Description Logics with Aggregates and Concrete Domains

Franz Baader, Ulrike Sattler [1]

Institut für Theoretische Informatik, TU Dresden, Germany

Abstract

Description Logics are a family of knowledge representation formalisms well-suited for intensional reasoning about conceptual models of databases/data warehouses. We extend Description Logics with concrete domains (such as integers and rational numbers) that include aggregation functions over these domains (such as min, max, count, and sum) which are usually available in database systems. We show that the presence of aggregation functions may easily lead to undecidability of (intensional) inference problems such as satisfiability and subsumption. However, there are also extensions for which satisfiability and subsumption are decidable, and we present decision procedures for the relevant inference problems.

1 Motivation

Description Logics (DLs) are a family of knowledge representation formalisms designed for the representation of and reasoning about *terminological knowledge* [35,29,4]. In the last years, DLs that have enough expressive power to capture standard formalisms for the conceptual modeling of databases such as entity-relationship diagrams or UML schemas [5,11,12,14,7,8,10,9] were developed. This means that a conceptual model described in one of these formalisms can be translated into a DL knowledge base. Additionally, one can add rather powerful (integrity) constraints to such a knowledge base—a useful feature when, for example, building an integrated schema for a heterogeneous database/data warehouse from the source schemas. In this case, these additional constraints can be used to describe the relationship between the entities/relations in the various source schemas and their relationship to the entities/relations in the integrated schema. Most importantly, one can use

[1] This work was partially supported by the ESPRIT LTR Project "Foundations of Data Warehouse Quality".

the reasoning services of a DL system to check the quality of the conceptual model. For example, one can infer implicit is-a links between entities and relations and detect inconsistent entities or relations. In the case that unintended is-a relations or inconsistencies are detected, one can go back and modify the conceptual model accordingly. If only intended is-a relations and no inconsistencies are detected, the designer of the conceptual model can be sure that no unintended is-a relations are implied by her model and that all entities/relations are consistent because the reasoning services in a DL system such as FaCT [22,32] or Racer [19] are provably correct decision procedures for the corresponding inference problems satisfiability and subsumption. In this and other ways, the reasoning services of the DL system can be used to enhance the quality of the model. Additionally, the inferred is-a relations can be used for semantic query optimisation. For more information on DLs for reasoning about conceptual models, see [10], and for a description of the tool icom, which implements these ideas, see [16].

Aggregation is a useful mechanism available in many expressive representation formalisms such as database schema and query languages. Most database systems provide a fixed set of aggregation functions like sum, min, max, average, and count, which can be used over concrete built-in domains (like the integers or the rational numbers) together with concrete built-in predicates (like comparisons $\leq$, $>$, or comparisons with constants). In the presence of huge amounts of data, summarising this data using aggregation functions plays a central rôle in databases and data warehouses. Hence it is only natural to assume that aggregation should also be present in the conceptual model of an information system in which aggregation is used. However, we are aware only of one extension of entity-relationship diagrams to model "abstract" aggregation, i.e., the aggregation of complex objects from less complex ones [17], but without explicit aggregation functions and built-in predicates.

Since Description Logics have proved to be useful for reasoning about conceptual models, we extend existing DLs with aggregation functions to evaluate the potential of DLs to serve also as a logical basis for conceptual modeling formalisms with aggregation functions and built-in predicates, and to provide the same reasoning services for such an extended modeling formalism as today's DL systems provide for standard ones.

As a basis for our investigation, we take the Description Logic $\mathcal{ALC}$ [35,21,15]. Even though $\mathcal{ALC}$ is rather expressive, it is far less expressive than the DLs used for the encoding of entity-relationship diagrams or UML schemas. However, it turns out that $\mathcal{ALC}$ is nevertheless an interesting starting point for this investigation. In $\mathcal{ALC}$, *concepts* (classes) can be built using Boolean operators, (i.e., *and* ($\sqcap$), *or* ($\sqcup$), and *not* ($\neg$)), and value restrictions on those individuals associated to an individual via a certain role (binary relation). The value

restrictions can be *existential* or *universal*. For example, the concept

$$\texttt{Human} \sqcap \exists \texttt{has\_child}.(\texttt{Human} \sqcap \texttt{Happy})$$

describes those humans having (at least) a happy child, whereas

$$\texttt{Human} \sqcap \forall \texttt{has\_child}.(\texttt{Human} \sqcap \texttt{Happy})$$

describes those humans having only happy children—without requiring that they have children at all.

Most Description Logics are restricted to talking about *abstract* objects (such as objects representing humans, employees, or projects) with *abstract* relations between them (such as "working for" or "being the boss of"). In [2], this restriction was overcome by providing the DL $\mathcal{ALC}$ with an interface to *concrete* domains (such as integers, rational numbers, or strings) and *concrete* relations (such as is-divisible-by, $\leq$, or is-prefix-of). In this extended DL, which is called $\mathcal{ALC}(\mathcal{D})$, abstract individuals can be related to values in a concrete domain $\mathcal{D}$ via *features*, i.e., functional roles. This allows us to describe extravagant managers by the concept

$$\texttt{Manager} \sqcap \forall \texttt{year}.\forall \texttt{month}.<(\texttt{income}, \texttt{expenses}).$$

Each instance of the above concept is a manager who, whenever she is related to a year which is related to a month, then this month is related via the features `income` and `expenses` to the amount of the her income and expenses, and the income is strictly smaller than the expenses.

In [2], it was shown that a rather weak condition (so-called *admissibility*) on the concrete domain suffices to yield decidability of the usual inference problems of this combined logic. Moreover, it was shown in [27] that the complexity of these inference problems scale nicely with the complexity of the concrete domain.

However, looking more closely at the above concept describing extravagant managers, we note that it is too strict. A more reasonable description would take the *annual* income (i.e., the sum over the income of each month) and compare it with the annual expenses. To achieve this expressivity, we view aggregation functions as a means to define new, computed features, like the annual income. In Figure 1, a person, `Josie`, is given who spends, in some months, more money than she earns, and in others less. If we want to know whether she has ever had an extravagant year, we can ask whether `Josie` is an instance of

$$\begin{aligned}\texttt{Human} \sqcap (\exists \texttt{year}.<(\, & \texttt{sum}(\texttt{month} \circ \texttt{income}), \\ & \texttt{sum}(\texttt{month} \circ \texttt{expenses}))),\end{aligned}$$

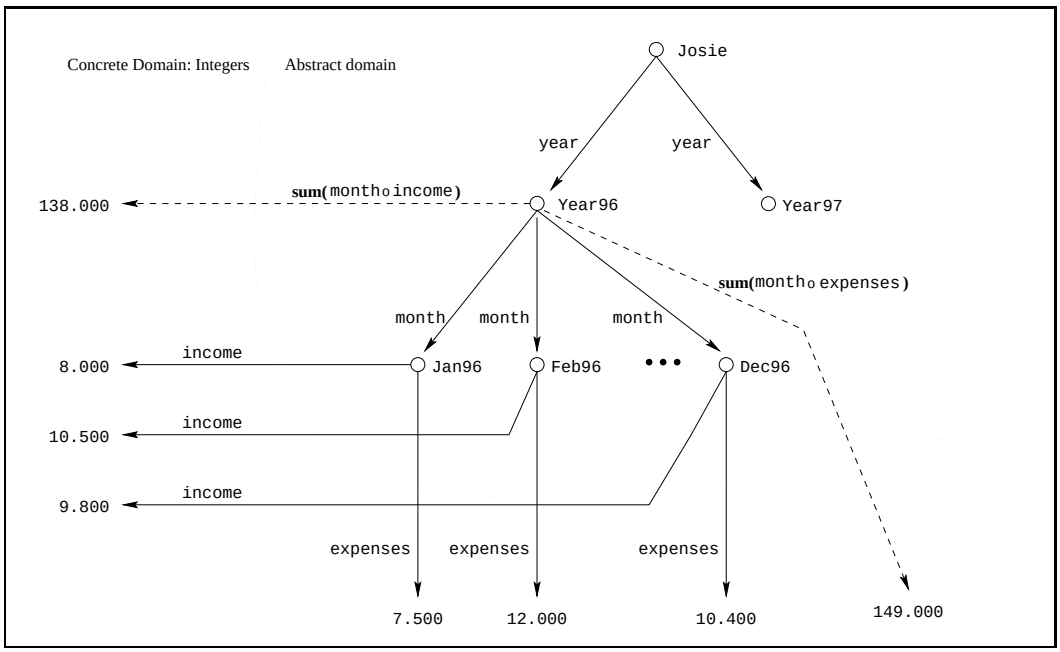

Fig. 1. An example of aggregation.

