The Complexity of Multiple Inheritance in Complex Object Data Models

Sonia Bergamaschi
CIOC-CNR
viale Risorgimento 2
I-40136 Bologna
Italy
sonia@deis64.cineca.it

Bernhard Nebel
DFKI GmbH
Stuhlsatzenhausweg 3
W-6600 Saarbrücken
Germany
nebel@dfki.uni-sb.de

Abstract

We specify a data model for complex objects, following closely recent approaches in the areas of object-oriented and deductive database research. The main extensions with respect to previous proposals are, firstly, the *conjunction* operator, which permits one to express *multiple inheritance* between types as a semantic property, and, secondly, a distinction between *primitive* and *derived* classes, the latter corresponding to database views. We sketch how such a data model can be exploited in schema design and specify an algorithm for computing specialization relationships between classes. Using results from formal analyses of knowledge representation languages, we note that the computational properties of interesting problems, such as detection of emptiness of a class interpretation and computation of a specialization ordering, appear to be computationally intractable from a theoretical point of view, but are feasible in almost all practical cases.

1 Introduction

The proposed model incorporates well-known notions such as types, complex values, classes, and objects with identity, in full analogy with recent complex object models proposed in the areas of deductive databases [ACCT90, AG88, AH87, CCCTZ90] and object oriented databases [LRV88, LR89a, LR89b]. The main extensions are, firstly, the *conjunction* operator, which permits one to express *multiple inheritance* as part of a class description. This avoids the specification of explicit redefinitions when multiple inheritance leads to the intersection of embedded types. Secondly, we make a distinction between *primitive* and *derived* classes,[1] where descriptions of derived classes are interpreted as a set of necessary and sufficient conditions on their elements. Thus, derived classes resemble database *views*. Note that the presence of derived classes can lead to situations such that additionally to the explicitly given inheritance relations some implicit specialization relationships are implied by a schema, i.e., all elements of one class are also elements of another class. In the following we call this relationship *subsumption relation*.

Traditionally, class descriptions are interpreted as necessary conditions in database models, which correspond to *primitive* classes in our terminology. A more active role on databases can be performed with derived classes. Given a new class, this class can be automatically *classified* (i.e., its right place in an already existing taxonomy can be found) by determining the set of its most specific subsumers and the set of its most general subsumees. Thus, *minimality* of a schema w.r.t. inheritance can be easily computed and *inconsistency* can be avoided by preventing the introduction of *incoherent*, i.e., necessarily empty classes [BCST88, BS91]. Additionally, *classification* is relevant for *query optimization* [BBMR89, BGN89]. The idea is that a query can be expressed as a class description and by classifying it in the given taxonomy, it is possible to directly access the set of objects satisfying the query. We will not address these issues in this paper, however.

The rest of the paper is organized as follows. In Section 2 we develop the formal specification of the data model, and Section 3 contains the necessary algorithms to solve the *subsumption* and *coherence* problems. Finally, Section 4 contains an analysis of the computational complexity of the problems.

[1] This distinction corresponds to the distinction between primitive and defined concepts in terminological representation languages [BrSc85]

2 A Data Model for Complex Objects

In the following, we will specify our data model. Instead of using a concrete syntax as in the example given above, an abstract syntax is used. However, the reader should have no problems mapping the expressions of the concrete syntax to the abstract syntax.

Basically, we follow [LR89b]. However, we assume a richer structure for the *base type system*, the system of nondecomposable, basic types. Besides the basic types *integer*, *boolean*, *string*, and *mono-valued types*, we consider also the possibility that subsets of the basic types are used, e.g., intervals of integers. The only restriction is that the system of base types is closed w.r.t. intersection and that the intersection of two base types can be computed in polynomial time.

Based on these basic types, *tuple*, *sequence*, *set*, and *class* types can be created. The latter denote sets of *objects with an identity and a value*, while the former three types – also called *value types* denote sets of *complex, finitely nested values* without object identity. Additionally, a conjunction operator can be used to create intersections of previously introduced types. This operator can be used to specify multiple inheritance. For instance, a *research assistant* can be defined as a *student* and a *researcher*.

Finally, types can be given names. For value types, no circular references are permitted in order to guarantee that values are always only finitely nested. Class types, on the other hand, can be defined making circular references. Named class types come in two flavors. A named class type may be *primitive*, which means that the user has to specify the membership of an *object* in the interpretation of a class. Second, a named class can be a *derived class*, in which case the *class interpretation* is computed—corresponding to a database *view*. Since the interpretation of such a class is completely specified by the class description, the place in the specialization hierarchy can be automatically computed.

