runs in $O(\|\Sigma\|)$ time, where $\|\Sigma\|$ is the size of Σ, and Step 2 requires $O(K)$ time to initialize CL_0. By the definition of $\Sigma_{\mathtt{Id}}$, each application of axiom A2 must be of the form: 'if $C_1(\mathtt{Id} : C_2)$ and $C_2(\mathtt{Id} : C_3)$, then $C_1(\mathtt{Id} : C_3)$.' That is, axiom A2 is considered as a *transitivity rule*. Clearly, axiom A1 is considered as a *reflexivity rule*. Thus CL_0 must coincide with the *reflexive transitive closure* of C with respect to $\Sigma_{\mathtt{Id}}$. Hence CL_0 can be computed by a usual algorithm for computing a reflexive transitive closure. In fact, CL_0 can be computed in $O(\|\Sigma_{\mathtt{Id}}\|)$ time. Since $\|\Sigma_{\mathtt{Id}}\| \leq D$, Step 2 can be executed in $O(D)$ time.

In Step 4, it takes $O(K)$ time to initialize CL. In order to compute CL, it suffices to test *once* for each $C_a(pd' : C_b) \in \Sigma_{\neg\mathtt{Id}}$ whether or not $pd' = P_{j+1}.P_{j+2}. \cdots .P_i$ and $C_a \in CL_j$ for some j. Testing $pd' = P_{j+1}.P_{j+2}. \cdots .P_i$ can be done in $O(len(pd'))$ time. Testing $C_a \in CL_j$ can be done in constant time. If both conditions holds, then C_b is inserted into CL, which can be done in constant time. That is, for each $C_a(pd' : C_b)$, it takes $O(len(pd'))$ time. Since $\sum_{C_a(pd':C_b)\in\Sigma_{\neg\mathtt{Id}}} len(pd') \leq \|\Sigma_{\neg\mathtt{Id}}\|$, it can be done in $O(\|\Sigma_{\neg\mathtt{Id}}\|)$ time as a whole. Since $\|\Sigma_{\neg\mathtt{Id}}\| \leq D$, Step 4 can be executed in $O(D)$ time.

In Step 5, it takes $O(K)$ time to initialize CL_i. As in Step 2, CL_i coincides with the reflexive transitive closure of CL with respect to $\Sigma_{\mathtt{Id}}$, and can be computed in $O(\|\Sigma_{\mathtt{Id}}\|)$ time. Thus Step 5 can be executed in $O(D)$ time.

As a result, one execution of Steps 4 and 5 can be done in $O(D)$ time. Since $n = len(pd)$, the for loop in Step 3 can be executed in $O(D \cdot len(pd))$ time as a whole. Consequently, we have the following lemma.

Lemma 8: $Chase_{\Sigma_{\mathtt{SC}}}(C, pd)$ can be computed in $O(D \cdot (len(pd) + 1))$ time. □

Since it follows from Lemma 6 that $pd \in PathDescs(C)$ if and only if $Cl(v_{i-1}) \cap Dom(P_i) \neq \emptyset$ for $1 \leq i \leq n$, it can be decided in $O(\|Chase_{\Sigma_{\mathtt{SC}}}(C, pd)\|)$

time whether or not $pd \in \mathit{PathDescs}(C)$. Hence Theorem 4(a) follows from Lemma 8.

The following lemma implies that it can be decided in $O(\|\mathit{Chase}_{\Sigma_{\mathtt{SC}}}(C, pd)\| + \|\Sigma_{\mathtt{FUNC}}\|)$ time whether or not $pd \in \mathit{PathFuncs}(C)$. Note that $\|\Sigma_{\mathtt{FUNC}}\| \leq D$. Hence Theorem 4(b) follows from Lemma 8.

Lemma 9: $pd \in \mathit{PathFuncs}(C)$ if and only if $pd \in \mathit{PathDescs}(C)$ and $\mathtt{FUNC}(P_i) \in \Sigma_{\mathtt{FUNC}}$ for $1 \leq i \leq n$.

Proof. *If part.* Assume that (1) $pd \in \mathit{PathDescs}(C)$ and (2) $\mathtt{FUNC}(P_i) \in \Sigma_{\mathtt{FUNC}}$ for $1 \leq i \leq n$. Let $G(V, A)$ be an interpretation satisfying Σ. Let u be a vertex in V such that $C \in \mathit{Cl}(u)$. In order to prove that $pd \in \mathit{PathFuncs}(C)$, it suffices to show that there is a unique path in $G(V, A)$ from u described by pd. Since $\Sigma = \Sigma \cup \{\mathtt{FUNC}(P_i) \mid 1 \leq i \leq n\}$ by assumption 2, $G(V, A)$ satisfies $\Sigma \cup \{\mathtt{FUNC}(P_i) \mid 1 \leq i \leq n\}$. Hence assumption 1 implies that there must be a path in $G(V, A)$ from u described by pd. (Strictly, $G(V, A)$ should satisfy not $\Sigma \cup \{\mathtt{FUNC}(P_i) \mid 1 \leq i \leq n\}$ but $\Sigma \cup S_{\mathtt{FUNC}}$ according to the definition of $\mathit{PathDescs}(C)$. Each FUNC constraint not in $\{\mathtt{FUNC}(P_i) \mid 1 \leq i \leq n\}$, however, is independent of the path described by pd.) Furthermore, assumption 2 implies that such a path must be unique.

Only if part. Assume that $pd \in \mathit{PathFuncs}(C)$. Then it holds that $pd \in \mathit{PathDescs}(C)$, since $\mathit{PathFuncs}(C) \subseteq \mathit{PathDescs}(C)$ by definition. We next prove that $\mathtt{FUNC}(P_i) \in \Sigma_{\mathtt{FUNC}}$ for $1 \leq i \leq n$.

Assume contrary that $\mathtt{FUNC}(P_i) \notin \Sigma_{\mathtt{FUNC}}$ for some i. Let $G(V, A)$ be the augmented graph of the pd_{i-1}-List $v_0 \xrightarrow{P_1} v_1 \xrightarrow{P_2} \cdots \xrightarrow{P_{i-1}} v_{i-1}$ with respect to S. It suffices to prove that $G(V, A)$ is an example showing that $pd \notin \mathit{PathFuncs}(C)$. It suffices to show that (1) $G(V, A)$ is an interpretation satisfying Σ and (2) there is no path in $G(V, A)$ from v_0 described by pd. Note that $C \in \mathit{Cl}(v_0)$.

Since $pd \in \mathit{PathDescs}(C)$, $\mathit{Chase}_{\Sigma_{\mathtt{SC}}}(C, pd)$ satisfies a property value integrity condition by Lemma 6. Thus so does the pd_{i-1}-List. Furthermore, the pd_{i-1}-List satisfies $\Sigma_{\mathtt{SC}}$ by Lemma 4(a). Hence $G(V, A)$ is an interpretation satisfying Σ $(\subseteq \Sigma \cup S_{\mathtt{FUNC}})$ by Lemma 2.

By Step 3 of Procedure 1, if there is an arc $v_{i-1} \xrightarrow{P} u \in A$, then $\mathtt{FUNC}(P) \in \Sigma_{\mathtt{FUNC}}$. Since $\mathtt{FUNC}(P_i) \notin \Sigma_{\mathtt{FUNC}}$, there is no arc $v_{i-1} \xrightarrow{P_i} u \in A$. Thus there is no path in $G(V, A)$ from v_0 described by $pd_{i-1} \circ P_i$. Since

$pd_{i-1} \circ P_i$ is a prefix of pd, there is no path in $G(V, A)$ from v_0 described by pd, either. This completes the only if part proof. □

Finally, let us consider Theorem 4(c). We must show that it can be decided in $O(D \cdot (len(pd) + 1))$ time whether or not $\Sigma \models C(pd : C')$. By Theorem 2, if $pd \in PathDescs(C)$, then $\Sigma \models C(pd : C')$ if and only if $C' \in Cl(v_n)$. Hence by Lemma 8, it can be decided in $O(D \cdot (len(pd) + 1))$ time. Theorem 2, however, does not apply to the case that $pd \notin PathDescs(C)$. In general, by Theorem 3, $\Sigma \models C(pd : C')$ if and only if $C(pd : C')$ satisfies either Condition A or B. In the following, let us consider how to decide whether or not $C(pd : C')$ satisfies either Condition A or B.

Since $C(pd : C')$ satisfies Condition A if and only if $C' \in Cl(v_n)$ by Lemma 5, Condition A can be tested in $O(\|Chase_{\Sigma_{SC}}(C, pd)\|)$ time, and hence in $O(D \cdot (len(pd) + 1))$ time by Lemma 8.

It remains to show that Condition B can be tested in $O(D \cdot (len(pd) + 1))$ time. A naive method for Condition B is to test whether or not $\Sigma \vdash_{\{A1,A2\}} C_i(P_i.P_{i+1}. \cdots .P_n : C')$ for every i and every $C_i \in Dom(P_i)$. It, however, takes exponential time in the worst case. There is a tricky way for testing Condition B.

For $0 \leq i \leq n-1$, let us define

$$\mathcal{CL}_i = \{C_i \in Classes(S) \mid \Sigma \vdash_{\{A1,A2\}} C_i(P_{i+1}.P_{i+2}. \cdots .P_n : C')\}.$$

Then $C(pd : C')$ satisfies Condition B if and only if $Dom(P_{i+1}) \subseteq \mathcal{CL}_i$ for some i. We consider how to compute $\mathcal{CL}_0, \mathcal{CL}_1, \cdots, \mathcal{CL}_{n-1}$. Note the analogy between $Cl(v_i)$ and $\mathcal{CL}_i$, where

$$Cl(v_i) = \{C_i \in Classes(S) \mid \Sigma \vdash_{\{A1,A2\}} C(P_1.P_2. \cdots .P_i : C_i)\}.$$

Hence $\mathcal{CL}_i$ can be obtained by executing Procedure 2 in the *reverse* direction as follows:

Procedure 4: (Computing $\mathcal{CL}_0, \mathcal{CL}_1, \cdots, \mathcal{CL}_{n-1}$.)
input: a database schema $\langle S, \Sigma \rangle$, a class $C' \in Classes(S)$, and a path description pd $(= P_1.P_2. \cdots .P_n.)$

1. Construct a pd-List $u_0 \xrightarrow{P_1} u_1 \xrightarrow{P_2} \cdots \xrightarrow{P_n} u_n$ such that $Cl(u_n) = \{ C' \}$ and $Cl(u_i) = \emptyset$ for $0 \leq i \leq n-1$.[8]

[8] Note that $Cl(v_0) = \{ C \}$ and $Cl(v_n) = \emptyset$ in Procedure 2.

2. Apply the following rule to the *pd*-List exhaustively:

   Reverse-SC-rule: For an SC $C_a(pd' : C_b) \in \Sigma_{\mathtt{SC}}$, if there are two vertices u_i, u_j such that $C_a \notin Cl(u_i)$, $C_b \in Cl(u_j)$, and $pd' = P_{i+1}.P_{i+2}.\cdots.P_j$, then add C_a to $Cl(u_i)$.[9]

3. Let $\mathcal{CL}_i \leftarrow Cl(u_i)$ for $0 \le i \le n-1$. □

Example 11: For the database schema $\langle S, \Sigma \rangle$ in Example 5, let us decide whether or not $\Sigma \models \mathtt{a}_2(\mathtt{B.A.C} : \mathtt{c}_2)$. $Chase_{\Sigma_{\mathtt{SC}}}(\mathtt{a}_2, \mathtt{B.A.C})$ is given in Figure 7. Since $\mathtt{c}_2 \notin Cl(v_3)$, Condition A does not hold for $\mathtt{a}_2(\mathtt{B.A.C} : \mathtt{c}_2)$. By Example 9, $\mathtt{B.A.C} \notin PathDescs(\mathtt{a}_2)$. Thus it must be decided whether or not Condition B holds for $\mathtt{a}_2(\mathtt{B.A.C} : \mathtt{c}_2)$. For SC $\mathtt{a}_2(\mathtt{B.A.C} : \mathtt{c}_2)$, let us compute $\mathcal{CL}_0, \mathcal{CL}_1, \mathcal{CL}_2$ by Procedure 4.

In Step 1, a ‘B.A.C’-List $u_0 \xrightarrow{\mathtt{B}} u_1 \xrightarrow{\mathtt{A}} u_2 \xrightarrow{\mathtt{C}} u_3$ is constructed, where $Cl(u_3) = \{\, \mathtt{c}_2 \,\}$ and $Cl(u_i) = \emptyset$ for $0 \le i \le 2$.

For SC $\mathtt{e(C : c_2)}$ in $\Sigma_{\mathtt{SC}}$, since $\mathtt{c}_2 \in Cl(u_3)$ and $\mathtt{e} \notin Cl(u_2)$, class e is added to $Cl(u_2)$ by applying the Reverse-SC-rule for $\mathtt{e(C : c_2)}$. By applying the Reverse-SC-rule for $\mathtt{c_1(Id : c_2)}$ in $\Sigma_{\mathtt{SC}}$, class $\mathtt{c}_1$ is added to $Cl(u_3)$, since $\mathtt{c}_2 \in Cl(u_3)$ and $\mathtt{c}_1 \notin Cl(u_3)$. By applying the Reverse-SC-rule for $\mathtt{a_1(C : c_2)}$ in $\Sigma_{\mathtt{SC}}$, class $\mathtt{a}_1$ is added to $Cl(u_2)$, since $\mathtt{c}_2 \in Cl(u_3)$ and $\mathtt{a}_1 \notin Cl(u_2)$. After that, no Reverse-SC-rule can be applied to the ‘B.A.C’-List any more. Thus we obtain the ‘B.A.C’-List given in Figure 8; that is, $\mathcal{CL}_0 = Cl(u_0) = \emptyset$, $\mathcal{CL}_1 = Cl(u_1) = \emptyset$, and $\mathcal{CL}_2 = Cl(u_2) = \{\mathtt{a}_1, \mathtt{e}\}$.

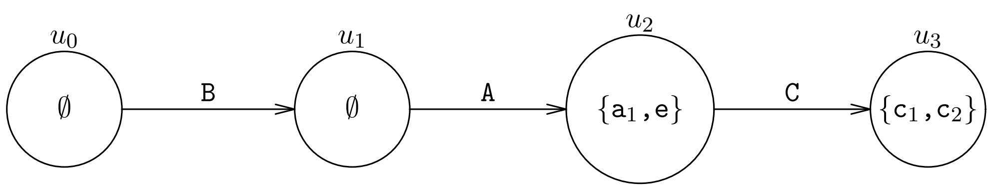

Figure 8: Computing $\mathcal{CL}_0, \mathcal{CL}_1, \mathcal{CL}_2$ by Procedure 4.

Note that $Dom(\mathtt{C}) = \{\mathtt{a}_1, \mathtt{e}\} \subseteq \mathcal{CL}_2$; that is, Condition B holds for SC $\mathtt{a}_2(\mathtt{B.A.C} : \mathtt{c}_2)$. Thus $\Sigma \models \mathtt{a}_2(\mathtt{B.A.C} : \mathtt{c}_2)$ by Theorem 3. In fact, it was shown in Example 7 that $\mathtt{a_2(B.A.C : c_2)}$ is derived from Σ by axiom A3. □

The correctness of Procedure 4 can be proved along the same line as proving Lemma 5. Furthermore, it can also be proved along the same line as

[9] Note that $C_a \in Cl(v_i)$ and $C_b \notin Cl(v_j)$ in SC-Rule of Procedure 2.

proving Lemma 8 that Procedure 4 can be executed in $O(D \cdot (len(pd) + 1))$ time. Thus it can be decided in that time whether or not $Dom(P_{i+1}) \subseteq \mathcal{CL}_i$ for some i. Hence Condition B can also be tested in that time. Consequently, Theorem 4(c) follows. By the discussions above, we have the following procedure that returns YES if and only if $\Sigma \models C(pd : C')$.

Procedure 5: (Deciding whether or not $\Sigma \models C(pd : C')$.)
input: a database schema $\langle S, \Sigma \rangle$ and an SC $C(pd : C')$.

1. Execute Procedure 3 to get $Cl(v_n)$.
2. **if** $C' \in Cl(v_n)$ **then** return YES

   else begin

   3. Execute Procedure 4 to get $\mathcal{CL}_0, \mathcal{CL}_1, \cdots, \mathcal{CL}_{n-1}$.
   4. **if** $Dom(P_{i+1}) \subseteq \mathcal{CL}_i$ for some i **then** return YES **else** return NO.

   end. □

By Theorems 2 and 3, if $C' \in Cl(v_n)$, then $\Sigma \models C(pd : C')$, no matter whether or not $pd \in PathDescs(C)$. The test is done in Step 2. If $C' \notin Cl(v_n)$, that is, $C(pd : C')$ does not satisfy Condition A, then it is tested in Step 4 whether or not $C(pd : C')$ satisfies Condition B.

3. The Most Specialized Class Rule (MSC)

Most object-oriented data models and many semantic data models impose an additional condition on a database that requires each object to be created with respect to one *particular* class. For example, in the case of the UNIVERSITY database schema in Figure 1, this would preclude the possibility of there existing an object in both the `student` and `prof` classes. As one might expect, limiting our notion of an interpretation in an analogous fashion will affect the various membership problems. In this section, we begin to explore these problems when imposing such a condition on interpretations. The condition, called the *most specialized class* rule, is formally defined as follows.

The most specialized class rule: Let $G(V, A)$ be an interpretation for S. For a vertex $v \in V$, if there is a class $C_1 \in Cl(v)$ such that $\Sigma \models C_1(\mathtt{Id} : C_2)$ for every $C_2 \in Cl(v)$, then the class C_1 is called the *most specialized class* (MSC) of v, denoted $Msc(v)$. $G(V, A)$ satisfies the most specialized class rule (MSC) with respect to Σ if and only if for every $v \in V$, whenever $Cl(v) \neq \emptyset$, then there exists $Msc(v)$. □

For example, the interpretation in Figure 3 for the ALGEBRA schema satisfies MSC with respect to the set of constraints in Table 1.