where the *complex feature* $\mathsf{sum}(\mathtt{month} \circ \mathtt{income})$ relates an individual to the sum over all values reachable over `month` followed by `income`. This new, complex feature is built using the aggregation function `sum`, the role name `month`, and the feature `income`.

In this paper, we present a generic extension of $\mathcal{ALC}(\mathcal{D})$ that is based on this idea of defining new, computed features using aggregation functions. Even though the underlying DL, $\mathcal{ALC}$, is not expressive enough to serve as a logical framework for the above mentioned formalisms for conceptual modeling, it turns out that, given a concrete domain together with aggregation functions satisfying some very weak conditions, satisfiability and subsumption of this extension is undecidable. Moreover, this result is not due to the underlying Description Logic $\mathcal{ALC}$: we show that even for the very weak Description Logic $\mathcal{FL}_0$ (which allows for conjunction and universal value restrictions only), satisfiability and subsumption become undecidable when extended with a few standard aggregation functions.

However, the undecidability proofs reveal that this high complexity is due to the interaction between universal value restrictions and aggregation functions. We describe three ways to regain decidability:

- Firstly, we restrict the underlying Description Logic by disallowing universal value restrictions. We present a tableau-based algorithm that decides satisfiability of this logic, provided that the concrete domain satisfies certain restrictions (see below).
- Secondly, we show that this tableau algorithm can be further extended to decide satisfiability of a restriction of $\mathcal{ALC}(\mathcal{D})$ with aggregation that allows

for (universal and existential) value restrictions and only disallows the interaction between aggregation functions and value restrictions. Since this logic is closed under negation, this tableau algorithm can also be used to decide subsumption between concepts.

Like for $\mathcal{ALC}(\mathcal{D})$, both tableau algorithms depend on the concrete domain, i.e., they require that satisfiability of certain conjunctions of concrete predicates is decidable. For example, the (non-negative) integers or rational numbers with comparisons $>, \geq, \ldots$ possibly involving constants, together with the aggregation functions min, max, and count are concrete domains for which the satisfiability of these conjunctions can be decided.

- Thirdly, we restrict the aggregation functions to contain only min and max. We show that, for standard concrete domains such as integers or rational numbers, together with comparisons and aggregation functions min and max, satisfiability and subsumption of $\mathcal{ALC}(\mathcal{D})$ with aggregation functions is decidable.

The paper is organised as follows: In Section 2, the basic Description Logic $\mathcal{ALC}(\mathcal{D})$ as introduced in [2] is defined. This logic is then extended with aggregation functions in Section 3. Next, to give the reader a better insight into the expressive power added by aggregation functions, we present in Section 4 two generic undecidability results. In Section 5, we present three generic decidability results and, finally, compare these results with similar ones in Section 6.

2 Preliminaries: The Basic Description Logic $\mathcal{ALC}(\mathcal{D})$

In this section, we recall syntax and semantics of $\mathcal{ALC}(\mathcal{D})$, the Description Logic introduced in [2], which underlies the following investigation. $\mathcal{ALC}(\mathcal{D})$ is an extension of the well-known Description Logic $\mathcal{ALC}$ (see [35,21,15]) by so-called *concrete domains*. Firstly, we formally specify a concrete domain.

Definition 1 (Concrete Domains)
A concrete domain $\mathcal{D} = (\mathsf{dom}(\mathcal{D}), \mathsf{pred}(\mathcal{D}))$ *consists of*

- *a set* $\mathsf{dom}(\mathcal{D})$ *(the domain), and*
- *a set of predicate symbols* $\mathsf{pred}(\mathcal{D})$.

Each predicate symbol $P \in \mathsf{pred}(\mathcal{D})$ *is associated with an arity* n *and an* n*-ary relation* $P^{\mathcal{D}} \subseteq \mathsf{dom}(\mathcal{D})^n$.

Secondly, for a given concrete domain $\mathcal{D}$, the syntax of $\mathcal{ALC}(\mathcal{D})$-concepts is defined in [2] as follows:

Definition 2 (Syntax of $\mathcal{ALC}(\mathcal{D})$) *Let N_C, N_R, and N_F be disjoint sets of* concept, role, *and* feature names. *A* feature chain *$u = f_1 \dots f_m$ is a non-empty sequence of features f_i. The set of $\mathcal{ALC}(\mathcal{D})$-*concepts *is the smallest set such that*

(1) every concept name is a concept and
(2) if C, D are concepts, R is a role or a feature name, $P \in \mathsf{pred}(\mathcal{D})$ is an n-ary predicate name, and $u_1, \dots, u_n$ are feature chains, then $(C \sqcap D)$, $(C \sqcup D)$, $(\neg C)$, $(\forall R.C)$, $(\exists R.C)$, and $P(u_1, \dots, u_n)$ are concepts.

In order to fix the exact meaning of these concepts, their semantics is defined in the usual model-theoretic way.

Definition 3 (Semantics of $\mathcal{ALC}(\mathcal{D})$) *An* interpretation *$\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ consists of a finite non-empty set $\Delta^{\mathcal{I}}$ disjoint from $\mathsf{dom}(\mathcal{D})$, called the* domain *of $\mathcal{I}$, and a function $\cdot^{\mathcal{I}}$ which maps*

- *every concept C to a subset $C^{\mathcal{I}}$ of $\Delta^{\mathcal{I}}$,*
- *every role R to a binary relation $R^{\mathcal{I}}$ over $\Delta^{\mathcal{I}}$, and*
- *every feature name $f \in N_F$ to a partial function $f^{\mathcal{I}} : \Delta^{\mathcal{I}} \to \Delta^{\mathcal{I}} \cup \mathsf{dom}(\mathcal{D})$.*