2.1 Base Type System

Let $\mathcal{D}$ be the countably infinite set of base-values (which will be denoted by $d_1, d_2, \ldots$), e.g., the union of the set of integers, the set of strings, and the booleans. We will not distinguish between base-values and their encoding.

Let $\mathbf{B}$ be a countable set of base-type designators that contains $\mathcal{D}$ (i.e.,

all mono-valued types), and let $\mathcal{I}_{\mathbf{B}}$ be the (fixed) standard interpretation function from $\mathbf{B}$ to $2^{\mathcal{D}}$ such that for all $d \in \mathbf{D}$: $\mathcal{I}_{\mathbf{B}}[d] = \{d\}$. Let "$\sqcap$" be an operation on $\mathbf{B}$ defined by:

$$B' \sqcap B'' = B \text{ iff } \mathcal{I}_{\mathbf{B}}[B'] \cap \mathcal{I}_{\mathbf{B}}[B''] = \mathcal{I}_{\mathbf{B}}[B].$$

We say that $\mathbf{B}$ is a *base-type system* iff $\mathbf{B}$ is complete with respect to $\sqcap$. The special type that has an empty interpretation will be called *empty type* and is denoted by $\bot$.[2] We will say that $\mathbf{B}$ is a *PTIME base-type system* iff $B' \sqcap B'' = B$ can be decided in polynomial time. In the following we assume that a base type system has this property.

Sometimes we will also talk about base-type systems with a particular simple structure, namely, systems such that for each subset $\mathbf{X} \subseteq \mathbf{B}$ with $\bigsqcap \mathbf{X} = B$, there are two elements $B', B'' \in \mathbf{X}$ such that $B' \sqcap B'' = B$. Such base-type systems will be called *binary compact*.

Consider the following set of base-type designators, which we will use in all examples:

$$\mathbf{B} = \{\texttt{Int}, \texttt{String}, \texttt{Bool}, i_1\text{–}j_1, i_2\text{–}j_2, \ldots, d_1, d_2, \ldots\},$$

where the i_k–j_k's denote all possible ranges of integers, and the d_k's denote all the elements of $\texttt{Int} \cup \texttt{String} \cup \texttt{Bool}$. Assuming the standard interpretation of the base-type designators, $\mathbf{B}$ is obviously a binary compact base-type system.

2.2 Types and Values, Classes and Objects

We suppose a countable set $\mathbf{A}$ of *attributes* (denoted by $a_1, a_2, \ldots$) and a countable set $\mathbf{N}$ of type names (denoted by N, N') such that $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{N}$ are pairwise disjoint. Further, $\mathbf{N}$ is partitioned into the sets $\mathbf{C}$, $\mathbf{D}$, and $\mathbf{T}$, where $\mathbf{C}$ consists of names for *primitive class-types* (denoted by $C, C' \ldots$), $\mathbf{D}$ consists of names for *derived class-types* (denoted by $D, D' \ldots$), and $\mathbf{T}$ consists of *value-type names* $(T, T', \ldots)$. Furthermore, $\mathbf{N}$ contains the special symbol $\top_C$ denoting the universal class.

$\mathbf{S}(\mathbf{A}, \mathbf{B}, \mathbf{N})$ denotes the set of all *finite type descriptions* $(S,\ S',\ \ldots)$, also briefly called *types*, over given $\mathbf{A}, \mathbf{B}, \mathbf{N}$, that are built according to the

[2] This type must be part of $\mathbf{B}$ because the conjunction of different mono-valued types is empty.

following abstract syntax rule assuming($a_i \neq a_j$ for $i \neq j$):

$$S \;\rightarrow\; B \mid N \mid \{S\} \mid \langle S \rangle \mid [a_1\colon S_1, \ldots, a_k\colon S_k] \mid S \sqcap S' \mid \triangle S.$$

Given the base-type system introduced above and

$$\begin{aligned} \mathbf{A} &= \{\texttt{street}, \texttt{number}, \texttt{city}, \ldots\} \\ \mathbf{N} &= \{\texttt{Division}, \texttt{Employee}, \texttt{Department}, \texttt{Secretary}, \texttt{Address}, \ldots\} \end{aligned}$$

the following are examples of well-formed type descriptions:

$\texttt{Int}, 5\text{–}10, \texttt{Department}, \{\texttt{Secretary}\}, \langle\texttt{Address}\rangle$
$[\texttt{street}\colon \texttt{String}, \texttt{number}\colon \texttt{Int}, \texttt{city}\colon \texttt{String}]$,
$\texttt{Secretary} \sqcap \texttt{Employee}, \triangle\texttt{Address}$.