A constraint σ is a logical consequence of Σ *satisfying MSC*, written $\Sigma \models_{\text{MSC}} \sigma$, if any interpretation satisfying *MSC* as well as Σ must satisfy σ. We also write $PathDescs_{\text{MSC}}(C)$ and $PathFuncs_{\text{MSC}}(C)$ to denote the sets of well-formed path descriptions and path functions for a class C, respectively, in which only interpretations satisfying MSC are considered. Since every interpretation satisfying MSC is also a usual interpretation, it holds that:

$$\Sigma \models \sigma \text{ implies } \Sigma \models_{\text{MSC}} \sigma \tag{3.1}$$

$$PathDescs(C) \subseteq PathDescs_{\text{MSC}}(C) \tag{3.2}$$

$$PathFuncs(C) \subseteq PathFuncs_{\text{MSC}}(C) \tag{3.3}$$

In the remainder of this section, we make an additional assumption about a database schema $\langle S, \Sigma \rangle$, beyond the above requirement that interpretations satisfy MSC; we shall assume S contains a unique *bottom* class, written $\bot$, satisfying

$$\Sigma \models \bot(\mathtt{Id} : C) \text{ for every } C \in Classes(S). \tag{3.4}$$

This implies, for example, that $\bot$ qualifies as the *most specialized* class in $Classes(S)$.

Our assumption concerning $\bot$ is really our means of avoiding issues relating to *schema evaluation*, which are beyond the scope this paper. For example, consider the database schema illustrated in Figure 9. Note that an object u in class `a` must have an `A` property value to some object v in class `b`. A reasonable grounds for schema well-formedness might be to require that v may also be in class `c`, since `c` is a subclass of class `b`. However, this is not possible if classes `d` and `e` do not have a common subclass—if there is no bottom class $\bot$, for example. Our assumption about the existence of $\bot$ is a sufficient (but not necessary) condition for avoiding this sort of problem. In particular, it is relatively straightforward to derive the following version of Theorem 1 with regard to finite implication.

Theorem 5: The following three statements are equivalent.

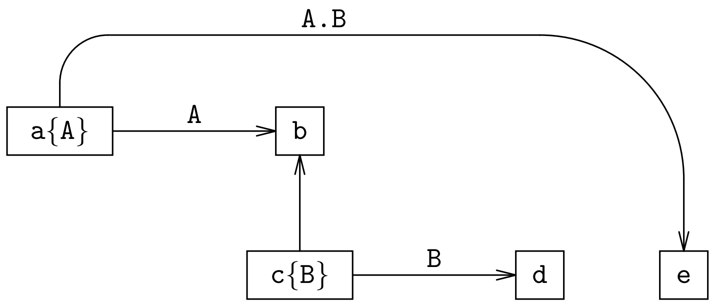

Figure 9: The need for $\perp$.

1. $\Sigma \not\models_{\text{MSC}} C(pd : C')$.
2. $\Sigma \not\models^{\text{finite}}_{\text{MSC}} C(pd : C')$.
3. There is a *pd*-List satisfying MSC as well as PDLs 1 to 3.

Proof. By definition, (2) implies (1). It can be proved along the same line as proving Lemma 1 that (1) implies (3). We prove that (3) implies (2).

Assume that there is a *pd*-List $v_0 \xrightarrow{P_1} v_1 \xrightarrow{P_2} \cdots \xrightarrow{P_n} v_n$ satisfying MSC as well as PDLs 1 to 3, where $pd = P_1.P_2.\cdots.P_n$. Let $G(V, A)$ be the augmented graph of the *pd*-List with respect to S. Note that $G(V, A)$ is constructed from the *pd*-List by adding one vertex u and a number of arcs. Since $\perp \in Classes(S) = Cl(u)$ by definition, it follows from 3.4 that $Msc(u) = \perp$, and thus $G(V, A)$ satisfies MSC. Furthermore, it can be proved along the same line as proving Lemma 2 that $G(V, A)$ is a finite interpretation satisfying $\Sigma \cup S_{\texttt{FUNC}}$ but violating $C(pd : C')$. Hence $\Sigma \not\models^{\text{finite}}_{\text{MSC}} C(pd : C')$. That is, (3) implies (2). $\square$

Example 12: Let $\langle S, \Sigma \rangle$ be a database schema illustrated in Figure 10, where $Dom(\texttt{A}) = \{\texttt{a}_1\}$, $Dom(\texttt{B}) = \{\texttt{a}_3\}$, $Dom(\texttt{C}) = \{\texttt{b}_2\}$, $Dom(\texttt{D}) = \{\texttt{b}_3\}$. Assume that $\Sigma_{\texttt{FUNC}} = \emptyset$. Then it holds that $\Sigma \not\models_{\text{MSC}} \texttt{a}_1\texttt{(A.B.C : c}_1\texttt{)}$. In fact, for the SC $\texttt{a}_1\texttt{(A.B.C : c}_1\texttt{)}$, there is an 'A.B.C'-List $v_0 \xrightarrow{\texttt{A}} v_1 \xrightarrow{\texttt{B}} v_2 \xrightarrow{\texttt{C}} v_3$ satisfying MSC as well as PDLs 1 to 3, as is given in Figure 11. Here, each vertex v_i is labeled $Msc(v_i)$ instead of $Cl(v_i)$ in order to clarify that the 'A.B.C'-List satisfies MSC. Since $Cl(v_i) = \{C \in Classes(S) \mid \Sigma \models Msc(v_i)(\texttt{Id} : C)\}$ by definition, $Cl(v_i)$ can be computed from $Msc(v_i)$. It holds that $Cl(v_0) = \{\texttt{a}_1\}$, $Cl(v_1) = \{\texttt{a}_1, \texttt{a}_2, \texttt{a}_3\}$, $Cl(v_2) = \{\texttt{b}_1, \texttt{b}_2, \texttt{b}_4\}$, and $Cl(v_3) = \{\texttt{c}_2\}$.

The augmented graph $G(V, A)$ of the 'A.B.C'-List with respect to S is given in Figure 12. Note that $G(V, A)$ is a finite interpretation satisfying MSC

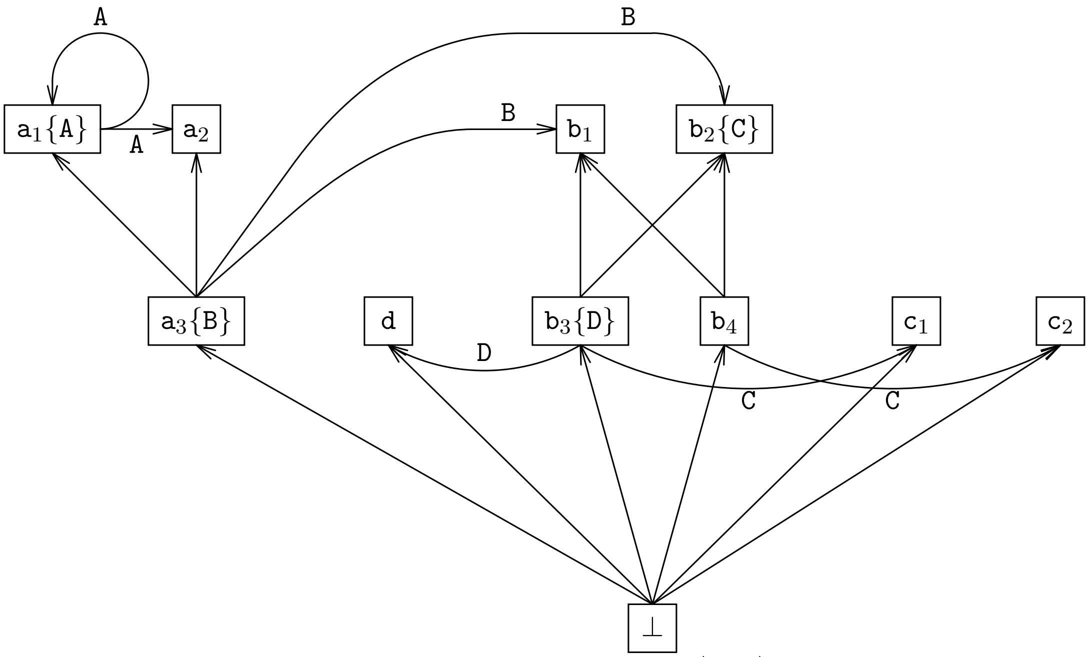

Figure 10: A database schema $\langle S, \Sigma \rangle$.

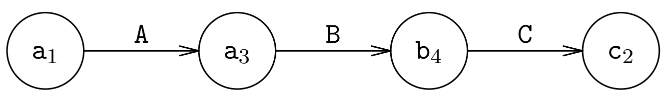

Figure 11: An '`A.B.C`'-List satisfying MSC.

as well as PDLs 1 to 3. Thus it holds that $\Sigma \not\models^{\text{finite}}_{\text{MSC}} \mathtt{a}_1\mathtt{(A.B.C : c}_1\mathtt{)}$. □

3.1 NP-completeness results

In this section, we will prove the following theorem.

Theorem 6: The following three decision problems are NP-complete.

a. $\Sigma \not\models_{\text{MSC}} C(pd : C')$? (It is still NP-complete, even if $C(pd : C')$ is well-formed with respect to $\langle S, \Sigma \rangle$.)

b. $pd \notin \mathit{PathDescs}_{\text{MSC}}(C)$?

c. $pd \notin \mathit{PathFuncs}_{\text{MSC}}(C)$? □

Theorem 6(a) implies that axioms A1 to A3 are no longer complete for deciding $\Sigma \models_{\text{MSC}} C(pd : C')$, though the axioms are still sound. Theorem 6(a) follows from the following two lemmas.

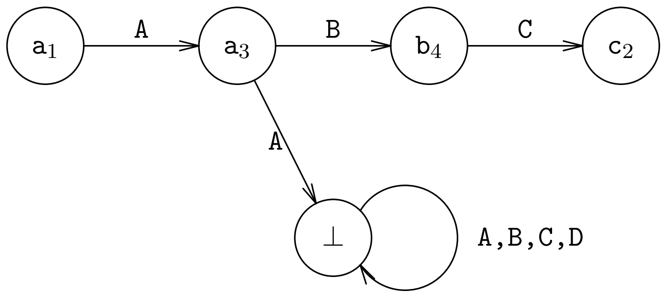

Figure 12: The augmented graph of the ‘A.B.C’-List in Figure 11.

Lemma 10: It is in NP to decide whether or not $\Sigma \not\models_{\text{MSC}} C(pd : C')$.

Proof. By Theorem 5, $\Sigma \not\models_{\text{MSC}} C(pd : C')$ if and only if there is a *pd*-List satisfying MSC as well as PDLs 1 to 3. Since the size of a *pd*-List is at most $D \cdot len(pd)$, where D is the size of $\langle S, \Sigma \rangle$, such a *pd*-List can be guessed in NP time. After that, it can be tested in (deterministic) polynomial time whether or not the *pd*-List satisfies MSC and PDLs 1 to 3. Hence Lemma 10 holds. □

Lemma 11: It is NP-hard to decide whether or not $\Sigma \not\models_{\text{MSC}} C(pd : C')$, even if $C(pd : C')$ is well-formed with respect to $\langle S, \Sigma \rangle$.

Proof. As an NP-complete problem, which will be transformed to the present problem, we consider the *dual* problem of 3-Satisfiability problem (3SAT). The problem is defined as follows:

A Boolean expression E is in *3DNF* if it is in disjunctive normal form such that each term consists of exactly three literals. The problem is: “Is a 3DNF Boolean expression E *not* a tautology; that is, is there a truth assignment that makes E false?” Since 3SAT is NP-complete, so is this problem.

Let $E = t_1 \vee t_2 \vee \cdots \vee t_m$ be a 3DNF Boolean expression over a set of variables $\{x_1, x_2, \cdots, x_n\}$. For $1 \leq j \leq m$, let us denote the three literals in t_j by $l_{j_1}, l_{j_2}, l_{j_3}$. That is, t_j is of the form $l_{j_1} l_{j_2} l_{j_3}$. We must construct, in polynomial time with respect to E, a database schema $\langle S, \Sigma \rangle$ and an SC $C(pd : C')$, which is well-formed with respect to $\langle S, \Sigma \rangle$, such that $\Sigma \not\models_{\text{MSC}} C(pd : C')$ if and only if E is not a tautology.

1. *The definition of S*: S contains the following $4n + 8m + 2$ class names:

$$C,\ Z,\ \{A_i, B_i, X_i, \overline{X}_i \mid 1 \leq i \leq n\},\ \ \text{and}$$

$$\{C_j, T_j, L_{j_k}, M_{j_k} \mid 1 \leq j \leq m \text{ and } 1 \leq k \leq 3\}$$

X_i and $\overline{X}_i$ correspond to x_i and $\overline{x}_i$, respectively. L_{j_k} corresponds to l_{j_k}, where $1 \leq k \leq 3$. The intention of X_i and $\overline{X}_i$ is that x_i is true and false, respectively. The intention of M_{j_1}, M_{j_2}, and M_{j_3} is that $L_{j_1}L_{j_2}$, $L_{j_2}L_{j_3}$, and $L_{j_3}L_{j_1}$ are true, respectively. The intention of T_j is that t_j is true.

S contains the following $n + m + 1$ properties:

$$R, \ \{P_i \mid 1 \leq i \leq n\} \ \text{ and } \ \{Q_j \mid 1 \leq j \leq m\}$$

The domain of each property is defined as follows:

$$\begin{array}{rlrl} Dom(R) = \{\, Z \,\} & & & \\ Dom(P_1) = \{\, C \,\} & \text{and} & Dom(P_i) = \{\, A_{i-1}, B_{i-1} \,\} & \text{for } 2 \leq i \leq n \\ Dom(Q_1) = \{\, A_n, B_n \,\} & \text{and} & Dom(Q_j) = \{\, C_{j-1} \,\} & \text{for } 2 \leq j \leq m \end{array}$$

2. *The definition of* $C(pd : C')$: Let $C' = T_m$ and assume pd has the form

$$P_1.P_2. \cdots .P_n.Q_1.Q_2. \cdots .Q_m.$$

3. *The definition of* Σ: Let $\Sigma_{\texttt{FUNC}} = \{\texttt{FUNC}(P) \mid P \text{ is a property in } S\}$; that is, Σ contains $\texttt{FUNC}(P)$ for every property P in S. $\Sigma_{\texttt{SC}}$ consists of $4n + 12m$ SCs of the form $C_a(\texttt{Id} : C_b)$ and other $2n+5m+1$ SCs. Only $3m$ SCs depend on the content of E. All other SCs are defined independently of the content of E; that is, these depend only on the number of variables and the number of terms in E. The former is a *varying* part and the latter is a *fixed* part.

3.1. *The fixed part*: Intuitively, the fixed part consists of n truth-setting components, and m satisfaction testing components, and additional $2n+2m+1$ SCs for communicating between the various components.

(a) *Truth-setting components*: For each variable x_i, $1 \leq i \leq n$, there is a truth-setting component that consists of the following four SCs:

$$\{X_i(\texttt{Id} : A_i), X_i(\texttt{Id} : B_i), \overline{X}_i(\texttt{Id} : A_i), \overline{X}_i(\texttt{Id} : B_i)\}$$

Note that every SC has Id as its path description. A truth-setting component is illustrated in Figure 13.

(b) *Satisfaction testing components*: For each term t_j, $1 \leq j \leq m$, there is a satisfaction testing component that consists of the following twelve SCs:

$$\{L_{j_k}(\texttt{Id} : C_j), M_{j_k}(\texttt{Id} : L_{j_k}), M_{j_k}(\texttt{Id} : L_{j_{k+1}}), T_j(\texttt{Id} : M_{j_k}) \mid 1 \leq k \leq 3\}$$

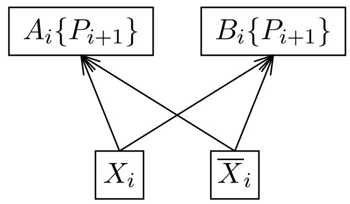

Figure 13: A truth-setting component.

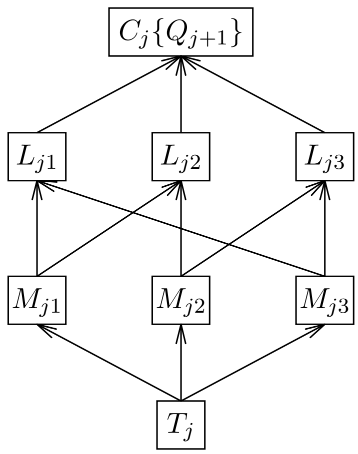

Figure 14: A satisfaction testing component.

where if $k = 3$ then let $k + 1 = 1$ for convenience. Note that every SC has `Id` as its path description. A satisfaction testing component is illustrated in Figure 14.

(c) *The other SCs*: The fixed part contains additional $2n + 2m + 1$ SCs as follows:

$$T_m(R : Z)$$
$$\{C(P_1 : A_1), C(P_1 : B_1)\} \quad \text{and} \quad \{A_{i-1}(P_i : A_i), B_{i-1}(P_i : B_i) \mid 2 \leq i \leq n\}$$
$$\{A_n(Q_1 : C_1), B_n(Q_1 : C_1)\} \quad \text{and} \quad \{C_{j-1}(Q_j : C_j), T_{j-1}(Q_j : T_j) \mid 2 \leq j \leq m\}$$

This completes constructing the fixed part. Its whole construction is illustrated in Figure 15. Class Z, property R, and SC $T_m(R : Z)$ are not used in this proof, but will be used for proving Theorems 6(b) and (c), later.