Furthermore, $\mathcal{I}$ has to satisfy the following properties:

$$
\begin{aligned}
(C \sqcap D)^{\mathcal{I}} &= C^{\mathcal{I}} \cap D^{\mathcal{I}},\\
(C \sqcup D)^{\mathcal{I}} &= C^{\mathcal{I}} \cup D^{\mathcal{I}},\\
\neg C^{\mathcal{I}} &= \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}},\\
(\exists R.C)^{\mathcal{I}} &= \{d \in \Delta^{\mathcal{I}} \mid \textit{There exists } e \textit{ with } (d,e) \in R^{\mathcal{I}} \textit{ and } e \in C^{\mathcal{I}}\},\\
(\forall R.C)^{\mathcal{I}} &= \{d \in \Delta^{\mathcal{I}} \mid \textit{For all } e, \textit{ if } (d,e) \in R^{\mathcal{I}}, \textit{ then } e \in C^{\mathcal{I}}\}, \textit{ and}\\
P(u_1, \dots, u_n)^{\mathcal{I}} &= \{d \in \Delta^{\mathcal{I}} \mid (u_1^{\mathcal{I}}(d), \dots, u_n^{\mathcal{I}}(d)) \in P^{\mathcal{D}}\},
\end{aligned}
$$

where, for $u = f_1 \dots f_m$ a feature chain, $u^{\mathcal{I}}(a) = f_m^{\mathcal{I}}(f_{m-1}^{\mathcal{I}}(\dots(f_1^{\mathcal{I}}(a)\dots)$. A concept C is called satisfiable *iff there is some interpretation $\mathcal{I}$ such that $C^{\mathcal{I}} \neq \emptyset$. Such an interpretation is called a* model *of C. A concept D* subsumes *a concept C (written $C \sqsubseteq D$) iff $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ holds for each interpretation $\mathcal{I}$. Two concepts are said to be* equivalent *(written $C \equiv D$) if they mutually subsume each other. For an interpretation $\mathcal{I}$, an individual $a \in \Delta^{\mathcal{I}}$ is called an* instance *of a concept C iff $a \in C^{\mathcal{I}}$. If $f^{\mathcal{I}}(a) = b$ (or $(a,b) \in R^{\mathcal{I}}$), then b is called an* f-successor *(or* R-successor*) of a.*

Please note that, in contrast to the semantics defined in [2], we restrict our attention to *finite* interpretations, i.e., those with a finite domain. For $\mathcal{ALC}(\mathcal{D})$, this does not make a difference since, as a corollary of the results in [2], $\mathcal{ALC}(\mathcal{D})$ has the finite model property. That is, each satisfiable concept has

a finite model. However, in the presence of aggregation functions, this will make a difference since adding aggregation functions makes $\mathcal{ALC}(\mathcal{D})$ lose the finite model property. That is, there are satisfiable concepts that have infinite models only. Since our investigation is motivated by the above mentioned database applications and databases are, in general, *finite* structures, it is indeed necessary to restrict our attention to *finite* models. For the same reason, finite model reasoning in Description Logics has, e.g., been considered in [6].

In [2], subsumption and satisfiability are proved to be decidable for $\mathcal{ALC}(\mathcal{D})$-concepts, provided that the concrete domain is *admissible*. A tableau-based decision procedure for these and other inference problems is presented. We recall the definition of admissibility:

Definition 4 Admissibility *A concrete domain $\mathcal{D}$ is called* admissible *iff*

(1) $\mathsf{pred}(\mathcal{D})$ *is closed under negation, i.e.,* $\mathsf{pred}(\mathcal{D})$ *contains, for each n-ary predicate symbol P in* $\mathsf{pred}(\mathcal{D})$*, an n-ary predicate symbol $\bar{P}$ with $\bar{P}^{\mathcal{D}} = \mathsf{dom}(\mathcal{D})^n \setminus P^{\mathcal{D}}$,*

(2) $\mathsf{pred}(\mathcal{D})$ *contains a unary predicate name $\top_{\mathcal{D}}$ for* $\mathsf{dom}(\mathcal{D})$*, and*

(3) satisfiability of finite conjunctions over $\mathsf{pred}(\mathcal{D})$ *is decidable, i.e., satisfiability of formulae of the form*

$$P_1(x_1^{(1)}, \ldots, x_{n_1}^{(1)}) \wedge \ldots \wedge P_1(x_1^{(k)}, \ldots, x_{n_k}^{(k)})$$

is decidable, where P_i are predicate names of arity n_i.

Moreover, the authors show how two disjoint concrete domains $\mathcal{D}_1$ and $\mathcal{D}_2$ (e.g., the integers and strings) can be combined into a single, new concrete domain $\mathcal{D}_{1,2}$. If $\mathcal{D}_1$ and $\mathcal{D}_2$ are admissible, then $\mathcal{D}_{1,2}$ is also admissible. Due to this observation, we will restrict our attention to extensions of $\mathcal{ALC}$ with single concrete domains.

As a consequence of Definition 3, an instance of a concept $P(u_1, \ldots, u_n)$ has necessarily a u_i-successor that is in $\mathsf{dom}(\mathcal{D})$ for each $1 \leq i \leq n$. Thus, to ensure that, for a concrete feature f, an individual has an f-successor in $\mathsf{dom}(\Sigma)$, we can make use of a predicate restriction $\top_\Sigma(f)$ if the predicate $\top_\Sigma$ is available. Otherwise, we can make use, for example, of the equality $P_=(f, f)$. To express that an individual has no f-successor at all, we will use the abbreviation $\mathsf{no}_f = \forall f.(A \sqcap \neg A)$.

As $\mathcal{ALC}(\mathcal{D})$ allows for negation and conjunction of concepts, all Boolean operators can be expressed, and we will use $C \Rightarrow D$ as a shorthand for $\neg C \sqcup D$. Another consequence of the presence of these two operators is that subsumption and (un)satisfiability can be reduced to each other:

- $C \sqsubseteq D$ iff $C \sqcap \neg D$ is unsatisfiable, and

- C is unsatisfiable iff $C \sqsubseteq A \sqcap \neg A$ (for some concept name A).

3 Extension of $\mathcal{ALC}(\mathcal{D})$ With Aggregation

In order to define aggregation appropriately, first, we will introduce the notion of *multisets*: in contrast to simple sets, an individual can occur more than once in a multiset—but only finitely often. For example, the multiset $\{\{1\}\}$ is different from the multiset $\{\{1, 1\}\}$. Multisets are needed to make sure, for example, that one's annual income is calculated correctly from one's monthly income in the case that the same amount is earned in several months.

Definition 5 (Multisets) *A* multiset *M over S is a mapping $M : S \to \mathbb{N}$, where $M(s)$ denotes the number of occurrences of s in M. A multiset M over S is said to be* finite *iff $\{s \mid M(s) \neq 0\}$ is a finite set. The set of all finite multisets of S is denoted $\mathsf{MS}(S)$. We use the notation $\{\{a_1, \ldots, a_n\}\}$ when enumerating the members a_i of a finite multiset to distinguish multisets from sets.*

For multisets M, M' over S, we write $M \subseteq M'$ if $M(s) \leq M'(s)$ for each $s \in S$, and we write $s \in M$ if $M(s) \geq 1$. For $M \subseteq M'$, we use $M' \setminus M$ to denote the multiset with $(M' \setminus M)(s) := M'(s) - M(s)$ for all $s \in S$.

Since the aggregation functions strongly depend on the specific concrete domains, the notion of a *concrete domain* is extended accordingly. Furthermore, the notion of *concrete features* is introduced. Such a concrete feature is either a feature name, a feature chain, or built using an aggregation function on a role and a feature name.

Definition 6 (Syntax of $\mathcal{ALC}(\Sigma)$) *The notion of a concrete domain $\mathcal{D}$ as introduced in Definition 1 is extended with a set of aggregation functions $\mathsf{agg}(\mathcal{D})$, where each $\Gamma \in \mathsf{agg}(\mathcal{D})$ is associated with a partial function $\Gamma^{\mathcal{D}}$ from the set of finite multisets of $\mathsf{dom}(\mathcal{D})$ into $\mathsf{dom}(\mathcal{D})$. To underline the fact that a concrete domain provides aggregation functions, it is denoted Σ.*

The set of concrete features *is defined as follows:*

- *Each feature name $f \in N_F$ is a concrete feature,*
- *a feature chain $f_1 \ldots f_n$ is a concrete feature, and*
- *an aggregated feature $f_1 \ldots f_n \, \Gamma(R \circ f)$ is a concrete feature, where $f, f_1, \ldots, f_n$ are feature names, R is a role name, and $\Gamma \in \mathsf{agg}(\Sigma)$ is an aggregation function.*

Finally, $\mathcal{ALC}(\Sigma)$-concepts are obtained from $\mathcal{ALC}(\mathcal{D})$-concepts by allowing, additionally, the use of concrete features f_i in predicate restrictions $P(f_1, \ldots, f_n)$

(recall that in $\mathcal{ALC}(\mathcal{D})$ only feature chains were allowed).