We assume a countable set $\mathcal{O}$ of *object identifiers* (denoted by $o, o', \ldots$) disjoint from $\mathcal{D}$ and define the set $\mathcal{V}(\mathcal{O})$ of all *values over* $\mathcal{O}$ (denoted by v, v') as follows (assuming $p \geq 0$ and $a_i \neq a_j$ for $i \neq j$):

$$v \;\rightarrow\; d \mid o \mid \{v_1, \ldots, v_p\} \mid \langle v_1, \ldots, v_p \rangle \mid [a_1\colon v_1, \ldots, a_p\colon v_p],$$

where $[a_1\colon v_1, \ldots, a_i\colon v_i, \ldots, a_p\colon v_p]$ denotes the set of pairs (a_i, v_i). Using the above assumptions and $\mathcal{O} = \{o_1, o_2, \ldots\}$, the following expressions are possible values:

$1, 2,$ *true*, *"Bologna"*, o_{128}, $\{1, 2, \text{"Bologna"}, o_{128}\}$, $\emptyset$, $\langle \mathit{true}, \mathit{false} \rangle$, $\langle\rangle$,
$[\texttt{street}\colon \text{"}\mathit{Stuhlsatzenhausweg}\text{"}, \texttt{number}\colon 3, \texttt{city}\colon \text{"}\mathit{Saarbrücken}\text{"}]$.

Object identifiers are assigned values by a (total) *value function* δ from $\mathcal{O}$ to $\mathcal{V}(\mathcal{O})$. For instance, we may have the following assignments of values to objects:

$$\delta\colon \begin{cases} o_1 & \mapsto & [\texttt{a}\colon \text{"}\mathit{xyz}\text{"}, \texttt{b}\colon 5] \\ o_2 & \mapsto & \langle \mathit{true}, \mathit{false} \rangle \\ & \vdots & \\ o_{128} & \mapsto & \{o_1, o_2\} \\ & \vdots & \end{cases}$$

For a given set of object identifiers $\mathcal{O}$ and a value function δ, the *interpretation function* $\mathcal{I}$ is a function from $\mathbf{N}$ to $2^{\mathcal{V}(\mathcal{O})}$ such that:

$$\begin{aligned}\mathcal{I}[B] &= \mathcal{I}_{\mathbf{B}}[B]\\ \mathcal{I}[C] &\subseteq \mathcal{O}\\ \mathcal{I}[D] &\subseteq \mathcal{O}\\ \mathcal{I}[T] &\subseteq \mathcal{V}(\mathcal{O}) - \mathcal{O}.\end{aligned}$$

The interpretation of types is defined inductively for all $S, S' \in \mathbf{S}(\mathbf{A}, \mathbf{B}, \mathbf{N})$ by:

$$\begin{aligned}\mathcal{I}[\{S\}] &= \Big\{\{v_1, \ldots, v_p\}\Big| v_i \in \mathcal{I}[S], 0 \leq i \leq p\Big\}\\ \mathcal{I}[\langle S\rangle] &= \Big\{\langle v_1, \ldots, v_p\rangle\Big| v_i \in \mathcal{I}[S], 0 \leq i \leq p\Big\}\\ \mathcal{I}\Big[[a_1\colon S_1, \ldots, a_p\colon S_p]\Big] &= \Big\{[a_1\colon v_1, \ldots, a_q\colon v_q]\Big|\ p \leq q, v_i \in \mathcal{I}[S_i], 0 \leq i \leq p,\\ &\qquad\qquad v_j \in \mathcal{V}(\mathcal{O}), p+1 \leq j \leq q\Big\}\\ \mathcal{I}[S \sqcap S'] &= \mathcal{I}[S] \cap \mathcal{I}[S']\\ \mathcal{I}[\triangle S] &= \Big\{o \in \mathcal{O}\Big|\delta(o) \in \mathcal{I}[S]\Big\}\\ \mathcal{I}[\top_C] &= \mathcal{O}.\end{aligned}$$