3.2. *The varying part*: This consists of $3m$ SCs, each of which corresponds to a literal occurring in E. (Hence this part depends on the content of E.) For $1 \leq j \leq m$ and $1 \leq k \leq 3$, if l_{j_k} is a positive (or negative) literal of a variable x_i, then the varying part contains an SC $X_i(pd_{ij} : L_{j_k})$ (or an SC $\overline{X}_i(pd_{ij} : L_{j_k})$), where

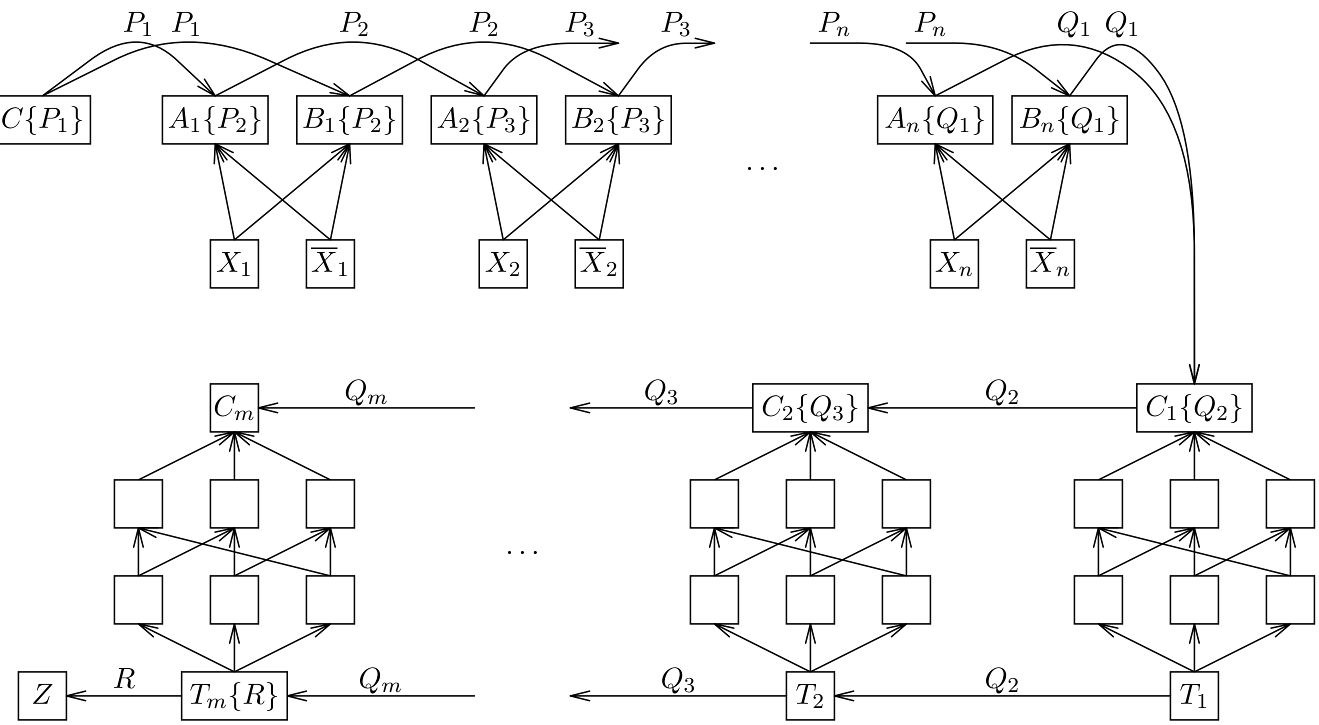

Figure 15: The fixed part of Σ_{SC}.

$$pd_{ij} = P_{i+1}.P_{i+2}.\cdots.P_n.Q_1.Q_2.\cdots.Q_j.^{10}$$

An example of the varying part will be illustrated in Figure 16 of Example 13, later.

This completes constructing $\langle S, \Sigma \rangle$ and $C(pd : C')$ from E. It is not hard to see that $\langle S, \Sigma \rangle$ and $C(pd : C')$ can be constructed from E in polynomial time. Strictly, $\langle S, \Sigma \rangle$ should contain the bottom class $\bot$ and its related SCs. The bottom class, however, is unimportant for this proof, and is not described explicitly. It remains to prove that (1) $C(pd : C')$ is well-formed with respect to $\langle S, \Sigma \rangle$ and (2) $\Sigma \not\models_{\text{MSC}} C(pd : C')$ if and only if E is not a tautology.

Let us first prove that $C(pd : C')$ is well formed with respect to $\langle S, \Sigma \rangle$; that is, $pd \in PathDescs_{\text{MSC}}(C)$. By 3.2, it suffices to show that $pd \in PathDescs(C)$. Let $Chase_{\Sigma_{\mathsf{SC}}}(C, pd)$ be

$$w_0 \xrightarrow{P_1} w_1 \xrightarrow{P_2} \cdots \xrightarrow{P_n} w_n \xrightarrow{Q_1} w_{n+1} \xrightarrow{Q_2} \cdots \xrightarrow{Q_m} w_{n+m}.$$

[10]If $i = n$, then let $pd_{ij} = Q_1.Q_2.\cdots.Q_j$.

Since Σ_{SC} contains $C(P_1 : A_1)$, $A_{i-1}(P_i : A_i)$ for $2 \leq i \leq n$, $A_n(Q_1 : C_1)$, and $C_{j-1}(Q_j : C_j)$ for $2 \leq j \leq m$, it holds that:

$$\Sigma_{\mathsf{SC}} \ \vdash_{\mathrm{A2}} \ C(P_1.P_2.\cdots.P_i : A_i) \quad \text{for } 1 \leq i \leq n \tag{3.5}$$

$$\Sigma_{\mathsf{SC}} \ \vdash_{\mathrm{A2}} \ C(P_1.P_2.\cdots.P_n.Q_1.Q_2.\cdots.Q_j : C_j) \quad \text{for } 1 \leq j \leq m \tag{3.6}$$

Thus it follows from Lemma 5 that $A_i \in \mathit{Cl}(w_i)$ and $C_j \in \mathit{Cl}(w_{n+j})$. Since (1) $C \in \mathit{Dom}(P_1)$, (2) $A_{i-1} \in \mathit{Dom}(P_i)$ for $2 \leq i \leq n$, (3) $A_n \in \mathit{Dom}(Q_1)$, and (4) $C_{j-1} \in \mathit{Dom}(Q_j)$ for $2 \leq j \leq m$, it holds that $\mathit{Cl}(w_{i-1}) \cap \mathit{Dom}(P_i) \neq \emptyset$ for $1 \leq i \leq n$ and $\mathit{Cl}(w_{n+j-1}) \cap \mathit{Dom}(Q_j) \neq \emptyset$ for $1 \leq j \leq m$. That is, $\mathit{Chase}_{\Sigma_{\mathsf{SC}}}(C, \mathit{pd})$ satisfies a property value integrity condition. Hence $\mathit{pd} \in \mathit{PathDescs}(C)$ by Lemma 6. This completes proving that $C(\mathit{pd} : C')$ is well-formed with respect to $\langle S, \Sigma \rangle$. In the following we will prove that $\Sigma \not\models_{\mathrm{MSC}} C(\mathit{pd} : C')$ if and only if E is not a tautology.

Only if part. Assume that $\Sigma \not\models_{\mathrm{MSC}} C(\mathit{pd} : C')$. By Theorem 5, there is a pd-List

$$v_0 \xrightarrow{P_1} v_1 \xrightarrow{P_2} \cdots \xrightarrow{P_n} v_n \xrightarrow{Q_1} u_1 \xrightarrow{Q_2} \cdots \xrightarrow{Q_m} u_m$$

satisfying MSC and PDLs 1 to 3. We first prove the following three claims.

Claim 1: $\mathit{Cl}(v_i)$ contains either X_i or $\overline{X}_i$ for $1 \leq i \leq n$.

Claim 2: $\{L_{j_1}, L_{j_2}, L_{j_3}\} \not\subseteq \mathit{Cl}(u_j)$ for $1 \leq j \leq m$.

Claim 3: If either (1) $X_i \in \mathit{Cl}(v_i)$ and l_{j_k} is a positive literal of x_i or (2) $\overline{X}_i \in \mathit{Cl}(v_i)$ and l_{j_k} is a negative literal of x_i, then $L_{j_k} \in \mathit{Cl}(u_j)$, where $1 \leq i \leq n$, $1 \leq j \leq m$, and $1 \leq k \leq 3$.

Proof of Claim 1: Since $\Sigma_{\mathsf{SC}} \models C(P_1.P_2.\cdots.P_i : A_i)$ by 3.5 and Lemma 3, it follows from 3.1 that $\Sigma_{\mathsf{SC}} \models_{\mathrm{MSC}} C(P_1.P_2.\cdots.P_i : A_i)$, where $1 \leq i \leq n$. Furthermore, since (1) $C \in \mathit{Cl}(v_0)$ by PDL 3 and (2) the pd-List satisfies Σ_{SC} by PDL 2, $\mathit{Cl}(v_i)$ should contain A_i. Similarly, $\mathit{Cl}(v_i)$ should also contain B_i by the symmetry between A_i and B_i. That is, $\{A_i, B_i\} \subseteq \mathit{Cl}(v_i)$ for $1 \leq i \leq n$. Since the pd-List satisfies MSC, $\mathit{Msc}(v_i)$ exists for v_i. From Figure 13, we can see that $\mathit{Cl}(v_i)$ should contain either X_i or $\overline{X}_i$.[11] Hence Claim 1 follows.

Proof of Claim 2: Assume that $\{L_{j_1}, L_{j_2}, L_{j_3}\} \subseteq \mathit{Cl}(u_j)$ for some j. From Figure 14, we can see that $\mathit{Cl}(u_j)$ should contain T_j in order that $\mathit{Msc}(u_j)$ exists for u_j. Since Σ_{SC} contains $T_{l-1}(Q_l : T_l)$ for $2 \leq l \leq m$, it holds that $\Sigma_{\mathsf{SC}} \models_{\mathrm{MSC}} T_j(Q_{j+1}.Q_{j+2}.\cdots.Q_m : T_m)$. Furthermore, since the pd-List satisfies Σ_{SC} by PDL 2, $T_j \in \mathit{Cl}(u_j)$ implies $T_m \in \mathit{Cl}(u_m)$. On the other hand, since

[11] $\mathit{Cl}(v_i)$ may contain both of them. Then $\mathit{Msc}(v_i)$ is the bottom class $\perp$.

the *pd*-List satisfies PDL 3, it holds that $T_m = C' \notin Cl(u_m)$. Contradiction. Hence $\{L_{j_1}, L_{j_2}, L_{j_3}\} \not\subseteq Cl(u_j)$ for any j. That is, Claim 2 follows.

Proof of Claim 3: Assume that l_{j_k} is a positive literal of x_i. Since (1) Σ_{SC} contains $X_i(pd_{ij} : L_{j_k})$ by definition and (2) the *pd*-List satisfies Σ_{SC} by PDL 2, $X_i \in Cl(v_i)$ implies $L_{j_k} \in Cl(u_j)$. Similarly, if l_{j_k} is a negative literal of x_i, then $\overline{X}_i \in Cl(v_i)$ implies $L_{j_k} \in Cl(u_j)$. Hence Claim 3 follows.

Let us define a truth assignment $\tau : \{x_1, x_2, \cdots, x_n\} \to \{T(rue), F(alse)\}$ such that if $X_i \in Cl(v_i)$, then $\tau(x_i) = T$; otherwise $\tau(x_i) = F$. We prove that τ makes E false; that is, E is not a tautology. It suffices to prove that τ makes t_j false for $1 \le j \le m$.

By Claim 2, there is a class L_{j_k} that is not in $Cl(w_j)$, where $1 \le k \le 3$. There are two cases to be considered.

Assume that l_{j_k} is a positive literal of a variable x_i. Then $L_{j_k} \notin Cl(w_j)$ implies $X_i \notin Cl(v_i)$ by Claim 3. Thus $\tau(x_i) = F$ by definition. Hence τ makes t_j false.

Assume that l_{j_k} is a negative literal of x_i. Then $L_{j_k} \notin Cl(w_j)$ implies $\overline{X}_i \notin Cl(v_i)$ by Claim 3. Furthermore, $\overline{X}_i \notin Cl(v_i)$ implies $X_i \in Cl(v_i)$ by Claim 1. Thus $\tau(x_i) = T$ by definition. Hence τ also makes t_j false in this case. Consequently, if $\Sigma \not\models_{\mathrm{MSC}} C(pd : C')$, then E is not a tautology.

If part. Assume that E is not a tautology. There is a truth assignment $\tau : \{x_1, x_2, \cdots, x_n\} \to \{T, F\}$ that makes E false. Let us define a *pd*-List

$$v_0 \xrightarrow{P_1} v_1 \xrightarrow{P_2} \cdots \xrightarrow{P_n} v_n \xrightarrow{Q_1} u_1 \xrightarrow{Q_2} \cdots \xrightarrow{Q_m} u_m,$$

which satisfies MSC (by showing $Msc(v_i)$ and $Msc(u_j)$ for all i and j), as follows:

1. $Cl(v_0) = \{\, C \,\}$ and $Msc(v_0) = C$.

2.1. If $\tau(x_i) = T$, then $Cl(v_i) = \{A_i, B_i, X_i\}$ and $Msc(v_i) = X_i$, where $1 \le i \le n$.

2.2. If $\tau(x_i) = F$, then $Cl(v_i) = \{A_i, B_i, \overline{X}_i\}$ and $Msc(v_i) = \overline{X}_i$.

3.1. If τ makes all the three literals in t_j false, then $Cl(u_j) = \{C_j\}$ and $Msc(u_j) = C_j$, where $1 \le j \le m$.

3.2. If τ makes just one literal l_{j_k} true, then $Cl(u_j) = \{C_j, L_{j_k}\}$ and $Msc(u_j) = L_{j_k}$, where $1 \le k \le 3$.

3.3. If τ makes two literals l_{j_k} and $l_{j_{k+1}}$ true, then $Cl(u_j)$ must be $\{C_j, L_{j_k}, L_{j_{k+1}}, M_{j_k}\}$ and $Msc(u_j) = M_{j_k}$.[12]

[12] Since τ makes E false, τ does not make all the three literals true.

By Theorem 5, in order to prove that $\Sigma \not\models_{\text{MSC}} C(pd : C')$, it suffices to show that the pd-List satisfies PDLs 1 to 3 (as well as MSC).

By considering the domain of each property, we can see that $C \in Cl(v_0) \cap Dom(P_1)$, $A_{i-1} \in Cl(v_{i-1}) \cap Dom(P_i)$ for $2 \leq i \leq n$, $A_n \in Cl(v_n) \cap Dom(Q_1)$, and $C_{j-1} \in Cl(u_{j-1}) \cap Dom(Q_j)$ for $2 \leq j \leq m$. Hence the pd-List satisfies PDL 1.

It is easy to see that the pd-List satisfies all the SCs in the fixed part. By 2.1 and 2.2 above, $X_i \in Cl(v_i)$ if and only if $\tau(x_i) = T$, and $\overline{X}_i \in Cl(v_i)$ if and only if $\tau(x_i) = F$. Furthermore, by 3.1 to 3.3, $L_{j_k} \in Cl(u_j)$ if and only if τ makes l_{j_k} true. Thus the pd-List also satisfies all the SCs in the varying part. That is, the pd-List satisfies PDL 2.

Since $C \in Cl(v_0)$ and $C' = T_m \notin Cl(u_m)$, the pd-List satisfies PDL 3.

This completes proving that $\Sigma \not\models_{\text{MSC}} C(pd : C')$ if and only if E is not a tautology. As a result, Lemma 11 holds. □

Example 13: Let $E = x_1\,\overline{x}_2\,x_3 \vee \overline{x}_1\,\overline{x}_2\,x_3$ be a 3DNF Boolean expression. Then the SC $C(pd : C')$ has the form $C(P_1.P_2.P_3.Q_1.Q_2 : T_2)$. Furthermore, the varying part of Σ_{SC} consists of the following six SCs, as illustrated in Figure 16:

$X_1(P_2.P_3.Q_1 : L_{11})$, $\overline{X}_2(P_3.Q_1 : L_{12})$, $X_3(Q_1 : L_{13})$,
$\overline{X}_1(P_2.P_3.Q_1.Q_2 : L_{21})$, $\overline{X}_2(P_3.Q_1.Q_2 : L_{22})$, $X_3(Q_1.Q_2 : L_{23})$

It is easy to construct the whole database schema $\langle S, \Sigma \rangle$ from E. □

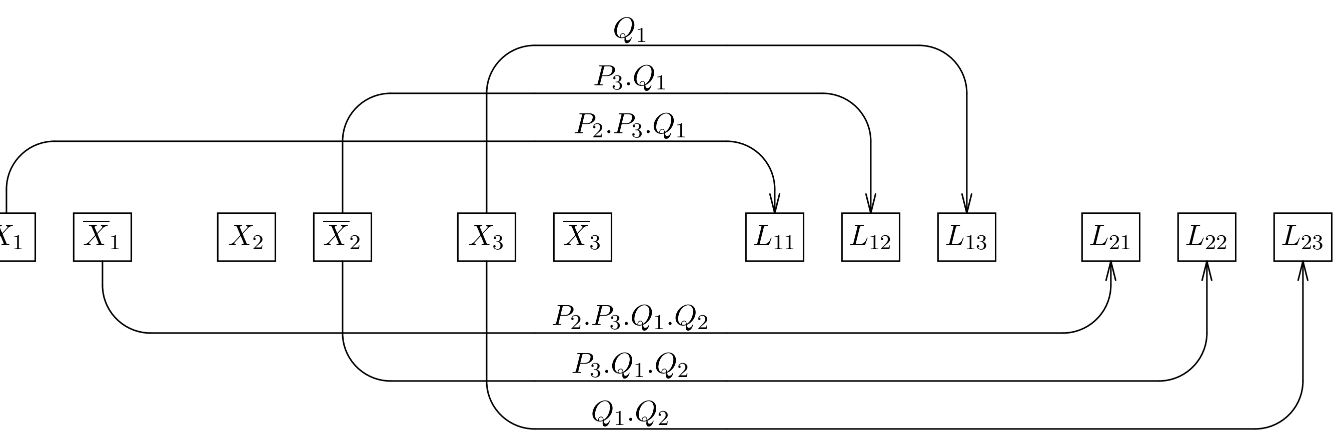

Figure 16: The varying part of Σ_{SC}.

This completes proving Theorem 6(a). Let us next prove Theorem 6(b); that is, it is NP-complete to decide whether or not $pd \notin PathDescs_{\text{MSC}}(C)$. The following lemma will be used for proving that the problem is in NP.

Lemma 12: Let $pd = P_1.P_2.\cdots.P_n$. Then $pd \notin PathDescs_{\text{MSC}}(C)$ if and only if there is a pd_i-List $v_0 \xrightarrow{P_1} v_1 \xrightarrow{P_2} \cdots \xrightarrow{P_i} v_i$ satisfying the following six conditions: (1) $i < n$, (2) $C \in Cl(v_0)$, (3) $Cl(v_i) \cap Dom(P_{i+1}) = \emptyset$, (4) MSC, (5) PDL 1, and (6) PDL 2.

Proof. *If part.* Assume that there is a pd_i-List $v_0 \xrightarrow{P_1} v_1 \xrightarrow{P_2} \cdots \xrightarrow{P_i} v_i$ satisfying the six conditions above. Let $G(V, A)$ be the augmented graph of the pd_i-List with respect to S. As is proving Theorem 5, $G(V, A)$ is an interpretation satisfying MSC as well as $\Sigma \cup S_{\texttt{FUNC}}$. By Step 3 of Procedure 1, if there is an arc $v_i \xrightarrow{P} u \in A$, then $Cl(v_i) \cap Dom(P) \neq \emptyset$. Thus by condition (3), there is no arc $v_i \xrightarrow{P_{i+1}} u \in A$. Hence there is no path in $G(V, A)$ from v_0 described by $pd_i \circ P_{i+1}$. Since $pd_i \circ P_{i+1}$ is a prefix of pd, there is no path in $G(V, A)$ from v_0 described by pd, either. As a result, $pd \notin PathDescs_{\text{MSC}}(C)$.