It remains to extend the semantics of $\mathcal{ALC}(\Sigma)$ to the new feature-forming operator:

Definition 7 (Semantics of $\mathcal{ALC}(\Sigma)$) *An $\mathcal{ALC}(\Sigma)$-interpretation $\mathcal{I}$ is an $\mathcal{ALC}(\mathcal{D})$-interpretation that, additionally, interprets aggregated features as follows. To define the semantics of aggregated features, we introduce the multiset $M_a^{R\circ f}$ which maps each element $z \in \mathsf{dom}(\Sigma)$ to the number of a's R-successors that have z as f-successor:*

$$M_a^{R\circ f}(z) := \#\{b \in \Delta^{\mathcal{I}} \mid (a,b) \in R^{\mathcal{I}} \text{ and } f^{\mathcal{I}}(b) = z\}.$$

Finally, the semantics of aggregated features is defined as follows:

$$(f_1 \ldots f_n\, \Gamma(R \circ f))^{\mathcal{I}}(a) := \begin{cases} \Gamma^{\Sigma}(M_{a'}^{R\circ f}) & \text{if } (f_1 \ldots f_n)^{\mathcal{I}}(a) = a' \in \Delta^{\mathcal{I}} \\ \text{undefined} & \text{if } (f_1 \ldots f_n)^{\mathcal{I}}(a) \notin \Delta^{\mathcal{I}} \end{cases}$$

and $\Gamma^{\Sigma}(M_{a'}^{R\circ f})$ is called the $(f_1 \ldots f_n\, \Gamma(R \circ f))$-successor of a, provided that it is defined.

We point out two consequences of this definition, which might not be obvious at first sight:

(a) If a has an R-successor b with an abstract f-successor, then b has no influence on $M_a^{R\circ f}$: it is defined in such a way that it takes only into account $R \circ f$-successors of a in the concrete domain $\mathsf{dom}(\Sigma)$.

(b) Since $\Delta^{\mathcal{I}}$ is finite, each $M_{a'}^{R\circ f}$ is necessarily a finite multiset. However, there are two reasons why $(f_1 \ldots f_n\, \Gamma(R \circ f))^{\mathcal{I}}(a)$ might not be defined: firstly, a might have no $f_1 \ldots f_n$-successor a' in $\Delta^{\mathcal{I}}$. Secondly, aggregation functions can be partial. For example, the (standard) min or max over an empty set is undefined. Hence if $\mathsf{dom}(\Sigma)$ is the set of rational numbers, integers, etc., and if a has no R-successor in $\mathcal{I}$ with an f-successor in the concrete domain, then $M_a^{R\circ f}$ is the empty multiset, and thus $(\mathsf{max}(R \circ f))^{\mathcal{I}}(a)$ is undefined.

In the following, we will make use of the aggregation functions count, sum, min, and max, which are supposed to be defined as usual, i.e., for finite multisets

M over the rational numbers (or any subset of the rational numbers) we have

$$\mathsf{count}(M) = \sum_{y \in M} M(y)$$

$$\mathsf{sum}(M) = \sum_{y \in M} M(y) \cdot y$$

$$\mathsf{min}(M) = \begin{cases} m & \text{if there exists } m \in M \text{ such that } n \geq m \text{ for all } n \in M \\ \text{undefined} & \text{if no such } m \text{ exists} \end{cases}$$

$$\mathsf{max}(M) = \begin{cases} m & \text{if there exists } m \in M \text{ such that } n \leq m \text{ for all } n \in M \\ \text{undefined} & \text{if no such } m \text{ exists} \end{cases}$$

4 Undecidability Results

In this section, the expressive power added to $\mathcal{ALC}(\mathcal{D})$ by aggregation functions is illustrated. It turns out that, for a concrete domain Σ satisfying some rather weak conditions, reasoning in $\mathcal{ALC}(\Sigma)$ and its restriction $\mathcal{FL}_0(\Sigma)$ become undecidable in the presence of standard aggregation functions like min, max, and sum.

4.1 A first undecidability result

The following theorem states that admissibility of a concrete domain does no longer guarantee decidability of the interesting inference problems:

Theorem 8 *For a concrete domain Σ where*

- $\mathsf{dom}(\Sigma)$ *includes the non-negative integers,*
- $\mathsf{pred}(\Sigma)$ *contains a (unary) predicate $P_{=1}$ that tests for equality with* 1*, and a (binary) equality $P_{=}$,*
- $\mathsf{agg}(\Sigma)$ *contains* min*,* max*, and* sum*,*

satisfiability and subsumption of $\mathcal{ALC}(\Sigma)$-concepts are undecidable.

Remark 9 (a) At first sight, this undecidability result may appear to be very restricted. Note, however, that it does not require that $\mathsf{dom}(\Sigma)$ is the set of non-negative integers, but that it just requires that $\mathsf{dom}(\Sigma)$ *contains* the non-negative integers. This makes the undecidability result not only more general,

but also stronger. For example, computations over the real numbers are, in general, easier than computations over the non-negative integers; e.g., the first order theory of $+, \cdot, \leq$ is undecidable over the non-negative integers, whereas it is decidable over the real numbers.

Furthermore, the aggregation functions min, max, and sum are among those normally considered as built-in functions for databases (see, for example, [18,28,26,36]). Finally, to test whether a certain value equals 1 or whether two values are equal is possible in all database systems with built-in predicates.

(b) We do not suppose that Σ is admissible—although this precondition would not weaken the undecidability result. Nevertheless, in the sequel, we will make use of the concept $\top_\Sigma(f)$ describing all those (abstract) individuals having an f-successor in the concrete domain. This is in accordance with the preconditions of Theorem 8 because $\top_\Sigma(f)$ (if not available in Σ) can be introduced as abbreviation, e.g., for $P_=(f, f)$.

(c) Undecidability is not due to the fact that we require $\Delta^\mathcal{I}$ to be finite. The proof works analogously for infinite interpretations (where $M_a^{R \circ f}$ is defined appropriately in the case that a has infinitely many R-successors); see [3].

Proof of Theorem 8: The proof is by reduction of Hilbert's 10th problem [13] to satisfiability of concepts, i.e., for polynomials $P, Q \in \mathbb{N}[x_1, \ldots, x_m]$, we construct an $\mathcal{ALC}(\Sigma)$-concept $C_{P,Q}$ that is satisfiable iff the polynomial equation

$$P(x_1, \ldots, x_m) = Q(x_1, \ldots, x_m) \tag{1}$$

has a solution in $\mathbb{N}^m$. In the sequel, we write $\mathbf{x}$ as shorthand for $(x_1, \ldots, x_m)$ and $\mathbf{x}^{\mathbf{i}_j}$ as shorthand for the monomial $x_1^{i_{j1}} \cdots x_m^{i_{jm}}$.