Note that the interpretation of tuples implies an open world semantics for tuple types similar to [Card84]. For the example above, it follows that

$$\begin{aligned}o_1 &\in \mathcal{I}\Big[\triangle[\mathtt{a}\colon \mathtt{String}]\Big],\\ o_2 &\in \mathcal{I}[\triangle\langle \mathtt{Bool}\rangle],\end{aligned}\qquad\qquad\begin{aligned}o_1 &\in \mathcal{I}\Big[\triangle[\mathtt{a}\colon \mathtt{String}, \mathtt{b} : \mathtt{Int}]\Big],\\ o_{128} &\in \mathcal{I}[\{\triangle\top_C\}].\end{aligned}$$

2.3 Database Schema

Given a base-type system $\mathbf{B}$ and finite sets of attributes $\mathbf{A}$ and names $\mathbf{N} = \mathbf{C} \cup \mathbf{D} \cup \mathbf{T}$, a *schema* σ over $\mathbf{S}(\mathbf{A}, \mathbf{B}, \mathbf{N})$ is a total function from $\mathbf{N}$ to $\mathbf{S}(\mathbf{A}, \mathbf{B}, \mathbf{N})$. We will require some well-formedness conditions on a schema, namely, *well-foundedness of value-type definitions* and of the *inheritance relation*. Let τ be a function from $\mathbf{S}(\mathbf{A}, \mathbf{B}, \mathbf{N})$ to $\mathbf{S}(\mathbf{A}, \mathbf{B}, \mathbf{N})$ such that

$$\begin{aligned}\tau(S) &= S \text{ if } S \in \mathbf{B} \cup \mathbf{C} \cup \mathbf{D} \text{ or } S = \triangle S'\\ \tau(S) &= \sigma(S) \text{ if } S \in \mathbf{T}\end{aligned}$$

$$\begin{aligned}
\tau(\{S\}) &= \{\tau(S)\} \\
\tau(\langle S\rangle) &= \langle\tau(S)\rangle \\
\tau([a_1\colon S_1, \ldots, a_k\colon S_k]) &= [a_1\colon \tau(S_1), \ldots, a_k\colon \tau(S_k)] \\
\tau(S \sqcap S') &= \tau(S) \sqcap \tau(S').
\end{aligned}$$

We say that the schema σ is *type well-founded* iff there is a natural number n such that $\tau^n = \tau^{n+1}$. This condition guarantees that value-types defined using other value-type names describe always finitely nested values.

Similarly to type well-foundedness, we will require that the inheritance relation expressed by conjunctions in a value-type or class-type definition is well-founded. Let ρ be a function from $\mathbf{S}(\mathbf{A}, \mathbf{B}, \mathbf{N})$ to $2^{\mathbf{N}}$ defined as follows:

$$\rho(S) = \begin{cases} \{S\} & \text{if } S \in \mathbf{N} \\ \rho(S') \cup \rho(S'') & \text{if } S = S' \sqcap S'' \\ \{\} & \text{otherwise.} \end{cases}$$

We say N *inherits directly from* N', written $N \prec\!\!\!\mid_\sigma N'$, iff $N' \in \rho(\sigma(N))$. A schema is *inheritance well-founded* iff the transitive closure of $\prec\!\!\!\mid_\sigma$, which is denoted by $\prec_\sigma$, is a strict partial order.[3] The reflexive and transitive closure will be denoted by $\preceq_\sigma$. If a schema is type well-founded and inheritance well-founded we call it *well-formed*.

2.4 Interpretation of a Schema

In the following, we will specify the structures described by our schemata—the legal database states—by defining the notion of *legal instances* of a schema.

We say that an interpretation function $\mathcal{I}$ as defined above is a *possible instance* of a schema σ iff the set $\mathcal{O}$ is *finite*, and for all $C \in \mathbf{C}, D \in \mathbf{D}, T \in \mathbf{T}$:

$$\begin{aligned}
\mathcal{I}[C] &\subseteq \mathcal{I}[\sigma(C)] \\
\mathcal{I}[D] &= \mathcal{I}[\sigma(D)] \\
\mathcal{I}[T] &= \mathcal{I}[\sigma(T)].
\end{aligned}$$

Given a schema σ over $\mathbf{S}(\mathbf{A}, \mathbf{B}, \mathbf{N})$, a set of object identifiers $\mathcal{O}$, a value function δ, and a partial interpretation defined over $\mathbf{C}$, there is at most

[3] Although this restriction is not necessary from a purely technical point of view, it is a restriction which ensures that the intuitive meaning of inheritance is satisfied.

one possible instance provided the schema is cycle-free. In the general case, however, there are many possible instances. Since the interpretation of derived classes shall be uniquely computable for a given assignment of object identifiers to interpretations of primitive classes, we have to single out one particular instance.