Only if part. Assume that $pd \notin PathDescs_{\text{MSC}}(C)$. Let i be the largest index such that $pd_i \in PathDescs_{\text{MSC}}(C)$. Then $pd_{i+1} \notin PathDescs_{\text{MSC}}(C)$. By definition, there is an interpretation $G(V, A)$ satisfying MSC and $\Sigma \cup S_{\texttt{FUNC}}$ such that for a vertex $u_0 \in V$ with $C \in Cl(u_0)$, there is no path in $G(V, A)$ from u_0 described by pd_{i+1}. Since $pd_i \in PathDescs_{\text{MSC}}(C)$, however, there must be a path in $G(V, A)$ from u_0 described by pd_i. Let us denote the path by $u_0 \xrightarrow{P_1} u_1 \xrightarrow{P_2} \cdots \xrightarrow{P_i} u_i$. Let $v_0 \xrightarrow{P_1} v_1 \xrightarrow{P_2} \cdots \xrightarrow{P_i} v_i$ be a pd_i-List such that $Cl(v_j) = Cl(u_j)$ for $0 \leq j \leq n$. We will prove that the pd_i-List satisfies all the six conditions.

Clearly, the pd_i-List satisfies conditions (1), (2), and (4). As is proving Lemma 1, the pd_i-List satisfies PDLs 1 and 2, that is, conditions (5) and (6). If the pd_i-List does not satisfy condition (3), then it holds that $Cl(u_i) \cap Dom(P_{i+1}) \neq \emptyset$, since $Cl(v_i) = Cl(u_i)$. Since $\texttt{FUNC}(P_{i+1}) \in S_{\texttt{FUNC}}$, this implies that there must be an arc $u_i \xrightarrow{P_{i+1}} w \in A$. Hence $G(V, A)$ contains a path from u_0 to w described by pd_{i+1}, since $G(V, A)$ contains a path from u_0 to u_i described by pd_i and $pd_{i+1} = pd_i \circ P_{i+1}$. Contradiction. Thus the pd_i-List must satisfy condition (3). This completes proving the only if part. □

Example 14: For the database schema $\langle S, \Sigma\rangle$ in Example 12, let us show that $\mathtt{A.B.D} \notin \mathit{PathDescs}_{\mathrm{MSC}}(\mathtt{a}_1)$ by Lemma 12. Consider the '$\mathtt{A.B}$'-List in Figure 17, where each vertex is labeled its MSC. It is easy to verify that the '$\mathtt{A.B}$'-List satisfies all the conditions of Lemma 12. For example, it satisfies condition (3), since $\mathit{Cl}(v_2) \cap \mathit{Dom}(\mathtt{D}) = \{\mathtt{c}_1, \mathtt{c}_2, \mathtt{c}_4\} \cap \{\mathtt{c}_3\} = \emptyset$. On the other hand, it holds that $\mathtt{A.B.C} \in \mathit{PathDescs}_{\mathrm{MSC}}(\mathtt{a}_1)$, since there is no list satisfying the conditions of Lemma 12. □

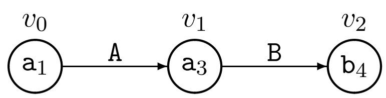

Figure 17: An '$\mathtt{A.B}$'-List showing $\mathtt{A.B.D} \notin \mathit{PathDescs}_{\mathrm{MSC}}(\mathtt{a}_1)$.

Proof of Theorem 6(b):
We first prove that it is in NP to decide whether or not $pd \notin \mathit{PathDescs}_{\mathrm{MSC}}(C)$. By Lemma 12, $pd \notin \mathit{PathDescs}_{\mathrm{MSC}}(C)$ if and only if there is a pd_i-List satisfying the six conditions of Lemma 12. Since the size of the pd_i-List is at most $D \cdot \mathit{len}(pd)$, where D is the size of $\langle S, \Sigma\rangle$, the pd_i-List can be guessed in NP time. After that, it can be tested in polynomial time whether or not the pd_i-List satisfies all the six conditions. Hence the problem is in NP.

We now prove that it is NP-hard to decide if $pd \notin \mathit{PathDescs}_{\mathrm{MSC}}(C)$. Consider the database schema $\langle S, \Sigma\rangle$ in Lemma 11. We will prove that

$$\Sigma \models_{\mathrm{MSC}} C(pd : C') \quad \text{if and only if} \quad pd \circ R \in \mathit{PathDescs}_{\mathrm{MSC}}(C).$$

Since $\Sigma \not\models_{\mathrm{MSC}} C(pd : C')$ if and only if E is not a tautology, this implies that $pd \circ R \notin \mathit{PathDescs}_{\mathrm{MSC}}(C)$ if and only if E is not a tautology. Hence it is NP-hard to decide whether or not $pd \circ R \notin \mathit{PathDescs}_{\mathrm{MSC}}(C)$.

If part. Assume that $pd \circ R \in \mathit{PathDescs}_{\mathrm{MSC}}(C)$. Let $G(V, A)$ be an interpretation satisfying MSC and Σ. Assume that there is a path in $G(V, A)$ from a vertex u to a vertex v described by pd, where $C \in \mathit{Cl}(u)$. In order to prove that $\Sigma \models_{\mathrm{MSC}} C(pd : C')$, it suffices to show that $C' = T_m \in \mathit{Cl}(v)$.

Since $\mathtt{FUNC}(P) \in \Sigma$ for every property P in S by definition, it holds that $\Sigma = \Sigma \cup S_{\mathtt{FUNC}}$. Thus $G(V, A)$ satisfies $\Sigma \cup S_{\mathtt{FUNC}}$ as well as MSC. Furthermore, since $pd \circ R \in \mathit{PathDescs}_{\mathrm{MSC}}(C)$, there must be a path from u to a vertex w described by $pd \circ R$. Because of property functionality, the path from u to w is unique. Hence the path from u to v should be on the path from u to w; that is, there is an arc $v \xrightarrow{R} w \in A$. Since $G(V, A)$ is an interpretation, it satisfies a property value integrity condition. Thus it holds that $Cl(v) \cap Dom(R) \neq \emptyset$. Since $Dom(R) = \{T_m\}$ by definition, it holds that $T_m \in Cl(v)$. Hence $\Sigma \models_{\mathrm{MSC}} C(pd : C')$.

Only if part. Assume that $\Sigma \models_{\mathrm{MSC}} C(pd : C')$. Let $G(V, A)$ be an interpretation satisfying MSC and $\Sigma \cup S_{\mathtt{FUNC}}$. Let $u \in V$, where $C \in Cl(u)$. In order to prove that $pd \circ R \in PathDescs_{\mathrm{MSC}}(C)$, it suffices to show that there is a path in $G(V, A)$ from u described by $pd \circ R$. Since $C(pd : C')$ is well-formed with respect to $\langle S, \Sigma \rangle$, that is, $pd \in PathDescs_{\mathrm{MSC}}(C)$, there is a path in $G(V, A)$ from u to a vertex v described by pd. Since $\Sigma \models_{\mathrm{MSC}} C(pd : C')$, $C \in Cl(u)$ implies $C' = T_m \in Cl(v)$. Since $T_m \in Dom(R)$ by definition, it holds that $Cl(v) \cap Dom(R) \neq \emptyset$. Furthermore, since $G(V, A)$ satisfies $\mathtt{FUNC}(R) \in S_{\mathtt{FUNC}}$, there must be an arc $v \xrightarrow{R} w \in A$. Thus $G(V, A)$ contains a path from u to w described by $pd \circ R$. Hence $pd \circ R \in PathDescs_{\mathrm{MSC}}(C)$. This completes proving Theorem 6(b). □

We finally prove Theorem 6(c); that is, it is NP-complete to decide whether or not $pd \notin PathFuncs_{\mathrm{MSC}}(C)$. It is easy to see that the proof of Lemma 9 applies also to the case of MSC, and thus we have the following corollary of Lemma 9.

Corollary 1: $pd \in PathFuncs_{\mathrm{MSC}}(C)$ if and only if $pd \in PathDescs_{\mathrm{MSC}}(C)$ and $\mathtt{FUNC}(P_i) \in \Sigma_{\mathtt{FUNC}}$ for $1 \leq i \leq n$, where $pd = P_1.P_2. \cdots .P_n$. □

Proof of Theorem 6(c):

Since (1) it is in NP to decide whether or not $pd \notin PathDescs_{\mathrm{MSC}}(C)$ by Theorem 6(b) and (2) it can be decided in polynomial time whether or not $\mathtt{FUNC}(P_i) \in \Sigma_{\mathtt{FUNC}}$ for $1 \leq i \leq n$, it follows from Corollary 1 that it is in NP to decide whether or not $pd \notin PathFuncs_{\mathrm{MSC}}(C)$.

Finally, we prove that it is NP-hard to decide if $pd \notin PathFuncs_{\mathrm{MSC}}(C)$. Consider the database schema $\langle S, \Sigma \rangle$ in Lemma 11. Since $\mathtt{FUNC}(P) \in \Sigma$ for every property P in S by definition, it follows from Corollary 1 that $pd \circ R \in PathFuncs_{\mathrm{MSC}}(C)$ if and only if $pd \circ R \in PathDescs_{\mathrm{MSC}}(C)$. Since it is NP-hard to decide whether or not $pd \circ R \notin PathDescs_{\mathrm{MSC}}(C)$, it is also NP-hard to decide whether or not $pd \circ R \notin PathFuncs_{\mathrm{MSC}}(C)$. Hence Theorem 6(c) holds. □

3.2 The case of bounded path lengths

Let $l = \max\{len(pd') \mid C_a(pd' : C_b) \in \Sigma\}$. If $l = 0$, that is, if every SC in $\Sigma_{\texttt{SC}}$ has the form $C_a(\texttt{Id} : C_b)$, then the problems in this section will be trivial. Thus consider the case that $l \geq 1$. Let $Classes(S) = \{C_1, C_2, \cdots, C_K\}$.

In the following we will prove the following theorem. By the theorem, if l is *bounded*, then the three decision problems of Theorem 6 can be solved in polynomial time.

Theorem 7: The following decision problems are solved in $O(K^{l+1} \cdot D \cdot l \cdot (len(pd) + 1))$ time, where D is the size of $\langle S, \Sigma \rangle$.

a. $\Sigma \models_{\text{MSC}} C(pd : C')$?

b. $pd \in PathDescs_{\text{MSC}}(C)$?

c. $pd \in PathFuncs_{\text{MSC}}(C)$? □

Proof of Theorem 7(a):

As before, let us denote pd by $P_1.P_2.\cdots.P_n$. For $0 \leq i \leq n$, let PDL_i denote the set of pd_i-Lists $v_0 \xrightarrow{P_1} v_1 \xrightarrow{P_2} \cdots \xrightarrow{P_i} v_i$ satisfying the four conditions: (1) MSC, (2) PDL 1, (3) PDL 2, and (4) $C \in Cl(v_0)$. Then by Theorem 5, $\Sigma \not\models_{\text{MSC}} C(pd : C')$ if and only if there is a pd-Lists $v_0 \xrightarrow{P_1} v_1 \xrightarrow{P_2} \cdots \xrightarrow{P_n} v_n$ in PDL_n such that $C' \notin Cl(v_n)$.[13] Thus by computing PDL_n, we can decide whether or not $\Sigma \models_{\text{MSC}} C(pd : C')$. It is important to note that a pd_i-List satisfying MSC can be represented by specifying $Msc(v_j)$ (instead of $Cl(v_j)$) for each vertex v_j, where $0 \leq j \leq i$, since

$$Cl(v_j) = \{C'' \in Classes(S) \mid \Sigma \models Msc(v_j)(\texttt{Id} : C'')\}.^{14}$$

Strictly, there may be a vertex v_j such that $Cl(v_j) = \emptyset$. (Then $Msc(v_j)$ is undefined.) In order to treat such a case uniformly, it is convenient to introduce a special class C_0, and to consider $Cl(v_j) = \emptyset$ if $Msc(v_j) = C_0$. By the observations above, the number of pd_i-Lists in PDL_i is at most $(K+1)^{i+1}$, since a pd_i-List consists of $i + 1$ vertices.

The following procedure computes PDL_i by (1) generating all pd_i-Lists satisfying MSC and (2) checking PDL 1, PDL 2, and $C \in Cl(v_0)$ for each generated pd_i-List.

---

[13] Note that $pd_n = pd$ by definition.

[14] If the pd_i-List does not satisfy MSC, then $Cl(v_j)$ may be an arbitrary subset of $Classes(S)$, so that the number of possible $Cl(v_j)$ becomes 2^K.

Procedure 6: (Computing PDL_i.)
input: a database schema $\langle S, \Sigma\rangle$, a class $C \in \mathit{Classes}(S)$, and a path description pd_i ($= P_1.P_2. \cdots .P_i$).

1. Let $PDL_i \leftarrow \emptyset$.

2. **for** $k_0 \leftarrow 0$ **to** K; **for** $k_1 \leftarrow 0$ **to** K; $\cdots$; **for** $k_i \leftarrow 0$ **to** K

   do begin

   3. Construct a pd_i-List $v_0 \xrightarrow{P_1} v_1 \xrightarrow{P_2} \cdots \xrightarrow{P_i} v_i$ such that $\mathit{Msc}(v_j) = C_{k_j}$, that is, $\mathit{Cl}(v_j) = \{C_j \in \mathit{Classes}(S) \mid \Sigma \models C_{k_j}(\mathtt{Id} : C_j)\}$ for $0 \le j \le i$.
   4. **if** the pd_i-List $v_0 \xrightarrow{P_1} v_1 \xrightarrow{P_2} \cdots \xrightarrow{P_i} v_i$ satisfies PDL 1, PDL 2, and $C \in \mathit{Cl}(v_0)$

      then add the pd_i-List to PDL_i.

   end □

Example 15: Consider the database schema $\langle S, \Sigma\rangle$ in Example 12. For a class $\mathtt{a}_3$ and a path description $\mathtt{A.B}$, let us execute Procedure 6. Then PDL_2 consists of the following twelve '$\mathtt{A.B}$'-Lists, where each '$\mathtt{A.B}$'-List $v_0 \xrightarrow{\mathtt{A}} v_1 \xrightarrow{\mathtt{B}} v_2$ is denoted by $(\mathit{Msc}(v_0), \mathit{Msc}(v_1), \mathit{Msc}(v_2))$ for simplicity:

$(\mathtt{a}_3, \mathtt{a}_3, \mathtt{b}_3)$, $(\mathtt{a}_3, \mathtt{a}_3, \mathtt{b}_4)$, $(\mathtt{a}_3, \mathtt{a}_3, \bot)$, $(\mathtt{a}_3, \bot, \mathtt{b}_3)$, $(\mathtt{a}_3, \bot, \mathtt{b}_4)$, $(\mathtt{a}_3, \bot, \bot)$,
$(\bot, \mathtt{a}_3, \mathtt{b}_3)$, $(\bot, \mathtt{a}_3, \mathtt{b}_4)$, $(\bot, \mathtt{a}_3, \bot)$, $(\bot, \bot, \mathtt{b}_3)$, $(\bot, \bot, \mathtt{b}_4)$, $(\bot, \bot, \bot)$

Since $\mathtt{b}_2 \in \mathit{Cl}(v_2)$ for every '$\mathtt{A.B}$'-List in PDL_2, it holds that $\Sigma \models_{\mathrm{MSC}} \mathtt{a}_3(\mathtt{A.B} : \mathtt{b}_2)$. On the other hand, since $\mathtt{b}_4 \notin \mathit{Cl}(v_2)$ for $(\mathtt{a}_3, \mathtt{a}_3, \mathtt{b}_3) \in PDL_2$, it holds that $\Sigma \not\models_{\mathrm{MSC}} \mathtt{a}_3(\mathtt{A.B} : \mathtt{b}_4)$. □

Let us estimate the time complexity of Procedure 6. By the for loop of Step 2, Steps 3 and 4 are executed exactly $(K+1)^{i+1}$ times, that is, $O(K^{i+1})$ times. Step 3 can be executed in $O(D \cdot (i+1))$ time, since each $\mathit{Cl}(v_j)$ can be computed in $O(D)$ time as in estimating Step 2 of Procedure 3. Step 4 can be executed in $O(D \cdot (i+1))$ time, since the size of the pd_i-List is at most $D \cdot (i+1)$. Hence Procedure 6 can be executed in $O(K^{i+1} \cdot D \cdot (i+1))$ time.

We now consider how to decide whether or not $\Sigma \models_{\mathrm{MSC}} C(pd : C')$. There are two cases to be considered: $\mathit{len}(pd) \le l$ and $\mathit{len}(pd) > l$.

Case 1. Assume that $len(pd) \leq l$. PDL_n can be computed in $O(K^{len(pd)+1} \cdot D \cdot (len(pd)+1))$ time by Procedure 6. After that, it can be decided in $O(K^{len(pd)+1} \cdot D \cdot (len(pd)+1))$ time whether or not there is a pd-Lists $v_0 \xrightarrow{P_1} v_1 \xrightarrow{P_2} \cdots \xrightarrow{P_n} v_n$ in PDL_n such that $C' \notin Cl(v_n)$, since the size of PDL_n is $O(K^{len(pd)+1} \cdot D \cdot (len(pd)+1))$. Thus it can be decided in $O(K^{len(pd)+1} \cdot D \cdot (len(pd)+1))$ time whether or not $\Sigma \models_{\mathrm{MSC}} C(pd : C')$. Hence Theorem 7(a) holds in this case.

Case 2. Assume that $len(pd) > l$. PDL_n cannot be used, since its size may exceed $O(K^{l+1} \cdot D \cdot l \cdot (len(pd)+1))$. Note that in order to decide whether or not $\Sigma \models_{\mathrm{MSC}} C(pd : C')$, we do not need PDL_n but only the set $\{v_n \mid v_0 \xrightarrow{P_1} v_1 \xrightarrow{P_2} \cdots \xrightarrow{P_n} v_n \in PDL_n\}$. The following lemma will be useful for computing the set.