The idea of the reduction is to represent the (sub)term structure of the polynomial P (Q) as a tree related to an instance of $C_{P,Q}$ via the feature P (Q). Each polynomial is supposed to be of the form

$$a_0 + a_1 \mathbf{x}^{\mathbf{i}_1} + \ldots + a_j \mathbf{x}^{\mathbf{i}_j} + \ldots a_n \mathbf{x}^{\mathbf{i}_n},$$

where, for simplicity, all monomials $\mathbf{x}^{\mathbf{i}_j}$ are assumed to be different.

When building the reduction concept $C_{P,Q}$, one encounters three main problems:

(a) We only know that $\mathsf{dom}(\Sigma)$ *contains* $\mathbb{N}$, but the solution of Equation 1 must be in $\mathbb{N}^m$, and Σ need not provide a predicate that tests for being a non-negative integer.
(b) It has to be guaranteed that (the representation of) each variable x_i is associated with the same non-negative integer wherever it occurs in a model

of $C_{P,Q}$.

(c) The reduction asks for the representation of calculations such as addition, multiplication, and exponentiation.

These problems can be overcome as follows:

(a) is solved by making use of the concept E_g^R,

$$E_g^R := (\forall R.P_{=1}(f)) \sqcap P_{=}(\mathsf{sum}(R \circ f), g),$$

whose instances have as g-successors the number of their R-successors. Hence their g-successor is defined and in $\mathbb{N}$.

(b) This problem is solved by introducing features $\mathtt{x}_i$ for each variable x_i and by making strong use of the concepts $E_{\mathtt{x}_i}^R$ defined above (to make sure that $\mathtt{x}_i$-successors are non-negative integers) and the following concept Inv:

$$\mathsf{Inv} := \bigsqcap_{1 \le i \le m} (\forall R.\top_\Sigma(\mathtt{x}_i) \sqcap P_{=}(\mathsf{min}(R \circ \mathtt{x}_i), \mathsf{max}(R \circ \mathtt{x}_i)) \sqcap P_{=}(\mathtt{x}_i, \mathsf{max}(R \circ \mathtt{x}_i))).$$

Let a be an instance of Inv. Then the first conjunct ensures that all R-successors of a have an $\mathtt{x}_i$-successor in $\mathsf{dom}(\Sigma)$. The second conjunct ensures that all $R \circ \mathtt{x}_i$-successors of a coincide and, finally, the third conjunct ensures that a's $\mathtt{x}_i$-successor coincides with the $\mathtt{x}_i$-successors of its R-successors.

Using Inv at all levels of nested concepts, we can guarantee that all "relevant" individuals in a model of $C_{P,Q}$ have the same $\mathtt{x}_i$-successor for each variable x_i.

(c) Addition can be realised by the aggregation function sum, and multiplication (and hence exponentiation) can be reduced to addition; for details see the explanation of the reduction concepts below.

For the representation of constants (like the coefficients) we will use the following abbreviations:

$$E_1^R := (\forall R.(P_{=1}(f))) \sqcap P_{=1}(\mathsf{sum}(R \circ f)) \quad \text{(exactly 1 } R\text{-successor)}$$

$$E_n^R := \forall R.\left(\bigsqcup_{1 \le i \le n} (P_{=1}(f_i) \sqcap \bigsqcap_{j \ne i} \mathsf{no}_{f_j})\right) \sqcap \quad \text{(exactly } n \text{ } R\text{-successors)}$$
$$\bigsqcap_{1 \le i \le n} P_{=1}(\mathsf{sum}(R \circ f_i))$$

where no_{f_j} is the abbreviation for $\forall f_j.(A \sqcap \neg A)$ mentioned in Section 2. It is easy to see that each instance of E_1^R has exactly 1 R-successor. Now, for an instance a of E_n^R, every R-successor has exactly one f_i-successor for some $i, 1 \le i \le n$, and this f_i-successor has value 1 (first line). The constraint on the concrete feature $\mathsf{sum}(R \circ f_i)$ (second line) makes sure that, for each i, there is

exactly one R-successor with an f_i-successor, which implies that a has exactly n R-successors. For those familiar with Description Logics, we point out that E_n^R is indeed equivalent to the number restriction $(= n\, R)$.

Summing up, for Σ as described in Theorem 8, we have defined the following abbreviations:

no_f	describes individuals with no f-successor
E_1^R	describes individuals with exactly 1 R-successor
E_n^R	describes individuals with exactly n R-successors
E_g^R	describes individuals a with exactly $g^{\mathcal{I}}(a)$ R-successors
Inv	describes individuals a whose $\mathtt{x}_i$-successor coincides with the $\mathtt{x}_i$-successor of each of its R-successors

The definition of the reduction concept $C_{P,Q}$ and the auxiliary concepts used in this definition can be found in Figures 3 and 4. Figure 2 sketches a model of $C_{P,Q}$. Let us now explain the definition of $C_{P,Q}$:

(1) First, we define $C_{P,Q}$ such that, for each interpretation $\mathcal{I}$, each instance $a \in C_{P,Q}^{\mathcal{I}}$ has exactly one P-successor p in $C_P^{\mathcal{I}}$ and exactly one Q-successor q in $C_Q^{\mathcal{I}}$. The individual p represents the polynomial P, and q represents Q; see Concept 2. Concept 3 is similar to Inv and makes sure that, for each j, the $\mathtt{x}_j$-successor of p is in $\mathsf{dom}(\Sigma)$ and the same as the $\mathtt{x}_j$-successor of q. Using the feature s to store the value of the evaluation of the polynomials, Concept 4 makes sure that the value of the polynomial P when evaluated with the $\mathtt{x}_j$-successors (which are already ensured to be the same for p and for q) is the same as of Q.

(2) An instance p of C_P has
- for each summand $A_j = a_j \mathbf{x}^{\mathbf{i}_j}$ of P one R-successor, which is an instance of C_{A_j}; see the first two conjuncts of Concept 5. The use of the concepts E_j^H ensures that all C_{A_j} are disjoint, and thus to ensure that each summand is represented by a different R-successor.
- an s-successor, which is the sum of the s-successors of its R-successors; see the last conjunct of Concept 5.

Given that the s-successor of each R-successor of p is the value of the jth summand, the s-successor of p is the corresponding value of P, namely the sum over P's summands. Again, the concept Inv makes sure that each $\mathtt{x}_i$-successor of p coincides with the $\mathtt{x}_i$-successors of its R-successors, and thus the summands are evaluated by the same tuple.

(3) The concept C_Q is defined analogously.

(4) For each summand $A_j = a_j \mathbf{x}^{\mathbf{i}_j}$, we use a concept C_{A_j}. An instance a of C_{A_j} has a_j R-successors, each of them representing the monomial $\mathbf{x}^{\mathbf{i}_j}$; see