The natural candidate for such a "canonical" interpretation would be the smallest possible interpretation (corresponding to the least fixpoint of an operator on interpretations). There are drawbacks of such an approach, however. The least possible instance is often "too small", namely, empty.

Let $\mathbf{P}$ be the *set of possible instances with identical* $\mathcal{O}$ *and* δ such that for all $\mathcal{I}, \mathcal{I}' \in \mathbf{P}$:

$$\mathcal{I}[C] \;=\; \mathcal{I}'[C] \text{ for all } C \in \mathbf{C}.$$

Further, let "$\overset{P}{\trianglelefteq}$" be a relation over $\mathbf{P}$ such that for all $\mathcal{I}, \mathcal{I}' \in \mathbf{P}$:

$$\mathcal{I} \overset{P}{\trianglelefteq} \mathcal{I}' \quad \text{iff} \quad \mathcal{I}[N] \subseteq \mathcal{I}'[N] \text{ for all } N \in \mathbf{N}.$$

Then $(\mathbf{P}, \overset{P}{\trianglelefteq})$ forms a partial ordering. We say that $\mathcal{I}$ is a *legal instance* of a schema σ iff it is the unique *greatest* instance of the set $\mathbf{P}$ w.r.t. $\overset{P}{\trianglelefteq}$.

Theorem 1 *If $\mathcal{I}$ is a possible instance, then there exists a legal instance $\mathcal{I}'$ such that $\mathcal{I} \overset{P}{\trianglelefteq} \mathcal{I}'$.*

2.5 Inheritance, Subsumption, and Coherence

Given a schema σ, there is the question for the semantic relationship between types. For example, we would expect that a named type N that inherits from another named type N', i.e., $N \preceq_\sigma N'$, is always interpreted as a subset of the interpretation of N'. The converse, however, does not hold necessarily. In order to capture this formally, let us define the *subsumption relation*, written $S \sqsubseteq_\sigma S'$ for $S, S' \in \mathbf{S}(\mathbf{A}, \mathbf{B}, \mathbf{N})$ of a schema σ:

$$S \sqsubseteq_\sigma S' \quad \text{iff} \quad \mathcal{I}[S] \subseteq \mathcal{I}[S'] \text{ for all legal instances } \mathcal{I} \text{ of } \sigma.$$

It follows immediately that $\sqsubseteq_\sigma$ is a preorder (i.e., transitive and reflexive) that induces an equivalence relation $\doteq_\sigma$ on types

$$S \doteq_\sigma S' \quad \text{iff} \quad S \sqsubseteq_\sigma S' \text{ and } S' \sqsubseteq_\sigma S$$

Proposition 1 *For a given well-formed schema σ and $N, N' \in \mathbf{N}$:*

$$\textit{If } N \preceq_\sigma N' \textit{ then } N \sqsubseteq_\sigma N'.$$

Thus, the converse does not hold in general. One reason is the presence of derived classes. Another reason is the fact that a type can be equivalent to $\bot$, the empty type. Class types and value types equivalent to $\bot$ will be called *incoherent*. We will require that all named class types and value types in a schema are *coherent* (i.e. $\not\equiv_\sigma \bot$), in which case the schema is called *coherent*. With this definition we can give a partial converse of the above proposition.

Proposition 2 *For a given well-formed and coherent schema σ, for all primitive classes $C, C' \in \mathbf{C}$: $C \sqsubseteq_\sigma C'$ iff $C \preceq_\sigma C'$.*

3 Algorithms

Checking whether a schema is well-formed is quite easy. It amounts to checking for cycle-freeness of the graphs induced by value-type declarations and the inheritance relation. Testing for coherence of a schema and computing subsumption between types is more difficult, however. Instead of specifying algorithms for these problems directly, we will take the detour of defining an algorithm on *canonical extensions* of a given schema. A *canonical schema* ν over $\mathbf{S}(\mathbf{A}, \mathbf{B}, \overline{\mathbf{N}})$, where $\overline{\mathbf{N}} = \overline{\mathbf{C}} \cup \overline{\mathbf{D}} \cup \overline{\mathbf{T}}$, has the following form (for all $C \in \overline{\mathbf{C}}, D \in \overline{\mathbf{D}}, T \in \overline{\mathbf{T}}$):