Lemma 13: A pd_{i+l}-List $v_0 \xrightarrow{P_1} v_1 \xrightarrow{P_2} \cdots \xrightarrow{P_{i+l}} v_{i+l}$ is in PDL_{i+l} if and only if (1) the pd_{i+l-1}-List $v_0 \xrightarrow{P_1} v_1 \xrightarrow{P_2} \cdots \xrightarrow{P_{i+l-1}} v_{i+l-1}$ is in PDL_{i+l-1} and (2) the $P_{i+1}.P_{i+2}.\cdots.P_{i+l}$-List $v_i \xrightarrow{P_{i+1}} v_{i+1} \xrightarrow{P_{i+2}} \cdots \xrightarrow{P_{i+l}} v_{i+l}$ satisfies MSC, PDL 1, and PDL 2, where $1 \leq i \leq n-l$.

Proof. The only if part is clear. Assume that (1) the pd_{i+l-1}-List $v_0 \xrightarrow{P_1} v_1 \xrightarrow{P_2} \cdots \xrightarrow{P_{i+l-1}} v_{i+l-1}$ is in PDL_{i+l-1} and (2) the $P_{i+1}.P_{i+2}.\cdots.P_{i+l}$-List $v_i \xrightarrow{P_{i+1}} v_{i+1} \xrightarrow{P_{i+2}} \cdots \xrightarrow{P_{i+l}} v_{i+l}$ satisfies MSC, PDL 1, and PDL 2. By the definition of PDL_{i+l}, it suffices to show that the pd_{i+l}-List $v_0 \xrightarrow{P_1} v_1 \xrightarrow{P_2} \cdots \xrightarrow{P_{i+l}} v_{i+l}$ satisfies MSC, PDL 1, PDL 2, and $C \in Cl(v_0)$.

Clearly, the pd_{i+l}-List satisfies MSC, PDL 1, and $C \in Cl(v_0)$. As for PDL 2, assume that there is an SC $C_a(pd' : C_b) \in \Sigma_{\mathsf{SC}}$ such that $C_a \in Cl(v_j)$ and $pd' = P_{j+1}.P_{j+2}.\cdots.P_{j+m}$ for some j and m. It suffices to show that $C_b \in Cl(v_{j+m})$. Note that $m \leq l$ by the definition of l. Hence the $P_{j+1}.P_{j+2}.\cdots.P_{j+m}$-List $v_j \xrightarrow{P_{j+1}} v_{j+1} \xrightarrow{P_{j+2}} \cdots \xrightarrow{P_{j+m}} v_{j+m}$ must be included in either the pd_{i+l-1}-List or the $P_{i+1}.P_{i+2}.\cdots.P_{i+l}$-List. Since both the pd_{i+l-1}-List and the $P_{i+1}.P_{i+2}.\cdots.P_{i+l}$-List satisfy $C_a(pd' : C_b)$ by PDL 2, $C_a \in Cl(v_j)$ implies $C_b \in Cl(v_{j+l})$. This completes proving Lemma 13. □

For $1 \leq i \leq n-l+1$, let PDL^i_{i+l-1} denote the set of $P_{i+1}.P_{i+2}.\cdots.P_{i+l-1}$-Lists such that $v_i \xrightarrow{P_{i+1}} v_{i+1} \xrightarrow{P_{i+2}} \cdots \xrightarrow{P_{i+l-1}} v_{i+l-1}$ is in PDL^i_{i+l-1} if and only if there is a pd_{i+l-1}-List $v_0 \xrightarrow{P_1} v_1 \xrightarrow{P_2} \cdots \xrightarrow{P_{i+l-1}} v_{i+l-1}$ in PDL_{i+l-1} satisfying the four conditions: MSC, PDL 1, PDL 2, and $C \in Cl(v_0)$. Note that if $l = 1$, then PDL^i_{i+l-1} is the set of vertices v_i such that v_i is in PDL^i_{i+l-1} if and only if there is a pd_{i+l-1}-List $v_0 \xrightarrow{P_1} v_1 \xrightarrow{P_2} \cdots \xrightarrow{P_{i+l-1}} v_{i+l-1}$ in PDL_{i+l-1} satisfying those four conditions. In order to decide whether or not $\Sigma \models_{\text{MSC}} C(pd : C')$, it suffices to compute PDL^{n-l+1}_n. Lemma 13 suggests that PDL^{i+1}_{i+l} is computed from PDL^i_{i+l-1}, as follows:

Procedure 7: (Computing PDL^{i+1}_{i+l} from PDL^i_{i+l-1}.)
input: a database schema $\langle S, \Sigma \rangle$, a path description pd $(= P_1.P_2.\cdots.P_n)$, and PDL^i_{i+l-1}.

1. Let $PDL^{i+1}_{i+l} \leftarrow \emptyset$.
2. **for** each $v_i \xrightarrow{P_{i+1}} v_{i+1} \xrightarrow{P_{i+2}} \cdots \xrightarrow{P_{i+l-1}} v_{i+l-1}$ in PDL^i_{i+l-1}; **for** $k \leftarrow 0$ **to** K

   do begin

   3. Construct a $P_{i+1}.P_{i+2}.\cdots.P_{i+l}$-List

      $$v_i \xrightarrow{P_{i+1}} v_{i+1} \xrightarrow{P_{i+2}} \cdots \xrightarrow{P_{i+l-1}} v_{i+l-1} \xrightarrow{P_{i+l}} v_{i+l}$$

      by adding a vertex v_{i+l} such that $Msc(v_{i+l}) = C_k$, that is, $Cl(v_{i+l}) = \{C_{i+l} \in Classes(S) \mid \Sigma \models C_k(\texttt{Id} : C_{i+l})\}$.
   4. **if** the $P_{i+1}.P_{i+2}.\cdots.P_{i+l}$-List satisfies PDLs 1 and 2

      then add the $P_{i+2}.P_{i+3}.\cdots.P_{i+l}$-List $v_{i+1} \xrightarrow{P_{i+2}} v_{i+2} \xrightarrow{P_{i+3}} \cdots \xrightarrow{P_{i+l}} v_{i+l}$ to PDL^{i+1}_{i+l}.

   end □

Example 16: In Example 15, let us compute PDL^2_2 from PDL^1_1. Note that $l = 1$ in the example. By executing Procedure 6 for class $\texttt{a}_3$ and path description $\texttt{A}$, we obtain PDL_1 as follows: $\{(\texttt{a}_3, \texttt{a}_3), (\texttt{a}_3, \bot), (\bot, \texttt{a}_3), (\bot, \bot)\}$. Thus $PDL^1_1 = \{(\texttt{a}_3), (\bot)\}$, which is obtained from PDL_1 by removing v_0 for each '$\texttt{A}$'-List $v_0 \xrightarrow{\texttt{A}} v_1 \in PDL_1$.

For $(\texttt{a}_3) \in PDL^1_1$, the set of '$\texttt{B}$'-Lists satisfying the if condition of Step 4 is $\{(\texttt{a}_3, \texttt{b}_3), (\texttt{a}_3, \texttt{b}_4), (\texttt{a}_3, \bot)\}$, which are constructed in Step 3. Thus $(\texttt{b}_3)$, $(\texttt{b}_4)$, and $(\bot)$ should be added to PDL^2_2 in Step 4. Similarly, Steps 3 and 4 are executed for $(\bot) \in PDL^1_1$. Finally, we obtain $PDL^2_2 = \{(\texttt{b}_3), (\texttt{b}_4), (\bot)\}$.

For $(\texttt{b}_3) \in PDL^2_2$, since $\texttt{b}_4 \notin Cl(v_2) = \{\texttt{b}_1, \texttt{b}_2, \texttt{b}_3\}$ where $Msc(v_2) = \texttt{b}_3$, it holds that $\Sigma \not\models_{\text{MSC}} \texttt{a}_3(\texttt{A.B} : \texttt{b}_4)$. On the other hand, since $\texttt{b}_2 \in Cl(v_2)$ for every list in PDL^2_2, it holds that $\Sigma \models_{\text{MSC}} \texttt{a}_3(\texttt{A.B} : \texttt{b}_2)$. These facts was also shown in Example 15. □

We prove that PDL^{i+1}_{i+l} is correctly computed by Procedure 7. Let $v_i \xrightarrow{P_{i+1}} v_{i+1} \xrightarrow{P_{i+2}} \cdots \xrightarrow{P_{i+l-1}} v_{i+l-1}$ be in PDL^{i}_{i+l-1}. By definition, there is a pd_{i+l-1}-List $v_0 \xrightarrow{P_1} v_1 \xrightarrow{P_2} \cdots \xrightarrow{P_{i+l-1}} v_{i+l-1}$ in PDL_{i+l-1} satisfying the four conditions: MSC, PDL 1, PDL 2, and $C \in Cl(v_0)$. By the for loop on variable k in Step 2, we check exhaustively whether or not the $P_{i+1}.P_{i+2}. \cdots .P_{i+l}$-List $v_i \xrightarrow{P_{i+1}} v_{i+1} \xrightarrow{P_{i+2}} \cdots \xrightarrow{P_{i+l}} v_{i+l}$ satisfies PDLs 1 and 2 for all possible $Msc(v_{i+l})$. Clearly, the $P_{i+1}.P_{i+2}. \cdots .P_{i+l}$-List satisfies MSC. Thus by Lemma 13, PDL^{i+1}_{i+l} can be computed by Procedure 7.

Now consider the time complexity of Procedure 7. Since a $P_{i+1}.P_{i+2} \cdots .P_{i+l-1}$-List consists of l vertices, the number of lists in PDL^{i}_{i+l-1} is at most $(K+1)^l$. Thus by the for loop of Step 2, Steps 3 and 4 are executed at most $(K+1)^{l+1}$ times, that is, $O(K^{l+1})$ times. Step 3 can be executed in $O(D)$ time, since $Cl(v_{i+l})$ can be computed in $O(D)$ time. Since the size of a $P_{i+1}.P_{i+2}. \cdots .P_{i+l}$-List is $O(D \cdot l)$, it can be tested in $O(D \cdot l)$ time whether or not the $P_{i+1}.P_{i+2}. \cdots .P_{i+l}$-List constructed in Step 4 satisfies PDLs 1 and 2. That is, one execution of Steps 3 and 4 can be done in $O(D \cdot l)$ time. Hence Procedure 7 can be executed in $O(K^{l+1} \cdot D \cdot l)$ time.

In order to decide whether or not $\Sigma \models_{\text{MSC}} C(pd : C')$, we want to compute PDL^{n-l+1}_{n} by Procedure 7. Initially, we must compute PDL^{1}_{l}. By Procedure 6, PDL_l can be computed in $O(K^{l+1} \cdot D \cdot l)$ time. After that, PDL^{1}_{l} can be constructed by removing vertex v_0 and its incident arc $v_0 \xrightarrow{P_1} v_1$ for each pd_l-List $v_0 \xrightarrow{P_1} v_1 \xrightarrow{P_2} \cdots \xrightarrow{P_l} v_l$ in PDL_l. This can be done in $O(K^{l+1} \cdot D \cdot l)$ time, since the size of PDL_l is $O(K^{l+1} \cdot D \cdot l)$. Thus PDL^{1}_{l} can be computed in $O(K^{l+1} \cdot D \cdot l)$ time.

By executing Procedure 7 for $2 \leq i \leq n-l+1$, we can compute PDL^{n-l+1}_{n}. It takes $O(K^{l+1} \cdot D \cdot l \cdot (n-l+1))$ time, since each execution of Procedure 7 takes $O(K^{l+1} \cdot D \cdot l)$ time. Hence PDL^{n-l+1}_{n} can be computed in $O(K^{l+1} \cdot D \cdot l \cdot (len(pd)+1))$ time, where $n = len(pd)$. Since the size of PDL^{n-l+1}_{n} is $O(K^{l+1} \cdot D \cdot l)$, it can be decided in $O(K^{l+1} \cdot D \cdot l)$ time whether or not there is a $P_{n-l+2}.P_{n-l+3}. \cdots .P_n$-List $v_{n-l+1} \xrightarrow{P_{n-l+2}} v_{n-l+2} \xrightarrow{P_{n-l+3}} \cdots \xrightarrow{P_n} v_n$ in PDL^{n-l+1}_{n} such that $C' \notin Cl(v_n)$.

Therefore, it can be decided in in $O(K^{l+1} \cdot D \cdot l \cdot (len(pd)+1))$ time whether or not $\Sigma \models_{\text{MSC}} C(pd : C')$. Consequently, Theorem 7(a) also holds in the case that $len(pd) > l$. This completes proving Theorem 7(a). □

Proof of Theorem 7(b):

Since $\mathtt{Id}$ is trivially in $PathDescs_{\mathrm{MSC}}(C)$, assume that $pd \neq \mathtt{Id}$. By Lemma 12, $pd \notin PathDescs_{\mathrm{MSC}}(C)$ if and only if there is a pd_i-List satisfying the six conditions of Lemma 12. We first show that given an integer i with $i < len(pd)$, it can be decided in $O(K^{i+1} \cdot D \cdot (i+1))$ time whether or not there is a pd_j-List satisfying the six conditions of Lemma 12 and $j \leq i$.

Intuitively, this can be done by (1) generating all pd_i-Lists satisfying MSC and (2) checking whether or not there is a pd_j-List $v_0 \xrightarrow{P_1} v_1 \xrightarrow{P_2} \cdots \xrightarrow{P_j} v_j$ satisfying the six conditions of Lemma 12 for each generated pd_i-List $v_0 \xrightarrow{P_1} v_1 \xrightarrow{P_2} \cdots \xrightarrow{P_i} v_i$. By slightly modifying Procedure 6, we can execute Step 1. Consider Step 2. Let $v_0 \xrightarrow{P_1} v_1 \xrightarrow{P_2} \cdots \xrightarrow{P_i} v_i$ be a pd_i-List satisfying MSC. We must check whether or not there is an index j such that the pd_j-List $v_0 \xrightarrow{P_1} v_1 \xrightarrow{P_2} \cdots \xrightarrow{P_j} v_j$ satisfies the six conditions of Lemma 12. This can be done by the following procedure.

Procedure 8: (Deciding whether or not there is an index j such that the pd_j-List $v_0 \xrightarrow{P_1} v_1 \xrightarrow{P_2} \cdots \xrightarrow{P_j} v_j$ satisfies the six conditions of Lemma 12.)
input: a class $C \in Classes(S)$, a path description pd $(= P_1.P_2. \cdots P_n)$, and a pd_i-List $v_0 \xrightarrow{P_1} v_1 \xrightarrow{P_2} \cdots \xrightarrow{P_i} v_i$ satisfying MSC.

1. **for** $j \leftarrow 0$ **to** i **do**
2. **if** $C \in Cl(v_0)$, $Cl(v_j) \cap Dom(P_{j+1}) = \emptyset$, and the pd_j-List $v_0 \xrightarrow{P_1} v_1 \xrightarrow{P_2} \cdots \xrightarrow{P_j} v_j$ satisfies $\Sigma_{\mathtt{SC}}$ **then** return YES. □

Example 17: Consider Example 14. Let us execute Procedure 8 for class $\mathtt{a}_1$, path description `A.B.C`, and the '`A.B`'-List given in Figure 17. Then YES should be returned when $j = 2$. In fact, the '`A.B`'-List satisfies all the conditions of Lemma 12. □

We prove that Procedure 8 returns YES if and only if there is an index j such that the pd_j-List satisfies the six conditions of Lemma 12. Assume that Procedure 8 returns YES when $j = m$. We must prove that the pd_m-List $v_0 \xrightarrow{P_1} v_1 \xrightarrow{P_2} \cdots \xrightarrow{P_m} v_m$ satisfies the six conditions of Lemma 12. By the if condition of Step 2, the pd_m-List satisfies conditions (2), (3), and (6) of Lemma 12. Since the pd_i-List satisfies MSC, so does the pd_m-List; that is, the pd_m-List satisfies condition (4). Since $m < i \leq len(pd)$, the pd_m-List satisfies conditions (1). It remains to show that the pd_m-List satisfies condition (5). Assume that the pd_m-List does not satisfy condition (5). Let m' be the smallest index such that the $pd_{m'}$-List satisfies condition (5). Then $m' < m$. Since the pd_m-List satisfies Σ_{SC} and $C \in Cl(v_0)$, so does the $pd_{m'}$-List. By the minimality of m', it holds that $Cl(v_{m'}) \cap Dom(P_{m'+1}) = \emptyset$. Thus the if condition of Step 2 should hold when $j = m'$; that is, Procedure 8 should return YES when $j = m'$. Contradiction. Hence the pd_m-List satisfies the six conditions of Lemma 12. Conversely, assume that there is an index m such that the pd_m-List satisfies the six conditions of Lemma 12. It is easy to see that Procedure 8 returns YES when $j = m$. This completes proving the correctness of Procedure 8.

We now estimate the time complexity. By slightly modifying Procedure 6, we can generate all pd_i-Lists $v_0 \xrightarrow{P_1} v_1 \xrightarrow{P_2} \cdots \xrightarrow{P_l} v_i$ satisfying MSC in $O(K^{i+1} \cdot D \cdot (i+1))$ time. After that, Procedure 8 is executed for each generated pd_i-List. Since the number of pd_i-Lists is at most $(K+1)^{i+1}$, Procedure 8 is executed at most $(K+1)^{i+1}$ times, that is, $O(K^{l+1})$ times. Furthermore, since the size of a pd_i-List is $O(D \cdot (i+1))$, Procedure 8 can be executed in $O(D \cdot (i+1))$ time. Thus the total time for Procedure 8 is $O(K^{i+1} \cdot D \cdot (i+1))$. Consequently, it can be decided in $O(K^{i+1} \cdot D \cdot (i+1))$ time whether or not there is a pd_j-List satisfying the six conditions of Lemma 12 and $j \leq i$.

Consider how to decide whether or not $pd \in PathDescs_{\mathrm{MSC}}(C)$. By the discussions above, it can be decided in $O(K^{len(pd)+1} \cdot D \cdot (len(pd)+1))$ time whether or not there is a pd_j-List satisfying the six conditions of Lemma 12 and $j < len(pd)$. That is, it can be decided in $O(K^{len(pd)+1} \cdot D \cdot (len(pd)+1))$ time whether or not $pd \in PathDescs_{\mathrm{MSC}}(C)$. Hence if $len(pd) \leq l$, then Theorem 7(b) holds.