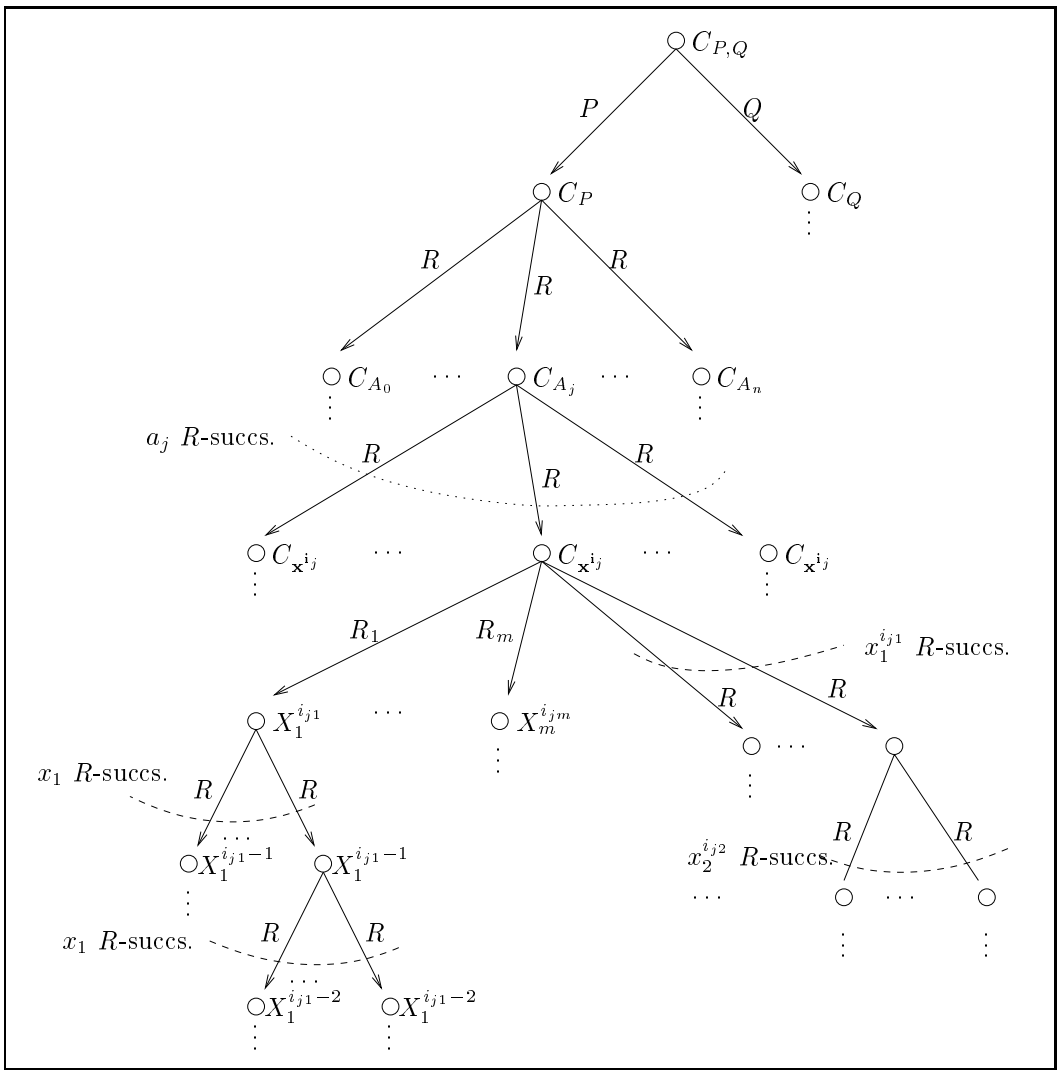

Fig. 2. The intuitive structure of a model of $C_{P,Q}$.

Concept 7. The last conjunct makes sure that the s-successor (representing the value of A_j) is computed correctly: since a has a_j R-successors, each of them representing $\mathbf{x}^{\mathbf{i}}{}_j$, the s-successor of a is the sum over the s-successors of its R-successors, namely a_j times $\mathbf{x}^{\mathbf{i}}{}_j$.

(5) $C_{\mathbf{x}^{\mathbf{i}}j}$ is more complicated. An instance c of it has two different kinds of role successors:

- For each of the m factors x_k^{ijk} in $\mathbf{x}^{\mathbf{i}}{}_j$, c has one R_k-successor in X_k^{ijk}, whose s_k-successor stands for its value x_k^{ijk}. The concept $\mathsf{Exp}_{\mathbf{x}^{\mathbf{i}}j}$ guarantees this fact. In $\mathsf{Exp}_{\mathbf{x}^{\mathbf{i}}j}$, we use the second conjunct instead of Inv to propagate the value of x_k down to the according subtree. The last conjunct of $\mathsf{Exp}_{\mathbf{x}^{\mathbf{i}}j}$ makes sure that the respective values s_k are propagated upwards to c.
- Then, in order to multiply the m factors x_k^{ijk}, we make use of the concept Mult_1^m explained below. Again, the s-successor of c denotes the value of this calculation, namely $\mathbf{x}^{\mathbf{i}}{}_j$.

$$C_{P,Q} := E_1^P \sqcap E_1^Q \sqcap \forall P.C_P \sqcap \forall Q.C_Q \sqcap \tag{2}$$

$$\bigsqcap_{1\le j\le m}\Big(P_{=}(\mathsf{sum}(P\circ \mathtt{x}_j),\mathsf{sum}(Q\circ \mathtt{x}_j))\Big)\sqcap \tag{3}$$

$$P_{=}(\mathsf{sum}(P\circ s),\mathsf{sum}(Q\circ s)) \tag{4}$$

$$C_P := E_{n+1}^R \sqcap \bigsqcap_{0\le j\le n} \exists R.(C_{A_j}\sqcap E_j^H)\sqcap \mathsf{Inv}\sqcap P_{=}(s,\mathsf{sum}(R\circ s)) \tag{5}$$

$$C_{A_j} := E_{a_j}^R \sqcap \forall R.C_{\mathbf{x}^{\mathtt{i}}j}\sqcap \mathsf{Inv}\sqcap P_{=}(s,\mathsf{sum}(R\circ s)) \tag{6}$$

$$C_{\mathbf{x}^{\mathtt{i}}j} := \mathsf{Exp}_{\mathbf{x}^{\mathtt{i}}j}\sqcap \mathsf{Mult}_1^m \tag{7}$$

Fig. 3. The reduction concept $C_{P,Q}$ and some of its subconcepts.

(6) For X_k^i, we have to distinguish two cases : If $i > 0$, an instance b of X_k^i is the root of an x_k-ary R-tree of depth i where the s-successor of each node is the sum of the s-successors of its R-successors. Finally, the s-successor of a node one level above the leaves (which represents x_k^1) equals its $\mathtt{x}_k$-successor—which is the same for all nodes in the whole tree. Since $\mathsf{dom}(\Sigma)$ is only required to contain the non-negative integers, we have to ensure that all $\mathtt{x}_k$-successors are non-negative integers. This is realised by making use of the concept $E_{\mathtt{x}_k}^R$.

Otherwise, $i = 0$, and the value associated to this factor is 1; see the concept X_k^0.

Thus, we use the possibilities to construct trees and to sum up in order to compute exponentiation.

(7) Finally, the situation in which we start multiplication looks as follows: An instance u of Mult_1^m is the root of the multiplication tree, u is also an instance of $C_{\mathbf{x}^{\mathtt{i}}j}$, and we want to multiply the s_k-successors $x_k^{i_{jk}}$ $(k = 1,\ldots,m)$ of u. To this purpose, we attach an additional R-tree of depth $m-1$ to u. This tree is, at level k, of outdegree $x_k^{i_{jk}}$, which is the value of s_k of at the node u, and its s-successor of nodes on level $k-1$ represents $x_k^{i_{jk}}\cdot\ldots\cdot x_{m-1}^{i_{jm-1}}x_m^{i_{jm}}$. At level $m-1$, we make sure that the s_m-successors coincide with the s-successor. Again, we sum up the values from the bottom to the top by using the concept $P_{=}(s,\mathsf{sum}(R\circ s))$, and we make sure that all nodes have the same s_i successor by a concept similar to Inv; see Concept 13.