$$\begin{array}{rcl}
\nu(C) & = & \top_C \\
\nu(D) & = & \bigsqcap C_i \sqcap \triangle N \text{ where } C_i \in \overline{\mathbf{C}}, 0 \leq i, N \in \overline{\mathbf{D}} \cup \overline{\mathbf{T}} \\
\nu(T) & = & \begin{cases}
\bot & \text{or} \\
B & \text{where } B \in \mathbf{B}, \text{ or} \\
\{N\} & \text{where } N \in \overline{\mathbf{N}}, \text{ or} \\
\langle N \rangle & \text{where } N \in \overline{\mathbf{N}}, \text{ or} \\
[a_1 \colon N_1, \ldots, a_p \colon N_p] & \text{where } p \geq 0, N_i \in \overline{\mathbf{N}}, 1 \leq i \leq p.
\end{cases}
\end{array}$$

A schema ν over $\mathbf{S}(\mathbf{A}, \mathbf{B}, \overline{\mathbf{N}})$ is a *conservative extension* of a schema σ over $\mathbf{S}(\mathbf{A}, \mathbf{B}, \mathbf{N})$ iff $\mathbf{N} \subseteq \overline{\mathbf{N}}$ and for any legal instance $\mathcal{I}$ of σ there exists a legal instance $\mathcal{I}'$ of ν and *vice versa* such that

$$\mathcal{I}[N] \;=\; \mathcal{I}'[N] \;\text{ for all } N \in \mathbf{N}.$$

Note that this implies that the subsumption relations on $\mathbf{N}$ are identical. A schema ν is a called *canonical extension* of σ iff ν is canonical and ν is a conservative extension of σ.

Theorem 2 *Any schema σ over $\mathbf{S}(\mathbf{A}, \mathbf{B}, \mathbf{N})$ can be effectively transformed into a schema ν over $\mathbf{S}(\mathbf{A}, \mathbf{B}, \overline{\mathbf{N}})$ that is a canonical extension of σ.*

Using canonical schemata, checking coherence of a schema and computing subsumption is easy. As a matter of fact, both problems can be solved in time polynomial in the size of the canonical schema. For the purpose of coherence checking, let us define the notion of *conditional incoherence* of a type name N in a canonical schema ν based on a set of type names $I \subseteq \overline{\mathbf{N}}$. N is *conditionally incoherent* in ν based on I iff[4]

$$\nu(N) \quad = \quad \begin{cases} \bot & \text{or} \\ [\ldots, a'\colon N', \ldots] & \text{where } N' \in I, \text{ or} \\ \bigsqcap C_i \sqcap \triangle N' & \text{where } N' \in I. \end{cases}$$

Let $I^0 = \emptyset$ and define

$$I^{i+1} = I^i \cup \{N \in \overline{\mathbf{N}} | N \text{ is incoherent in } \nu \text{ based on } I^i.\}$$

Since I^i grows monotonically and $\overline{\mathbf{N}}$ is finite, there exists a natural number n such that $I^n = I^{n+1}$ and we set $\widetilde{I_\nu} = I^n$.

Theorem 3 *If ν is a canonical schema on $\mathbf{S}(\mathbf{A}, \mathbf{B}, \overline{\mathbf{N}})$, then $\nu(N) \doteq_\sigma \bot$ iff $N \in \widetilde{I_\nu}$.*

Corollary 1 *Let σ be schema over $\mathbf{S}(\mathbf{A}, \mathbf{B}, \mathbf{N})$, and let ν be a schema over $\mathbf{S}(\mathbf{A}, \mathbf{B}, \overline{\mathbf{N}})$ that is a canonical extension of σ. Then σ is coherent iff $\widetilde{I_\nu} \cap \mathbf{N} = \emptyset$.*

Turning now to subsumption computation, we define *conditional subsumption* based on a relation $\leq \subseteq \overline{\mathbf{N}} \times \overline{\mathbf{N}}$. We say that S is *conditionally*

[4]Note that $\{\bot\}$ and $\langle\bot\rangle$ are coherent types denoting the empty set and the empty sequence, respectively.