Assume that $len(pd) > l$. By the discussions above, it can be decided in $O(K^{l+1} \cdot D \cdot l)$ time whether or not there is a pd_j-List satisfying the six conditions of Lemma 12 and $j < l$. It remains to check whether or not there is a pd_j-List satisfying the six conditions of Lemma 12 for some j such that $l \leq j \leq n-1$. Note that PDL^i_{i+l-1} is the set of $P_{i+1}.P_{i+2}.\cdots.P_{i+l-1}$-Lists such that $v_i \xrightarrow{P_{i+1}} v_{i+1} \xrightarrow{P_{i+2}} \cdots \xrightarrow{P_{i+l-1}} v_{i+l-1}$ is in PDL^i_{i+l-1} if and only if there is a pd_{i+l-1}-List $v_0 \xrightarrow{P_1} v_1 \xrightarrow{P_2} \cdots \xrightarrow{P_{i+l}} v_{i+l-1}$ satisfying the four conditions: MSC, PDL 1, PDL 2, and $C \in Cl(v_0)$. Thus there is a pd_{i+l-1}-List satisfying the six conditions of Lemma 12 if and only if there is a $P_{i+1}.P_{i+2}.\cdots.P_{i+l-1}$-List $v_i \xrightarrow{P_{i+1}} v_{i+1} \xrightarrow{P_{i+2}} \cdots \xrightarrow{P_{i+l-1}} v_{i+l-1}$ in PDL^i_{i+l-1} satisfying $Cl(v_{i+l-1}) \cap Dom(P_{i+l}) = \emptyset$. That is, by using PDL^i_{i+l-1} for $1 < i \leq n-l$, we can check whether or not there is a pd_j-List satisfying the six conditions of Lemma 12 for some j such that $l \leq j \leq n-1$. Since the size of PDL^i_{i+l-1} is $O(K^{l+1} \cdot D \cdot l)$, it can be decided in $O(K^{l+1} \cdot D \cdot l)$ time whether or not there is a $P_{i+1}.P_{i+2}. \cdots .P_{i+l-1}$-List $v_i \xrightarrow{P_{i+1}} v_{i+1} \xrightarrow{P_{i+2}} \cdots \xrightarrow{P_{i+l-1}} v_{i+l-1}$ in PDL^i_{i+l-1} satisfying $Cl(v_{i+l-1}) \cap Dom(P_{i+l}) = \emptyset$. Thus we can check in $O(K^{l+1} \cdot D \cdot l \cdot (len(pd)+1))$ time, provided that PDL^i_{i+l-1} is known for all i such that $1 \leq i \leq n-1$. Note that when deciding whether or not $\Sigma \models_{\text{MSC}} C(pd : C')$, we have computed PDL^i_{i+l-1} for all i in $O(K^{l+1} \cdot D \cdot l \cdot (len(pd)+1))$ time. As a result, it can be checked in $O(K^{l+1} \cdot D \cdot l \cdot (len(pd)+1))$ time whether or not there is a pd_j-List satisfying the six conditions of Lemma 12 for some j such that $l \leq j \leq n-1$. Consequently, Theorem 7(b) also holds in the case that $len(pd) > l$. This completes proving Theorem 7(b). □

Proof of Theorem 7(c):
Consider how to decide whether or not $pd \in PathFuncs_{\text{MSC}}(C)$. Since it can be decided in $O(D \cdot (len(pd)+1))$ time whether or not $\texttt{FUNC}(P_i) \in \Sigma_{\texttt{FUNC}}$ for $1 \leq i \leq n$, Theorem 7(c) follows from Corollary 1 and Theorem 7(b). □

4. MSC with the Lower Semilattice Condition

If we restrict our attention to interpretations which satisfy MSC, then the decision problems related to SCs are NP-complete, as shown in Section 3.1. In this section, we consider a special case in which a given database schema $\langle S, \Sigma \rangle$ satisfies the following additional condition.

The lower semilattice condition: A class C is called a *greatest common subclass* of C_1 and C_2 if and only if the following two conditions hold.

1. $\Sigma \models C(\texttt{Id} : C_1)$ and $\Sigma \models C(\texttt{Id} : C_2)$.

2. If there is a class $C_3 \in Classes(S)$ such that $\Sigma \models C_3(\texttt{Id} : C_1)$ and $\Sigma \models C_3(\texttt{Id} : C_2)$, then $\Sigma \models C_3(\texttt{Id} : C)$.

Then $\langle S, \Sigma \rangle$ satisfies the lower semilattice condition if and only if for all $C_1, C_2 \in Classes(S)$, there is a *unique* greatest common subclass of C_1 and C_2, denoted $C_1 \sqcap C_2$.[15] □

For example, the taxonomy illustrated in Figure 14 satisfies the lower semilattice condition (if the taxonomy is considered as a database schema

[15]This condition implies that $Classes(S)$ contains a bottom class $\perp$.

by itself). On the other hand, the taxonomy illustrated in Figure 13 does not satisfy the lower semilattice condition, since classes A_i and B_i do not have a unique greatest common subclass. Thus the whole database schema constructed in Lemma 11 does not satisfy the lower semilattice condition, either. Through this section, we consider the case that $\langle S, \Sigma \rangle$ satisfies the lower semilattice condition.

4.1 Axiomatization

In this section, it will be shown that the following axiom together with A1 and A2 in Section 2.2 are sound and complete for deciding whether or not $\Sigma \models_{\text{MSC}} C(pd : C')$, provided that $pd \in \mathit{PathDescs}_{\text{MSC}}(C)$.

A4: (*restriction*) If $C_1(pd : C_2)$ and $C_1(pd : C_3)$, then $C_1(pd : C_2 \sqcap C_3)$.

Soundness of axiom A4 is straightforward. The proof of completeness is analogous to Theorem 2 in Section 2.2. In particular, if $\Sigma \not\vdash_{\{\text{A1,A2,A4}\}} C(pd : C')$, then we will construct a *pd*-List satisfying MSC as well as PDLs 1 to 3. (Thus $\Sigma \not\models_{\text{MSC}} C(pd : C')$ by Theorem 5.) We need to modify the concept of chase in order to satisfy MSC.

The *MSC-chase* of C and pd under Σ_{SC}, written $\mathit{Chase}^{\text{MSC}}_{\Sigma_{\mathsf{SC}}}(C, pd)$, is a *pd*-List $v_0 \xrightarrow{P_1} v_1 \xrightarrow{P_2} \cdots \xrightarrow{P_n} v_n$ obtained by the following procedure, where $pd = P_1.P_2.\cdots.P_n$.

Procedure 9: (Computing $\mathit{Chase}^{\text{MSC}}_{\Sigma_{\mathsf{SC}}}(C, pd)$.)
input: a database schema $\langle S, \Sigma \rangle$ satisfying the lower semilattice condition, a class $C \in \mathit{Classes}(S)$, and a path description pd $(= P_1.P_2.\cdots.P_n)$.

1. Construct a *pd*-List $v_0 \xrightarrow{P_1} v_1 \xrightarrow{P_2} \cdots \xrightarrow{P_n} v_n$ such that $\mathit{Cl}(v_0) = \{ C \}$ and $\mathit{Cl}(v_i) = \emptyset$ for $1 \le i \le n$.

2. Apply the following rule to the *pd*-List exhaustively. Note that the *pd*-List always satisfies MSC, as will be shown in Lemma 14 below.

   MSC-SC-rule: For an SC $C_a(pd' : C_b) \in \Sigma_{\mathsf{SC}}$, if there are two vertices v_i, v_j such that $C_a \in \mathit{Cl}(v_i)$, $C_b \notin \mathit{Cl}(v_j)$, and pd' has the form $P_{i+1}.P_{i+2}.\cdots.P_j$, then add not only C_b but also $\mathit{Msc}(v_j) \sqcap C_b$ to $\mathit{Cl}(v_j)$.[16] For convenience, if $\mathit{Cl}(v_j) = \emptyset$, then let $\mathit{Msc}(v_j) \sqcap C_b = C_b$. □

[16] $\mathit{Msc}(v_j) \sqcap C_b$ is unique, since $\langle S, \Sigma \rangle$ satisfies the lower semilattice condition.

Example 18: Let $\langle S, \Sigma\rangle$ be a database schema illustrated in Figure 18, where $Dom(\mathtt{A}) = \{\mathtt{a}_1\}$, $Dom(\mathtt{B}) = \{\mathtt{a}_3\}$, $Dom(\mathtt{C}) = \{\mathtt{b}_2\}$. For class $\mathtt{a}_1$ and path description A.B.C, let us execute Procedure 9.

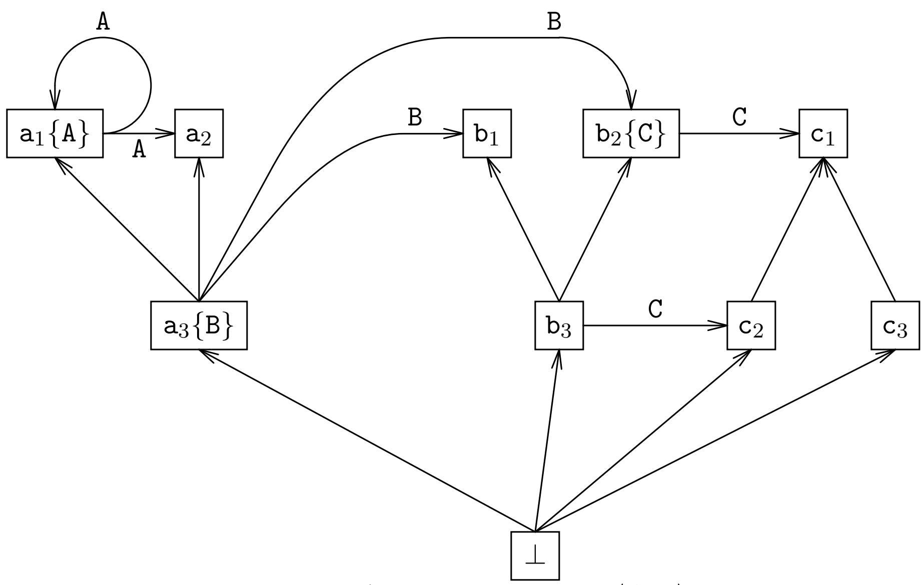

Figure 18: A database schema $\langle S, \Sigma\rangle$.

In Step 1, an 'A.B.C'-List $v_0 \xrightarrow{\mathtt{B}} v_1 \xrightarrow{\mathtt{A}} v_2 \xrightarrow{\mathtt{C}} v_3$ is constructed, where $Cl(v_0) = \{\mathtt{a}_1\}$ and $Cl(v_i) = \emptyset$ for $1 \leq i \leq 3$.

For SC $\mathtt{a}_1(\mathtt{A} : \mathtt{a}_1)$ in $\Sigma_{\mathtt{SC}}$, since $\mathtt{a}_1 \in Cl(v_0)$ and $\mathtt{a}_1 \notin Cl(v_1)$, class $\mathtt{a}_1$ is added to $Cl(v_1)$ by applying the MSC-SC-rule for $\mathtt{a}_1(\mathtt{A} : \mathtt{a}_1)$, where $Msc(v_1) \sqcap \mathtt{a}_1 = \mathtt{a}_1$ since $Cl(v_1) = \emptyset$. The MSC of v_1 becomes $\mathtt{a}_1$ by the application. For SC $\mathtt{a}_1(\mathtt{A} : \mathtt{a}_2)$ in $\Sigma_{\mathtt{SC}}$, since $\mathtt{a}_1 \in Cl(v_0)$ and $\mathtt{a}_2 \notin Cl(v_1)$, classes $Msc(v_1) \sqcap \mathtt{a}_2$ as well as $\mathtt{a}_2$ are added to $Cl(v_1)$ by applying the MSC-SC-rule for $\mathtt{a}_1(\mathtt{A} : \mathtt{a}_2)$, where $Msc(v_1) \sqcap \mathtt{a}_2 = \mathtt{a}_1 \sqcap \mathtt{a}_2 = \mathtt{a}_3$. The MSC of v_1 changes from $\mathtt{a}_1$ to $\mathtt{a}_3$ by the application. Finally, we obtain the 'A.B.C'-List given in Figure 19 as $Chase^{\mathrm{MSC}}_{\Sigma_{\mathtt{SC}}}(\mathtt{a}_1, \mathtt{A.B.C})$, where each vertex is labeled its MSC. Note that $Chase^{\mathrm{MSC}}_{\Sigma_{\mathtt{SC}}}(\mathtt{a}_1, \mathtt{A.B.C})$ satisfies $\Sigma_{\mathtt{SC}}$. □

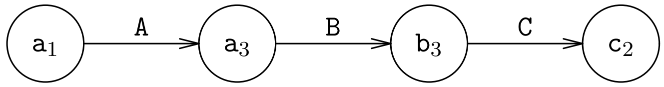

Figure 19: $Chase^{\mathrm{MSC}}_{\Sigma_{\mathtt{SC}}}(\mathtt{a}_1, \mathtt{A.B.C})$.

It will be shown that if $\Sigma \not\vdash_{\{\mathrm{A1,A2,A4}\}} C(pd : C')$ and $pd \in PathDescs_{\mathrm{MSC}}(C)$, then $Chase^{\mathrm{MSC}}_{\Sigma_{\mathtt{SC}}}(C, pd)$ satisfies MSC as well as PDLs 1 to 3.

Lemma 14: $\mathit{Chase}^{\mathrm{MSC}}_{\Sigma_{\mathsf{SC}}}(C, pd)$ satisfies MSC.

Proof. We prove by induction on the number of applying MSC-SC-rules in Step 2 that the *pd*-List $v_0 \xrightarrow{P_1} v_1 \xrightarrow{P_2} \cdots \xrightarrow{P_n} v_n$ satisfies MSC during Procedure 9. Thus Lemma 14 will hold.

The basis is trivial, since $\mathit{Cl}(v_0) = \{ C \}$ and $\mathit{Cl}(v_i) = \emptyset$ for $1 \leq i \leq n$ in Step 1.

As an induction hypothesis, assume that the *pd*-List satisfies MSC during an execution of Step 2. Let $\mathit{Msc}(v_j) = C_j$ and assume that $C_j \sqcap C_b$ as well as C_b are added to $\mathit{Cl}(v_j)$ by applying an MSC-SC-rule to the *pd*-List. We show that $\mathit{Msc}(v_j)$ is changed from C_j to $C_j \sqcap C_b$ by the application. Since $\mathit{Msc}(v_j) = C_j$ before the application, $\Sigma \models C_j(\mathtt{Id} : C'_j)$ for every $C'_j \in \mathit{Cl}(v_j) - \{C_b, C_j \sqcap C_b\}$. By definition, $\Sigma \models C_j \sqcap C_b(\mathtt{Id} : C_j)$ and $\Sigma \models C_j \sqcap C_b(\mathtt{Id} : C_b)$. Thus $\Sigma \models C_j \sqcap C_b(\mathtt{Id} : C'_j)$ for every $C'_j \in \mathit{Cl}(v_j)$. That is, $\mathit{Msc}(v_j) = C_j \sqcap C_b$ after the application. Hence the *pd*-List still satisfies MSC. This completes the induction proof. □

Consider PDL 2. By the definition of MSC-SC-rule, unless the *pd*-List satisfies Σ_{SC}, Procedure 9 does not terminate. Thus $\mathit{Chase}^{\mathrm{MSC}}_{\Sigma_{\mathsf{SC}}}(C, pd)$ satisfies Σ_{SC}; that is, PDL 2 holds.

Consider PDL 3. By Step 1 of Procedure 9, it holds that $C \in \mathit{Cl}(v_0)$. By the following lemma, if $\Sigma \not\vdash_{\{\mathrm{A1,A2,A4}\}} C(pd : C')$, then it holds that $C' \notin \mathit{Cl}(v_n)$. Thus if $\Sigma \not\vdash_{\{\mathrm{A1,A2,A4}\}} C(pd : C')$, then $\mathit{Chase}^{\mathrm{MSC}}_{\Sigma_{\mathsf{SC}}}(C, pd)$ satisfies PDL 3.

Lemma 15: $\mathit{Cl}(v_i) = \{C_i \in \mathit{Classes}(S) \mid \Sigma_{\mathsf{SC}} \vdash_{\{\mathrm{A1,A2,A4}\}} C(pd_i : C_i)\}$ for $0 \leq i \leq n$.

Proof. The proof is analogous to Lemma 5.

We first prove that if $\Sigma_{\mathsf{SC}} \vdash_{\{\mathrm{A1,A2,A4}\}} C(pd_i : C_i)$, then $C_i \in \mathit{Cl}(v_i)$. Assume that $\Sigma_{\mathsf{SC}} \vdash_{\{\mathrm{A1,A2,A4}\}} C(pd_i : C_i)$. Then $\Sigma_{\mathsf{SC}} \models_{\mathrm{MSC}} C(pd_i : C_i)$ by soundness of the axioms. Thus $\mathit{Chase}^{\mathrm{MSC}}_{\Sigma_{\mathsf{SC}}}(C, pd)$ satisfies $C(pd_i : C_i)$, since it satisfies Σ_{SC} by PDL 2. Hence $C \in \mathit{Cl}(v_0)$ implies $C_i \in \mathit{Cl}(v_i)$.

We next prove that if $C_i \in \mathit{Cl}(v_i)$, then $\Sigma_{\mathsf{SC}} \vdash_{\{\mathrm{A1,A2,A4}\}} C(pd_i : C_i)$. If $\mathit{Cl}(v_i) = \emptyset$, then there is nothing to prove. Assume that $C_i \in \mathit{Cl}(v_i)$. By definition, $\Sigma \models \mathit{Msc}(v_i)(\mathtt{Id} : C_i)$. Thus by Theorem 2, $\Sigma \vdash_{\{\mathrm{A1,A2}\}} \mathit{Msc}(v_i)(\mathtt{Id} : C_i)$, since $\mathtt{Id} \in \mathit{PathDescs}(C)$. Hence in order to prove that $\Sigma_{\mathsf{SC}} \vdash_{\{\mathrm{A1,A2,A4}\}} C(pd_i : C_i)$, it suffices to show that $\Sigma \vdash_{\{\mathrm{A1,A2,A4}\}} C(pd_i : \mathit{Msc}(v_i))$. Induction on the number of applying MSC-SC-rules in Step 2.

The basis follows from axiom A1, since $Cl(v_0) = \{ C \}$ and $Cl(v_i) = \emptyset$ for $1 \leq i \leq n$ in Step 1.