It remains to be shown that $C_{P,Q}$ is satisfiable iff $P(\mathbf{x}) = Q(\mathbf{x})$ admits a solution in the non-negative integers.

$$\mathsf{Exp}_{\mathbf{x}^{\mathbf{i}}j} := \bigsqcap_{1\le k\le m}\Big(E_1^{R_k} \sqcap P_{=}(\mathtt{x}_k, \mathsf{sum}(R_k \circ \mathtt{x}_k)) \sqcap \tag{8}$$

$$\forall R_k.X_k^{i_{jk}} \sqcap P_{=}(s_k, \mathsf{sum}(R_k \circ s_k))\Big) \tag{9}$$

$$\mathsf{Mult}_m^m := P_{=}(s, s_m) \tag{10}$$

$$\text{for } 1 \le k < m: \tag{11}$$

$$\mathsf{Mult}_k^m := E_{s_k}^R \sqcap P_{=}(s, \mathsf{sum}(R \circ s)) \sqcap \forall R.\mathsf{Mult}_{k+1}^m \sqcap \tag{12}$$

$$\bigsqcap_{\ell=k+1}^{m}\Big(P_{=}(\mathsf{min}(R \circ s_\ell), \mathsf{max}(R \circ s_\ell)) \sqcap P_{=}(\mathsf{min}(R \circ s_\ell), s_\ell)\Big) \tag{13}$$

$$X_k^0 := P_{=1}(s) \tag{14}$$

$$X_k^1 := E_{\mathtt{x}_k}^R \sqcap P_{=}(s, \mathtt{x}_k) \tag{15}$$

$$X_k^\ell := E_{\mathtt{x}_k}^R \sqcap \forall R.X_k^{\ell-1} \sqcap P_{=}(s, \mathsf{sum}(R \circ s)) \sqcap \tag{16}$$
$$P_{=}(\mathsf{min}(R \circ \mathtt{x}_k), \mathsf{max}(R \circ \mathtt{x}_k)) \sqcap P_{=}(\mathtt{x}_k, \mathsf{max}(R \circ \mathtt{x}_k)), \quad \ell \ge 2$$

Fig. 4. Subconcepts of $C_{P,Q}$ used for the representation of calculations.

"$\Leftarrow$" The construction of a (finite) model M of $C_{P,Q}$ from P, Q, and a solution $n_1, \ldots, n_m \in \mathbb{N}^m$ for $\mathbf{x}$ is not difficult. M can be constructed along the explanations given for $C_{P,Q}$ in the following way: We start at the bottom of the tree M by introducing instances

- x_k^1 of X_k^1 that have n_k R-successors, each of them having 1 as f-successor (to satisfy $E_{\mathtt{x}_k}^R$), n_k as $\mathtt{x}_k$ successor, and n_k as s-successor.
- x_k^0 of X_k^0 that have n_k R-successors, each of them having 1 as f-successor, n_k as $\mathtt{x}_k$ successor, and 1 as s-successor.

Then, for each monomial $\mathbf{x}^{\mathbf{i}}{}_j$, the corresponding subtrees representing $n_k^{i_{jk}}$ are built. Starting with (copies of) x_k^1 and x_k^0, we build trees of depth i_{jk} and degree n_k. Next, instances c of $C_{\mathbf{x}^{\mathbf{i}}j}$ are introduced, where each c has as R_k-successor the subtree representing the factor $n_k^{i_{jk}}$ in $n_1^{i_{j1}} \cdots n_m^{i_{jm}}$. Now, we append another subtree to each c, namely the one representing the multiplication of the values $n_k^{i_{jk}}$. This tree is of depth $m-1$ and degree $n_k^{i_{jk}}$ at level $k-1$. The remaining construction is straightforward. We first take a_j disjoint copies of the c's standing for $C_{\mathbf{x}^{\mathbf{i}}j}$ (including the corresponding subtree) as R-successors of an instance a of C_{A_j}, then we append these as as R-successors to an instance

p of C_P. We suppose that the same construction has been carried out for Q, which lead to an instance q of C_Q. Finally, p and q are P (resp. Q) -successors of an instance c of $C_{P,Q}$.

At each node of the tree constructed in this way (except for the root node), the s-successor of an individual equals the sum over the s-successors of its R-successors, and each node has the same $\mathtt{x}_k$-successors. The fact that a solution $n_1, \ldots, n_m \in \mathbb{N}^m$ for $\mathbf{x}$ has been used implies that p's s-successor coincides with q's s-successor, as required by the definition of $C_{P,Q}$.

"$\Rightarrow$" Given a model M for $C_{P,Q}$ with $c \in C_{P,Q}^{\mathcal{I}}$, due to the presence of Inv and similar concepts in $C_{P,Q}$, all $\mathtt{x}_k$-successors of all "relevant" role successors of c coincide—where "relevant" role successors are those whose existence is explicitly required by $C_{P,Q}$. Again, following the description of $C_{P,Q}$, we have that $(\mathtt{x}_1^{\mathcal{I}}(c), \ldots, \mathtt{x}_m^{\mathcal{I}}(c))$ is a solution for $P(\mathbf{x}) = Q(\mathbf{x})$. Due to the use of the concepts $E^R_{\mathtt{x}_k}$, this solution is in $\mathbb{N}^m$. $\square$

4.2 Tightening the result

A closer investigation of the concept $C_{P,Q}$ reveals that (a) negation occurs only in the concept no_f, (b) the only place where existential restriction occurs is in the concepts C_P and C_Q, and (c) the only place where disjunction $\sqcup$ occurs is in the concepts E^R_n describing individuals having exactly n R-successors.

We will show that the concepts no_f, E^R_n and C_P can be rewritten into equivalent concepts without negation, disjunction and existential restriction, by extending only slightly the set of concrete predicates. Hence, the reduction concept $C_{P,Q}$ can be written using only conjunction $\sqcap$ and universal value restriction $\forall R.C$. As introduced in [1], let $\mathcal{FL}_0$ denote the set of those concepts that are built using conjunction and universal value restriction only, and let $\mathcal{FL}_0(\Sigma)$ denote the extension of this language by concrete domains with aggregation. Then the following undecidability result is an immediate consequence of the possibility to rewrite the reduction concept $C_{P,Q}$ without using negation, disjunction, and existential restriction.

Theorem 10 *For a concrete domain Σ where*

- $\mathsf{dom}(\Sigma)$ *includes the non-negative integers* $\mathbb{N}$,
- $\mathsf{pred}(\Sigma)$ *contains, for all non-negative integers n, (unary) predicates $P_{=n}$ that test for equality with n, the (binary) equality predicate $P_=$, and the (binary) inequality predicate $P_{\neq}$,*
- $\mathsf{agg}(\Sigma)$ *contains* $\mathsf{min}, \mathsf{max}, \mathsf{sum}$,

satisfiability and subsumption of $\mathcal{FL}_0(\Sigma)$-concepts are undecidable.

Remark 11 (a) Admissible concrete domains as defined in [2] are closed under negation, e.g., the presence of a predicate $P_=$ in $\mathsf{pred}(\Sigma)$ implies the presence of its negation $P_{\neq}$. Hence for admissible concrete domains, the only difference between the preconditions of Theorem 8 and Theorem 10 are the unary predicates $P_{=n}$.

(b) We recall that, according to the semantics of $\mathcal{FL}_0(\Sigma)$, an individual a can only be an instance of the concept $P_{\neq}(f,g)$ if a has an f- as well as a g-successor in the concrete domain $\mathsf{dom}(\Sigma)$.

PROOF. As observed above, it suffices to define $\mathcal{FL}_0(\Sigma)$-concepts no'_f, ${E'_n}^R$, and C'_P which can play the rôle of no_f, E_n^R, and C_P in the reduction concept $C_{P,Q}$ of the proof of Theorem 8.