subsumed by S' based on $\leq$, written $S \sqsubseteq_{\leq} S'$ under the following conditions:

$$\begin{array}{rll} B \sqsubseteq_{\leq} B' & \text{iff} & B \sqcap B' = B \\ \{N\} \sqsubseteq_{\leq} \{N\} & \text{iff} & N \leq N' \\ \langle N \rangle \sqsubseteq_{\leq} \langle N' \rangle & \text{iff} & N \leq N' \\ [\ldots, a_i\colon N_i, \ldots] \sqsubseteq_{\leq} [\ldots, a'_j\colon N'_j, \ldots] & \text{iff} & \forall j \exists i\colon (a_i = a'_j \wedge N_i \leq N_j) \\ \bigsqcap_i C_i \sqcap \triangle N \sqsubseteq_{\leq} \bigsqcap_j C'_j \sqcap \triangle N' & \text{iff} & N \leq N' \wedge \forall j \exists i\colon C_i = C'_j. \end{array}$$

Let $\leq^0 = \overline{\mathbf{D}}^2 \cup \overline{\mathbf{T}}^2 \cup \{(C, C) |\, C \in \mathbf{C}\}$ and define $\leq^{i+1}$ in the following way:

$$\leq^{i+1} = \left\{(N, N') \in \overline{\mathbf{N}} \times \overline{\mathbf{N}} \,\middle|\, \nu(N) \sqsubseteq_{\leq^i} \nu(N') \vee N \in \widetilde{I_\nu}\right\}.$$

As above, there exists always a natural number n such that $\leq^n = \leq^{n+1}$ and we set $\lesssim_\nu = \leq^n$.

Theorem 4 *Given a canonical schema ν over $\mathbf{S}(\mathbf{A}, \mathbf{B}, \overline{\mathbf{N}})$, for all $N \in \overline{\mathbf{N}}$:*

$$N \sqsubseteq_\nu N' \quad \textit{iff} \quad N \lesssim_\nu N'.$$

Further, using the property that a canonical extension of a schema is a conservative extension, it follows immediately that subsumption in an arbitrary schema can be computed by computing $\lesssim_\nu$ on its canonical extension.

Corollary 2 *Let σ be schema over $\mathbf{S}(\mathbf{A}, \mathbf{B}, \mathbf{N})$, and let ν be a schema over $\mathbf{S}(\mathbf{A}, \mathbf{B}, \overline{\mathbf{N}})$ that is a canonical extension of σ. Then for all $N, N' \in \mathbf{N}$:*

$$N \sqsubseteq_\sigma N' \quad \textit{iff} \quad N \lesssim_\nu N'.$$

4 Computational Complexity

Although coherence checking and subsumption computation can be easily performed on canonical schemata, there are probably no efficient algorithms for the general case. Considering coherence checking first, it turns out that binary compactness of the base-type system (see Section 2.1) is a crucial condition for efficiency.

Theorem 5 *Checking coherence of a schema can be done in polynomial time provided the base-type system is binary compact, and is NP-hard in the general case.*

The NP-hardness result follows from a reduction of the emptiness of intersection problem for finite state automata, which is NP-hard [GJ79], to coherency of a schema. Subsumption computation is even more difficult. Even a restriction to binary compact base-type systems does not help because the language inclusion problem for finite state automata can be reduced to subsumption (see [Neb90, Baa90, Neb91]).

Theorem 6 *Subsumption is PSPACE-hard.*

Although these results may suggest that subsumption computation may be not feasible in our model, it turns out that the intractability of the problem does not show up very often in practice—an observation also made in the area of knowledge representation [Neb90], where we have to deal with quite similar problems. The reason is probably that schemata are usually formulated in a way such that they are almost canonical, and, as is obvious from the description of the algorithms, coherence checking and subsumption computation on such schemata can be done in polynomial time.

More evidence for the fact that subsumption computation can be done efficiently in most practical cases can be found in related data models. In the data model O_2 [LR89a], for instance, a relation called *refinement* is computed. This relation is identical to our subsumption relation if we assume that there are no primitive classes. Multiple inheritance in O_2 leads always to a request for a "conflict resolution" by the user [LR89a]. Thus, instead of automatically using conjunctions of types—as we do—the user has to provide a type that refines the inherited types. The computation of the refinement relation is then performed on an internal normalized description, where no further inheritance is needed [LR89b]. It is easy to see that this can be done in polynomial time.[5] Since the internal form can be handled efficiently, it must be the case that there exist schemata that would lead to exponentially many conflict-resolution requests. However, such schemata are not very likely to appear in practice (a worst case example is given in [Neb90]).

[5]Ignoring the disjunction operator discussed in that paper, which leads to NP-hardness.