As an induction hypothesis, assume that $\Sigma \vdash_{\{A1,A2,A4\}} C(pd_i : Msc(v_i))$ during an execution of Step 2, where $Cl(v_i) \neq \emptyset$. Assume that $Msc(v_j) \sqcap C_b$ as well as C_b should be added to $Cl(v_j)$ by applying an MSC-SC-rule for $C_a(pd' : C_b) \in \Sigma_{\mathsf{SC}}$. Then by the definition of SC-rule, it holds that $C_b \notin Cl(v_j)$, $C_a \in Cl(v_k)$, and $pd' = P_{k+1}.P_{k+2}. \cdots .P_j$ for some k. It suffices to show that $\Sigma \vdash_{\{A1,A2,A4\}} C(pd_j : Msc(v_j) \sqcap C_b)$, since the MSC of v_j is changed from $Msc(v_j)$ to $Msc(v_j) \sqcap C_b$. Since $C_a \in Cl(v_k)$, it follows from the induction hypothesis that $\Sigma \vdash_{\{A1,A2,A4\}} C(pd_k : C_a)$. By axiom A2, $C(pd_k : C_a)$ and $C_a(pd' : C_b)$ imply $C(pd_k \circ pd_j : C_b)$, where $pd_k \circ pd' = pd_j$. That is, $\Sigma \vdash_{\{A1,A2,A4\}} C(pd_j : C_b)$. Furthermore, $\Sigma \vdash_{\{A1,A2,A4\}} C(pd_j : Msc(v_j))$ by the induction hypothesis. Thus $C(pd_j : Msc(v_j))$ and $C(pd_j : C_b)$ imply $C(pd_j : Msc(v_j) \sqcap C_b)$ by axiom A4. Hence $\Sigma \vdash_{\{A1,A2,A4\}} C(pd_j : Msc(v_j) \sqcap C_b)$. This completes the induction proof. Consequently, Lemma 15 holds. □

By the following lemma, if $pd \in PathDescs_{\text{MSC}}(C)$, then $Chase^{\text{MSC}}_{\Sigma_{\mathsf{SC}}}(C, pd)$ satisfies PDL 3.

Lemma 16: $pd \in PathDescs_{\text{MSC}}(C)$ if and only if $Chase^{\text{MSC}}_{\Sigma_{\mathsf{SC}}}(C, pd)$ satisfies a property value integrity condition.

Proof. (Almost the same as proving Lemma 6.) □

Consequently, we have the following theorem, which can be proved in the same way as proving Theorem 2.

Theorem 8: Assume that $\langle S, \Sigma \rangle$ satisfies the lower semilattice condition. If $pd \in PathDescs_{\text{MSC}}(C)$, then the following three statements are equivalent.[17]

1. $\Sigma \models_{\text{MSC}} C(pd : C')$.
2. $\Sigma \vdash_{\{A1,A2,A4\}} C(pd : C')$.
3. $C' \in Cl(v_n)$. □

[17] Σ is not necessarily well-formed with respect to $\langle S, \Sigma \rangle$.

Example 19: Consider Example 18. We have obtained the '`A.B.C`'-List in Figure 19 as $\mathit{Chase}^{\mathrm{MSC}}_{\Sigma_{\mathtt{SC}}}(\mathtt{a}_1, \mathtt{A.B.C})$. The '`A.B.C`'-List satisfies a property value integrity condition. Thus $\mathtt{A.B.C} \in \mathit{PathDescs}_{\mathrm{MSC}}(\mathtt{a}_1)$ by Lemma 16. Since $\mathtt{c}_2 \in \mathit{Cl}(v_2) = \{\mathtt{c}_1, \mathtt{c}_2\}$ where $\mathit{Msc}(v_2) = \mathtt{c}_2$, it follows from Theorem 8 that $\Sigma \models_{\mathrm{MSC}} \mathtt{a}_1(\mathtt{A.B.C} : \mathtt{c}_2)$. On the other hand, $\Sigma \not\models_{\mathrm{MSC}} \mathtt{a}_1(\mathtt{A.B.C} : \mathtt{c}_3)$ by Theorem 8, since $\mathtt{c}_3 \notin \mathit{Cl}(v_2)$. □

4.2 The three decision problems

In this section, we will prove the following theorem.

Theorem 9: Assume that $\langle S, \Sigma \rangle$ satisfies the lower semilattice condition. The following three decision problems are solved in $O(D \cdot (\mathit{len}(pd) + 1))$ time, where D is the size of $\langle S, \Sigma \rangle$.

a. $\Sigma \models_{\mathrm{MSC}} C(pd : C')$ (provided that $pd \in \mathit{PathDescs}_{\mathrm{MSC}}(C)$) ?

b. $pd \in \mathit{PathDescs}_{\mathrm{MSC}}(C)$?

c. $pd \in \mathit{PathFuncs}_{\mathrm{MSC}}(C)$? □

As in Section 2.3, the time for computing $\mathit{Chase}^{\mathrm{MSC}}_{\Sigma_{\mathtt{SC}}}(C, pd)$ dominates the time complexities of the three decision problems. In the following, we will present a procedure for computing $\mathit{Chase}^{\mathrm{MSC}}_{\Sigma_{\mathtt{SC}}}(C, pd)$. After that, it will be proved that the procedure can be executed in $O(D \cdot (\mathit{len}(pd) + 1))$ time. Hence Theorem 10 can be proved in the same way as Theorem 4.

The procedure is a modification of Procedure 3. Note that $\mathit{Cl}(v_i)$ is computed from $\mathit{Msc}(v_i)$. In fact, the procedure will compute $\mathit{Msc}(v_i)$ and then $\mathit{Cl}(v_i)$ for $1 \leq i \leq n$.

Procedure 10: (Computing $\mathit{Chase}^{\mathrm{MSC}}_{\Sigma_{\mathtt{SC}}}(C, pd)$.)
input: a database schema $\langle S, \Sigma \rangle$ satisfying the lower semilattice condition, a class $C \in \mathit{Classes}(S)$, and a path description pd $(= P_1.P_2.\cdots.P_n)$.

1. Divide $\Sigma_{\mathtt{SC}}$ into two sets: $\Sigma_{\mathtt{Id}} = \{C_a(pd' : C_b) \in \Sigma_{\mathtt{SC}} \mid pd' = \mathtt{Id}\}$ and $\Sigma_{\neg\mathtt{Id}} = \Sigma_{\mathtt{SC}} - \Sigma_{\mathtt{Id}}$.

2. Let $\mathit{CL}_0 \leftarrow \{C_0 \in \mathit{Classes}(S) \mid \Sigma_{\mathtt{Id}} \vdash_{\{\mathrm{A1,A2}\}} C(\mathtt{Id} : C_0)\}$.

3. **for** $i \leftarrow 1$ **to** n

   do begin

4. Let $CL \leftarrow \{C_b \in \mathit{Classes}(S) \mid$ there is an SC $C_a(pd' : C_b) \in \Sigma_{\neg\mathtt{Id}}$ such that $pd' = P_{j+1}.P_{j+2}.\cdots.P_i$ and $C_a \in CL_j$ for some j $\}$.
5. **if** $CL = \emptyset$ **then** let $CL_i \leftarrow \emptyset$

   else begin

   6. Let $M_i \leftarrow \sqcap_{C_b \in CL} C_b$.[18]
   7. Let $CL_i \leftarrow \{C_i \in \mathit{Classes}(S) \mid \Sigma_{\mathtt{Id}} \vdash_{\{\mathrm{A1,A2}\}} M_i(\mathtt{Id} : C_i)\}$.

   end

end □

The correctness of Procedure 10 follows from the following lemma.

Lemma 17: $CL_i = Cl(v_i)$ for $0 \leq i \leq n$, where $\mathit{Chase}^{\mathrm{MSC}}_{\Sigma_{\mathsf{SC}}}(C, pd)$ is $v_0 \xrightarrow{P_1} v_1 \xrightarrow{P_2} \cdots \xrightarrow{P_n} v_n$.

Proof. The proof is analogous to Lemma 7. By Lemma 15, it suffices to show that

$$CL_i = \{C_i \in \mathit{Classes}(S) \mid \Sigma_{\mathsf{SC}} \vdash_{\{\mathrm{A1,A2,A4}\}} C(pd_i : C_i)\}. \tag{4.1}$$

We prove 4.1 by induction on i.

Basis. Consider the case that $i = 0$. Note that Steps 1 and 2 of Procedure 10 is the same as Steps 1 and 2 of Procedure 3. Thus it follows from Lemmas 5 and 7 that $CL_0 = \{C_0 \in \mathit{Classes}(S) \mid \Sigma_{\mathsf{SC}} \vdash_{\{\mathrm{A1,A2}\}} C(\mathtt{Id} : C_0)\}$. Hence v_0 satisfies Σ_{SC} by itself. Clearly, $\mathit{Msc}(v_0) = C$; that is, v_0 satisfies MSC. These facts imply that no class is added to CL_0 by applying axiom A4. Thus 4.1 holds for $i = 0$.

Induction. As an induction hypothesis, assume that 4.1 holds for $j \leq i-1$, where $i \geq 1$.

Proof of '$\subseteq$': We prove that if $C_i \in CL_i$, then $\Sigma_{\mathsf{SC}} \vdash_{\{\mathrm{A1,A2,A4}\}} C(pd_i : C_i)$. If $CL_i = \emptyset$, then there is nothing to prove. Assume that $CL_i \neq \emptyset$. We claim that $\Sigma_{\mathsf{SC}} \vdash_{\{\mathrm{A1,A2,A4}\}} C(pd_i : C_b)$ for every $C_b \in CL$.

Let $C_b \in CL$. By Step 4, there is an SC $C_a(pd' : C_b) \in \Sigma_{\neg\mathtt{Id}}$ such that $pd' = P_{j+1}.P_{j+2}.\cdots.P_i$ and $C_a \in CL_j$ for some j. Since $pd' \neq \mathtt{Id}$ by the definition of $\Sigma_{\neg\mathtt{Id}}$, it holds that $j \leq i-1$. Thus by the induction hypothesis, $C_a \in CL_j$ implies that $\Sigma_{\mathsf{SC}} \vdash_{\{\mathrm{A1,A2,A4}\}} C(pd_j : C_a)$. By axiom A2, $C(pd_j : C_a)$

[18] M_i is well-defined, since $\langle S, \Sigma \rangle$ satisfies the lower semilattice condition.

and $C_a(pd' : C_b)$ imply $C(pd_j \circ pd' : C_b)$, where $pd_j \circ pd' = pd_i$. That is, $\Sigma_{\mathtt{SC}} \vdash_{\{\mathrm{A1,A2,A4}\}} C(pd_i : C_b)$. Hence the claim holds.

Note that $M_i \leftarrow \sqcap_{C_b \in CL} C_b$ by Step 6. Thus $C(pd_i : M_i)$ can be derived by applying axiom A4 repeatedly to $\{C(pd_i : C_b) \mid C_b \in CL\}$. Hence by the claim above, it holds that $\Sigma_{\mathtt{SC}} \vdash_{\{\mathrm{A1,A2,A4}\}} C(pd_i : M_i)$. Since $M_i = Msc(v_i)$ by Step 7, this implies that $\Sigma_{\mathtt{SC}} \vdash_{\{\mathrm{A1,A2,A4}\}} C(pd_i : C_i)$ for every $C_i \in CL_i$. This completes the proof of '$\subseteq$'.

Proof of '$\supseteq$': We prove that if $\Sigma_{\mathtt{SC}} \vdash_{\{\mathrm{A1,A2,A4}\}} C(pd_i : C_i)$, then $C_i \in CL_i$. Assume that $\Sigma_{\mathtt{SC}} \vdash_{\{\mathrm{A1,A2,A4}\}} C(pd_i : C_i)$. Then $\Sigma_{\mathtt{SC}} \models_{\mathrm{MSC}} C(pd_i : C_i)$ by soundness of the axioms. Let $u_0 \xrightarrow{P_1} u_1 \xrightarrow{P_2} \cdots \xrightarrow{P_i} u_i$ be a pd_i-List such that $Cl(u_j) = CL_j$ for $0 \le j \le i$. We will prove that the pd_i-List satisfies $\Sigma_{\mathtt{SC}}$, and thus $C(pd_i : C_i)$. Since $C \in CL_0$, this will imply that $C_i \in CL_i$.

Since $CL_j = Cl(v_j)$ for $0 \le j \le i-1$ by the induction hypothesis and Lemma 15, the pd_{i-1}-List $u_0 \xrightarrow{P_1} u_1 \xrightarrow{P_2} \cdots \xrightarrow{P_i} u_{i-1}$ satisfies $\Sigma_{\mathtt{SC}}$, which can be proved along the same line as proving Lemma 4(a). By Step 4, for every SC $C_a(pd' : C_b) \in \Sigma_{\neg\mathtt{Id}}$, if $pd' = P_{j+1}.P_{j+2}.\cdots.P_i$ and $C_a \in CL_j$, then $C_b \in CL_i$. Thus the pd_i-List satisfies $\Sigma_{\neg\mathtt{Id}}$. Similarly, by Step 7, for every SC $C_a(\mathtt{Id} : C_b) \in \Sigma_{\mathtt{Id}}$, if $C_a \in CL_i$, then $C_b \in CL_i$. Thus the pd_i-List also satisfies $\Sigma_{\mathtt{Id}}$. Since $\Sigma_{\mathtt{SC}} = \Sigma_{\neg\mathtt{Id}} \cup \Sigma_{\mathtt{Id}}$ by definition, the pd_i-List satisfies $\Sigma_{\mathtt{SC}}$. This completes the induction proof. Consequently, Lemma 17 holds. □

Let us estimate the time complexity of Procedure 10. As before, let *Classes*(S) consist of K classes and use a bit array of size K in order to represent a subset of *Classes*(S). Procedure 10 is essentially the same as Procedure 3 except Step 6. One obvious way for executing Step 6 is to construct *in advance* a table containing $C_a \sqcap C_b$ for every pair of $C_a, C_b \in Classes(S)$. Once the table is constructed, Step 6 can be executed in $O(K)$ time, since the size of CL is at most K. Note that $K \le D$. Thus it will be shown in the same way as estimating Procedure 3 that Procedure 10 can be executed in $O(D \cdot (len(pd) + 1))$ time. In the following we will present a procedure for constructing the table in $O(K^3)$ time. The procedure consists of the following three parts.

1. Sort the classes in *Classes*(S) in a *topological order* with respect to the generalization taxonomy. Here, a sequence $C_{t_1}, C_{t_2}, \cdots, C_{t_K}$ is in a topological order if and only if $\Sigma_{\mathtt{Id}} \models C_{t_i}(\mathtt{Id} : C_{t_j})$ implies $i \le j$.

2. Compute $\{C_i \in Classes(S) \mid \Sigma_{\mathtt{Id}} \vdash_{\{\mathrm{A1,A2}\}} C_{t_i}(\mathtt{Id} : C_i)\}$ for $1 \le i \le K$.

3. Compute $C_a \sqcap C_b$ for every pair of $C_a, C_b \in Classes(S)$.

Consider how to execute Part 1. Let $G(V, A)$ be a directed graph such that $V = \mathit{Classes}(S)$ and $A = \{C_a \to C_b \mid C_a(\mathtt{Id} : C_b) \in \Sigma_{\mathtt{Id}}\}$. $G(V, A)$ can be constructed in $O(K + \|\Sigma_{\mathtt{Id}}\|)$ time. Clearly, there is a directed path in $G(V, A)$ from a vertex C_1 to a vertex C_2 if and only if $\Sigma_{\mathtt{Id}} \models C_1(\mathtt{Id} : C_2)$. Since S is a generalization taxonomy with respect to Σ by assumption, $G(V, A)$ contains no directed cycle. Thus the vertices in V can be sorted in a topological order. Let $C_{t_1}, C_{t_2}, \cdots, C_{t_K}$ be a sequence in a topological order; that is, if there is a directed path in $G(V, A)$ from C_{t_i} to C_{t_j}, then $i \leq j$. In other words, if $\Sigma_{\mathtt{Id}} \models C_{t_i}(\mathtt{Id} : C_{t_j})$, then $i \leq j$. Such a sequence $C_{t_1}, C_{t_2}, \cdots, C_{t_K}$ can be obtained in $O(K + \|\Sigma_{\mathtt{Id}}\|)$ time by using $G(V, A)$.[19] Since the number of SCs of the form $C_a(\mathtt{Id} : C_b)$ is at most K^2, the size of $\Sigma_{\mathtt{Id}}$ is $O(K^2)$. Hence Part 1 can be execute in $O(K^2)$ time.

Consider Part 2. Since $\{C_i \in \mathit{Classes}(S) \mid \Sigma_{\mathtt{Id}} \vdash_{\{\mathrm{A1,A2}\}} C_{t_i}(\mathtt{Id} : C_i)\}$ can be computed in $O(K + \|\Sigma_{\mathtt{Id}}\|)$ time by using an algorithm for computing a reflexive transitive closure as discussed in Section 2.3, Part 2 can be executed in $O(K \cdot (K + \|\Sigma_{\mathtt{Id}}\|))$ time, that is, $O(K^3)$ time.

Finally, consider how to execute Part 3. For a pair of $C_a, C_b \in \mathit{Classes}(S)$, let i be the largest integer satisfying

$$\{C_a, C_b\} \subseteq \{C_i \in \mathit{Classes}(S) \mid \Sigma_{\mathtt{Id}} \vdash_{\{\mathrm{A1,A2}\}} C_{t_i}(\mathtt{Id} : C_i)\}. \tag{4.2}$$

We claim that $C_{t_i} = C_a \sqcap C_b$.

Let t_j be an index such that $C_{t_j} = C_a \sqcap C_b$. Since $\Sigma_{\mathtt{Id}} \models C_{t_i}(\mathtt{Id} : C_a)$ and $\Sigma_{\mathtt{Id}} \models C_{t_i}(\mathtt{Id} : C_b)$ by 4.2, it follows from the definition of $C_a \sqcap C_b$ that $\Sigma_{\mathtt{Id}} \models C_{t_i}(\mathtt{Id} : C_{t_j})$. Since $C_{t_1}, C_{t_2}, \cdots, C_{t_K}$ is in a topological order, this implies that $i \leq j$. On the other hand, it must hold that $j \leq i$, since i is the largest integer satisfying 4.2. Thus $i = j$, that is, $C_{t_i} = C_a \sqcap C_b$. This completes proving the claim.

Since a subset of $\mathit{Classes}(S)$ can be represented by a bit array of size K, a set membership can be tested in constant time. Thus it can be decided in constant time whether or not an integer i satisfies 4.2, where $1 \leq i \leq K$. Hence we can find in $O(K)$ time the largest integer i satisfying 4.2, and therefore $C_a \sqcap C_b$. Since the number of entries of the table is $O(K^2)$, we can construct the table in $O(K^3)$ time. That is, Part 3 can be executed in $O(K^3)$ time. Consequently, Parts 1 to 3 can also be executed in $O(K^3)$ time.

[19] In general, given a directed acyclic graph $G(V, A)$, the vertices in V can be sorted in a topological order in linear time.