5 Conclusion

We specified a database model for complex objects following recent approaches in the areas of object-oriented and complex object data models. Instead of viewing inheritance as a syntactic transformation, inheritance is expressed by conjunctions of types. Additionally, we introduce the notion of derived classes, which is similar to database views. Since the interpretation of such classes is completely determined by their description, their placement in the specialization hierarchy has to be computed according to the subsumption ordering on classes. As we have sketched, this process can also be exploited in schema design and query optimization.

After specifying an algorithm for computing the subsumption relation, we analyzed the computational complexity of the subsumption and coherence problems. Using recent results from knowledge representation research, we showed that these problems are computationally intractable. The results apply, of course, also to syntactic treatments of inheritance. Since in practical applications this problem does not seem to occur, we conclude that schemata are probably almost always formulated as "almost canonical" schemata, for which subsumption can be computed efficiently.

References

[ACCT90] Atzeni, P., Cacace, F., Ceri, S., and Tanca L. The LOGIDATA$^+$ model. Technical Report 5/20, CNR, Progetto Finalizzato Sistemi Informatica e Calcolo Parallelo, Sottoprogetto 5, 1990.

[AG88] Abiteboul, S. and Grumbach, S. Col: a logic based language for complex objects. In *EDBT'88 International Conference on Extending Database Technology*, pages 271–293, Springer-Verlag, Berlin, 1988.

[AH87] Abiteboul, S. and Hull, R. B. Ifo: a formal semantic data model. *ACM Transactions on Database Systems*, 12, 4 (1987), 297-314.

[AitK86] Aït-Kaci, H. Type Subsumption as a Model of Computation. In Kerschberg, L., editor, *First International Workshop on Expert Database Systems*, Benjamin Cummings, Menlo Park, Cal., 1986.

[Baa90] Baader, F. Terminological cycles in KL-ONE-based KR-languages. In *Proceedings of the 8th National Conference of the American Association for Artificial Intelligence*, Boston, Mass., 1990.

[BBMR89] Borgida, A.,Brachman, R. J., McGuinness, D. L. and Resnick, L. A. CLASSIC: a structural data model for objects, in: *Proceedings of the 1989 ACM SIGMOD International Conference on Mangement of Data*, pages 59–67, Portland, Oreg. (1989). .

[BCST88] Bergamaschi, S., Cavedoni, L. Sartori, C., and Tiberio, P. On taxonomical reasoning in E/R environments. In Batini, C., editor, *Entity Relationship Approach*, pages 443–453, Elsevier, Amsterdam, 1989.

[BS91] Bergamaschi, S., Sartori, C. On taxonomic reasoning in conceptual design, *Transactions on Database Systems*. To appear.

[BGN89] Beck, H. W., Gala, S. K., and Navathe, S. B. Classification as a query processing technique in the CANDIDE semantic data model. In *Proceedings of the International Data Engineering Conference, IEEE*, pages 572–581, Los Angeles, Cal., 1989.

[BrSc85] Brachman, R. J. and Schmolze, J. G. An overview of the KL-ONE knowledge representation system, *Cognitive Science* 9, 2 (1985), 171–216.

[Card84] Cardelli, L. A. Semantics of multiple inheritance. In *Semantics of Data Types*, pages 51–67, Springer-Verlag, Berlin, 1984.

[CCCTZ90] Cacace, F., Ceri, S., Crespi-Reghizzi, S., Tanca, L., and Zicari, R. Integrating object-oriented data modeling with rule-based programming paradigm. In *Symposium on Principles of database Systems, ACM SIGMOD*, pages 225–236.

[GJ79] Garey, M. R. and Johnson, D. S. *Computers and Intractability*, Freeman, San Francisco, Cal., 1979.

[LRV88] Lécluse C., Richard, P., and Velez, F. O_2, an object-oriented data model. In *Proceedings of the 1988 ACM SIGMOD International*

Conference on Mangement of Data, pages 424–433, Chicago, Ill., 1988.

[LR89a] Lécluse, C. and Richard, P. The O_2 database programming language. In *Proceedings of the 15th International Conference on Very Large Data Bases*, pages 411–422, Amsterdam, 1989.

[LR89b] Lécluse, C. and Richard, P. Modeling complex structures in object-oriented databases. In *Proceedings of the 8th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database-Systems*, pages 360–367, March 1989.

[Neb90] Nebel, B. Terminological reasoning is inherently intractable. *Artificial Intelligence*, 43:235–249, 1990.

[Neb91] Nebel, B. Terminological cycles: Semantics and computational properties. In J. Sowa, editor, *Principles of Semantic Networks*. Morgan Kaufmann, San Mateo, Cal., 1991. To appear.