Example 20: For the database schema $\langle S, \Sigma \rangle$ in Example 18, let us construct the table by the procedure above. In Part 1, the directed graph $G(V, A)$ is constructed as in Figure 20. By using $G(V, A)$, we may have the following sequence (as an example), which is in a topological order:

$$\langle \bot, \mathtt{a}_3, \mathtt{a}_2, \mathtt{c}_2, \mathtt{b}_3, \mathtt{a}_1, \mathtt{b}_1, \mathtt{b}_2, \mathtt{c}_3, \mathtt{c}_1 \rangle$$

Part 2 is easy to execute. In Part 3, consider how to compute, for example, $\mathtt{b}_1 \sqcap \mathtt{b}_2$. It is easy to see that only $\bot$ and $\mathtt{b}_3$ satisfy 4.2 with respect to $\mathtt{b}_1$ and $\mathtt{b}_2$. Since $\mathtt{b}_3$ occurs later than $\bot$ in the sequence above; that is, $\mathtt{b}_3$ has a larger index than $\bot$, it holds that $\mathtt{b}_1 \sqcap \mathtt{b}_2 = \mathtt{b}_3$. □

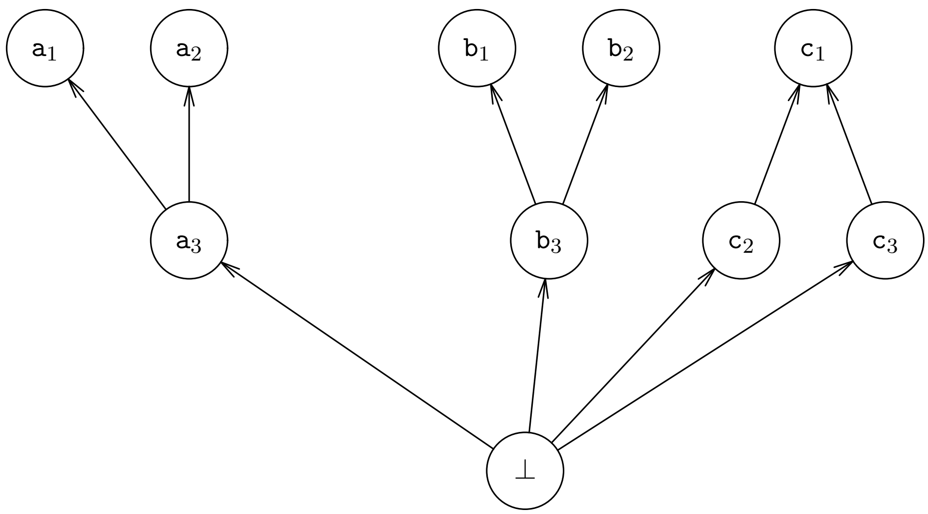

Figure 20: The directed graph $G(V, A)$ in Part 1.

Once the table above is constructed, for any class C and any path description pd, $Chase^{\mathrm{MSC}}_{\Sigma_{\mathtt{SC}}}(C, pd)$ can be computed in $O(D \cdot (len(pd) + 1))$ time by Procedure 10. Thus it is reasonable to exclude the time for constructing the table from the time complexity of Procedure 10.

4.3 An NP-completeness result

By assuming the lower semilattice condition for $\langle S, \Sigma \rangle$, the NP-complete decision problems given in Theorem 6 largely become solvable in polynomial time. Unfortunately, it is still NP-complete to decide whether or not $\Sigma \not\models_{\mathrm{MSC}} C(pd : C')$, if $pd \notin PathDescs_{\mathrm{MSC}}(C)$. (The fact will be proved in this section.) At a glance, one might consider that $\Sigma \models_{\mathrm{MSC}} C(pd : C')$ if and only if

$\Sigma \vdash_{\{A1,A2,A3,A4\}} C(pd : C')$. But this is not the case. The intuitive reason is that even if two *pd*-Lists $v_0 \xrightarrow{P_1} v_1 \xrightarrow{P_2} \cdots \xrightarrow{P_n} v_n$ and $u_0 \xrightarrow{P_1} u_1 \xrightarrow{P_2} \cdots \xrightarrow{P_n} u_n$ satisfy MSC, a *pd*-List $w_0 \xrightarrow{P_1} w_1 \xrightarrow{P_2} \cdots \xrightarrow{P_n} w_n$ does not always satisfy MSC, where $Cl(w_i) = Cl(u_i) \cup Cl(v_i)$ for $0 \leq i \leq n$. Thus we cannot use the same discussion as in proving Theorem 3.

Theorem 10: Assume that $\langle S, \Sigma \rangle$ satisfies the lower semilattice condition. It is NP-complete to decide whether or not $\Sigma \not\models_{\text{MSC}} C(pd : C')$, if $pd \notin PathDescs_{\text{MSC}}(C)$.

Proof. By Lemma 10, the problem is in NP. We prove its NP-hardness. The idea is essentially the same as Lemma 11. We will slightly modify the database schema $\langle S, \Sigma \rangle$ constructed in Lemma 11.

1. As for the definition of S, remove class names Z, B_i for $1 \leq i \leq n$, and property R. The domain of each property is changed as follows:

$$Dom(P_1) = \{ C \} \quad \text{and} \quad Dom(P_i) = \{ X_{i-1}, \overline{X}_{i-1} \} \quad \text{for } 2 \leq i \leq n$$
$$Dom(Q_1) = \{ X_n, \overline{X}_n \} \quad \text{and} \quad Dom(Q_j) = \{ C_{j-1} \} \quad \text{for } 2 \leq j \leq m$$

2. The SC $C(pd : C')$ remains unchanged.
3. As for the definition of Σ, we have only to modify the truth-setting components (3.1.a) and the other SCs (3.1.c) in the fixed part of $\Sigma_{\texttt{SC}}$ as follows:

For each variable x_i, $1 \leq i \leq n$, replace the truth-setting component illustrated in Figure 13 by the one illustrated in Figure 21. That is, the modified truth-setting component consists of the following two SCs:

$$\{X_i(\texttt{Id} : A_i), \overline{X}_i(\texttt{Id} : A_i)\}$$

Clearly, the taxonomy illustrated in Figure 21 satisfies the lower semilattice condition (if the taxonomy is considered as a database schema by itself). Note that the bottom class $\perp$ is explicitly described in Figure 21, though it is unimportant for this proof as in Lemma 11.

Replace the other SCs by the following $2n + 2m - 1$ SCs:

$$\{C(P_1 : A_1)\} \quad \text{and} \quad \{X_{i-1}(P_i : A_i), \overline{X}_{i-1}(P_i : A_i) \mid 2 \leq i \leq n\}$$
$$\{X_n(Q_1 : C_1), \overline{X}_n(Q_1 : C_1)\} \quad \text{and} \quad \{C_{j-1}(Q_j : C_j), T_{j-1}(Q_j : T_j) \mid 2 \leq j \leq m\}$$

Since class name B_i is removed from S, all the SCs containing B_i (such as $\overline{X}_i(\texttt{Id} : B_i)$, $B_i(P_i : B_{i+1})$) are removed from $\Sigma_{\texttt{SC}}$ accordingly. Similarly, the SC $T_m(R : Z)$ is also removed from $\Sigma_{\texttt{SC}}$, since R and Z are removed from S.

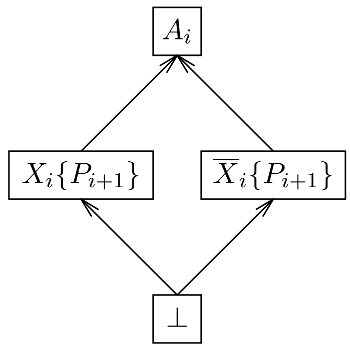

Figure 21: A modified truth-setting component.

The modified fixed part of $\Sigma_{\texttt{SC}}$ is illustrated in Figure 22. The varying part of $\Sigma_{\texttt{SC}}$ remains unchanged. It is easy to verify that the resulting database schema satisfies the lower semilattice condition.

Note that $pd \notin PathDescs_{\mathrm{MSC}}(C)$. In fact, the interpretation $G(V, A)$ for S given in Figure 23 satisfies MSC and $\Sigma \cup S_{\texttt{FUNC}}$ but does not have a path from the vertex u described by pd, where $C \in Cl(u)$. We must prove that $\Sigma \not\models_{\mathrm{MSC}} C(pd : C')$ if and only if E is not a tautology.

Only if part. Assume that $\Sigma \not\models_{\mathrm{MSC}} C(pd : C')$. By Theorem 5, there is a pd-List

$$v_0 \xrightarrow{P_1} v_1 \xrightarrow{P_2} \cdots \xrightarrow{P_n} v_n \xrightarrow{Q_1} u_1 \xrightarrow{Q_2} \cdots \xrightarrow{Q_m} u_m$$

satisfying MSC and PDLs 1 to 3. It is easy to see that Claims 2 and 3 given in the proof of Lemma 11 still hold in this case. As for Claim 1, since the pd-List satisfies PDL 1, it holds that $Cl(v_i) \cap Dom(P_{i+1}) \neq \emptyset$ for $1 \leq i \leq n-1$ and that $Cl(v_n) \cap Dom(Q_1) \neq \emptyset$. Since $Dom(P_{i+1}) = \{ X_i, \overline{X}_i \}$ and $Dom(Q_1) = \{ X_n, \overline{X}_n \}$ by definition, $Cl(v_i)$ must contain either X_i or $\overline{X}_i$ for $1 \leq i \leq n$. That is, Claim 1 still holds. Hence it can be shown that E is not a tautology, along the same line as in the only if part proof of Lemma 11.

If part. Assume that E is not a tautology. There is a truth assignment $\tau : \{x_1, x_2, \cdots, x_n\} \rightarrow \{T, F\}$ that makes E false. We can define the same pd-List

$$v_0 \xrightarrow{P_1} v_1 \xrightarrow{P_2} \cdots \xrightarrow{P_n} v_n \xrightarrow{Q_1} u_1 \xrightarrow{Q_2} \cdots \xrightarrow{Q_m} u_m$$

as in the if part proof of Lemma 11 except 2.1 and 2.2. The definitions of 2.1 and 2.2 are changed as follows:

2.1. If $\tau(x_i) = T$, then $Cl(v_i) = \{A_i, X_i\}$ and $Msc(v_i) = X_i$, where $1 \leq i \leq n$.

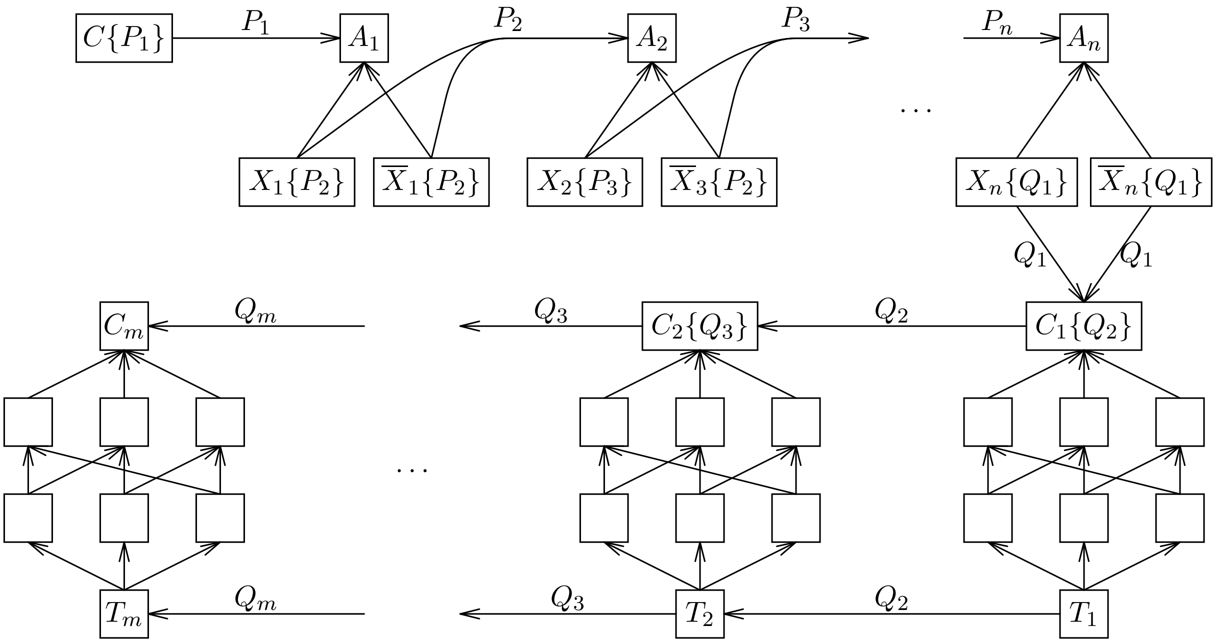

Figure 22: The modified fixed part of $\Sigma_{\texttt{SC}}$.

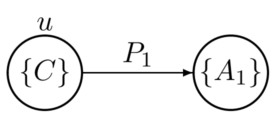

Figure 23: An interpretation $G(V, A)$ for S.

2.2. If $\tau(x_i) = F$, then $Cl(v_i) = \{A_i, \overline{X}_i\}$ and $Msc(v_i) = \overline{X}_i$.

It can be shown that $\Sigma \not\models_{\mathrm{MSC}} C(pd : C')$, along the same line as in the if part proof of Lemma 11. Consequently, Theorem 10 holds. □

5. Conclusion

We have considered a more general form of specialization constraint for data models supporting complex objects and object identity. The generalization is achieved by allowing range restrictions to be associated with descriptions of property value paths, instead of with individual properties. By admitting a path description, called `Id`, for paths of zero length (i.e. consisting of a single object), specialization constraints can also be used to declare subclass relationships. In our introductory comments, we demonstrated how specialization constraints can enable a form of *molecular abstraction*, and indicated how this can be useful not only in modeling, but also in query formulation and physical database design.

In this paper, we have considered the various membership problems for specialization constraints, including the problems of identifying path descriptions corresponding to single or set-valued functions which are total with respect to a given class. We considered these problems for two models. The first imposed no constraints on class membership for objects beyond those implied by subclassing constraints; the second imposed a *most specialized class* (MSC) condition on class membership for objects, which effectively required each object to have originally been created with respect to at most one class. Table 2 summarizes the various complexity results derived in the paper. For each case in which Table 2 indicates that a membership problem can be solved in polynomial time, we have exhibited a polynomial time procedure. Also, sound and complete axiomatizations are given for two cases: arbitrary specialization constraints for the first model; and well-formed specialization constraints for the second model, when problem schema satisfy an additional lower-semilattice condition.

A complete axiomatization for arbitrary schema, assuming the MSC condition, remains an open problem at this time. At the least, this requires a more general form of specialization constraint in which *union extended generalizations* (UXG), as defined in [3], may be used in place of class names. An example of a UXG for the ALGEBRA illustrated in Figure 2 is `sel+proj`, which represents the *union* of the (extensions of) classes `sel` and `proj`. Note that UXG's may also be used to express so-called *cover* constraints, such as `unExp(Id:sel+proj)`, which asserts that any unary expression must be at least one of either a selection or projection. However, note that reasoning about constraints of the form UXG_1`(Id:`UXG_2`)` can be expensive, even *not* assuming MSC. The membership procedure outlined in [3] has $O(n^4)$ time complexity, where n is the description length of the constraints.

References

[1] S. Abiteboul and S. Grumbach. COL: a logic-based langauge for complex objects. In *Proc. 1st International Conference on Extending Database Technology (EDBT)*, pages 271–293, 1988.

[2] S. Abiteboul and P. C. Kanellakis. Object identity as a query language primitive. In *Proc. ACM SIGMOD International Conference on Management of Data*, pages 159–173, June 1989.

[3] H. Arisawa and T. Miura. On the properties of extended inclusion de-

pendencies. In *Proc. 12th International Conference on Very Large Data Bases (VLDB)*, pages 449–456, August 1986.

[4] D. S. Batory and Won Kim. Modeling concepts for vlsi cad objects. *ACM Transactions on Programming Languages and Systems*, 10(1):322–346, January 1989.

[5] C. Beeri. Formal models for object-oriented databases. In *Proc. 1st International Conference on Deductive and Object-Oriented Databases*, pages 370–395, December 1989.

[6] C. Beeri. A formal approach to object-oriented databases. *Data and Knowledge Engineering*, 5:353–382, 1990.

[7] M. A. Casanova, R. Fagin, and C. H. Papadimitriou. Inclusion dependencies and their interaction with functional dependencies. *Journal of Computer and System Sciences*, 28, March 1984.

[8] N. Coburn and G. E. Weddell. Path constraints for graph-based data models: Towards a unified theory of typing constraints, equations and functional dependencies. In *Proc. 2nd International Conference on Deductive and Object-Oriented Databases*, pages 312–331, December 1991.

[9] Computer Corporation of America. *ADAPLEX: Rational and reference manual*, cca-83-08 edition, May 1983.

[10] U. Dayal. Queries and views in an object-oriented data model. In *Proc. 2nd International Workshop on Database Programming Languages*, pages 80–102, June 1989.

[11] G. Di Battista and M. Lenzerini. A deductive method for entity-relationship modeling. In *Proc. 15th International Conference on Very Large Data Bases (VLDB)*, pages 13–21, August 1989.

[12] P. C. Kanellakis, S. S. Cosmadakis, and M. Y. Vardi. Unary inclusion dependencies have polynomial time inference problems. In *Proc. 15th ACM Symposium on Theory of Computing*, pages 264–277, 1983.

[13] C. Lécluse, P. Richard, and F. Velez. o_2: an object-oriented data model. In *Proc. ACM SIGMOD International Conference on Management of Data*, pages 424–433, June 1988.

[14] J. Mylopoulos, P. A. Bernstein, and H. K. T. Wong. A language facility for designing database-intensive applications. *ACM Transactions on Database Systems*, 5(2):185–207, June 1980.

[15] D. W. Shipman. The functional data model and the data language daplex. *ACM Transactions on Database Systems*, 6(1):140–173, March 1981.

[16] G. E. Weddell. Reasoning about functional dependencies generalized for semantic data models. Research Report CS-89-14, Department of Computer Science, University of Waterloo, April 1989.

[17] G. E. Weddell. A theory of functional dependencies for object-oriented data models. In *Proc. 1st International Conference on Deductive and Object-Oriented Databases*, pages 150–169, December 1989.

[18] G. E. Weddell. Reasoning about functional dependencies generalized for semantic data models. *To appear, ACM Transactions on Database Systems*, 1992.

[19] G. E. Weddell and N. Coburn. A theory of specialization constraints for complex objects. In *Proc. 3rd International Conference on Database Theory*, pages 229–244, December 1990.