$\mathbf{\mathsf{no}'_f}$: This concept is used to make sure that an individual has no f-successor. It can clearly be replaced by

$$\mathsf{no}'_f := \forall f.P_{\neq}(g,g),$$

where $P_{\neq}(g,g)$ plays the rôle of the empty concept $A \sqcap \neg A$ used in the definition of no_f.

$\mathbf{{E'_n}^R}$: Given a concrete domain Σ that provides, for all non-negative integers n, a unary predicate $P_{=n}$ that tests for equality with n, we can define a concept ${E'_n}^R$ whose instances have exactly n R-successors:

$${E'_n}^R := \forall R.P_{=1}(f) \sqcap P_{=n}(\mathsf{sum}(R \circ f)).$$

Obviously, replacing E_n^R by ${E'_n}^R$ in $C_{P,Q}$ preserves its property of serving as a reduction concept for Hilbert's 10th problem. Avoiding existential restriction in C_P is more complicated.

$\mathbf{C'_P}$: In C_P, existential restrictions are used to make sure that, for each monomial A_j, there is one R-successor representing this monomial. This can also be expressed by introducing, for each j, exactly one R_j-successor (using $E_1^{R_j}$), and then using universal value restrictions to make sure that this R_j-successor is an instance of C_{A_j}. Additionally, the $\mathtt{x}_j$-successors are propagated to the R_j-successors. All this is ensured by the first line of C'_P.

$$\begin{aligned} C'_P := \bigsqcap_{0\le j\le n} \Big(& E_1^{R_j} \sqcap \forall R_j.C_{A_j} \sqcap \bigsqcap_{0\le \ell\le m} P_=(\mathtt{x}_\ell, \mathsf{sum}(R_j \circ \mathtt{x}_\ell)) \sqcap \\ & P_=(s_j, \mathsf{sum}(R_j \circ s)) \Big) \\ & \sqcap \mathsf{Add}_{s_0,\ldots,s_n} \end{aligned}$$

It remains to enforce that the sum over all s-successors of all R_j-successors of an instance p of C'_P coincides with p's s-successor. For this purpose, the second line of C'_P makes sure that p has an s_j-successor which coincides with the s-successor of its R_j-successor, and the concept $\mathsf{Add}_{s_0,\ldots,s_n}$ is used to sum up p's s_j-successors. It is defined as follows,

$$\mathsf{Add}_{s_0,\ldots,s_n} = \mathsf{add}^{s_{01}}_{s_0,s_1} \sqcap \mathsf{add}^{s_{012}}_{s_{01},s_2} \sqcap \ldots \sqcap \mathsf{add}^{s}_{s_{012\ldots n-1},s_n},$$

where

$$\begin{aligned}\mathsf{add}^{u}_{t,t'} := \; & {E'_2}^R \sqcap \forall R.P_=(g,g) \sqcap P_=(t,\mathsf{max}(R\circ g)) \sqcap P_=(t',\mathsf{min}(R\circ g)) \sqcap \\ & P_=(u,\mathsf{sum}(R,g))\end{aligned}$$

The idea underlying this addition is the following. Firstly, the addition of $n+1$ numbers is reduced to the addition of two numbers: In $\mathsf{Add}_{s_0,\ldots,s_n}$, the s_0- and the s_1-successor of p are summed up and the result is stored as s_{01}-successor of p. Similarly, the s_{01}- and the s_2-successor are summed up and the result is stored as s_{012}-successor of p, and so forth, until only two arguments are left. The sum of these last numbers is the result of the whole addition, and stored as s-successor of p.

Secondly, the addition of two numbers given as t- and t'-successors and the storage of the result as u-successor is realised by the concept $\mathsf{add}^u_{t,t'}$. Let p be an instance of $\mathsf{add}^u_{t,t'}$, let x be p's t-successor, and let x' be p's t'-successor. The first two conjuncts of $\mathsf{add}^u_{t,t'}$ ensure that p has exactly two R-successors, each of which has a g-successor in the concrete domain. Next, we ensure that x coincides with the maximum of p's $R\circ g$-successors, and that x' coincides with the minimum of p's $R\circ g$-successors. Hence $M_p^{R\circ g} = \{\{x,x'\}\}$, and thus the last conjunct ensures that p's u-successor coincides with $x+x'$.

Again, replacing C_P by C'_P and C_Q by C'_Q in $C_{P,Q}$ preserves its property of serving as a reduction concept for Hilbert's 10th problem, which is—together with the aforementioned replacements—an $\mathcal{FL}_0(\Sigma)$-concept.

Undecidability of subsumption follows from undecidability of satisfiability because a concept C is satisfiable iff it is not subsumed by an unsatisfiable concept, and because the $\mathcal{FL}_0(\Sigma)$-concept $P_{\neq}(f,f)$ is such an unsatisfiable concept. □

5 Decidability Results

The undecidability proofs in the previous section heavily use universal value restriction in combination with aggregation functions, in particular sum. In

this section, we will show that this interaction is indeed the cause for the undecidability: we will give three generic decidability results, which are all obtained by disallowing this kind of interaction.

The first result is obtained by restricting the abstract part of the Description Logic. In Section 5.1, $\mathcal{EL}(\Sigma)$ is obtained from $\mathcal{ALC}(\Sigma)$ by disallowing universal value restrictions. We present a tableau algorithm that decides satisfiability for this restricted logic. This algorithm is generic in that we give a rather weak property of concrete domains that implies decidability of satisfiability for $\mathcal{EL}(\Sigma)$-concepts. In Section 5.3, we present several concrete domains that satisfy this property, all of them involving the aggregation functions min, max, and count. However, since $\mathcal{EL}(\Sigma)$ is not closed under negation, the tableau algorithm cannot be used to decide subsumption.

For the second result, we have chosen a Description Logic that is closed under negation, i.e., where subsumption can be reduced to satisfiability. In Section 5.2, $\mathcal{ALC}(\Sigma)^-$ is obtained by restricting $\mathcal{ALC}(\Sigma)$ in such a way that no interaction between aggregation functions and universal value restrictions can occur. Since this Description Logic is propositionally closed, also existential restrictions cannot interact with aggregation functions. This interaction is possible in $\mathcal{EL}(\Sigma)$, and therefore $\mathcal{ALC}(\Sigma)^-$ is not an extension of $\mathcal{EL}(\Sigma)$. The concrete domains Σ for which $\mathcal{ALC}(\Sigma)^-$ is decidable are the same as those for which $\mathcal{EL}(\Sigma)$ has been proved decidable; they are described in Section 5.3. However, this second decidability result is not as generic as the first one because the way in which the algorithm treats negated concrete predicates strongly depends on the aggregation functions.

Finally, the third result is obtained by restricting the aggregation functions to min and max. For concrete domains Σ involving only min and max, decidability of satisfiability and subsumption of $\mathcal{ALC}(\Sigma)$-concepts is shown by a reduction to a known decidable Description Logic. From Section 4, it is clear that this result cannot be extended to a concrete domain also containing sum.

5.1 Decidability of $\mathcal{EL}(\Sigma)$

In this section, a generic decidability result is presented for $\mathcal{EL}(\Sigma)$, a restriction of $\mathcal{ALC}(\Sigma)$ that does not contain universal value restrictions. We start by defining $\mathcal{EL}(\Sigma)$.

Definition 12 (Syntax of $\mathcal{EL}(\Sigma)$) *$\mathcal{EL}(\Sigma)$ denotes the Description Logic that is obtained from $\mathcal{ALC}(\Sigma)$ by disallowing universal value restrictions ($\forall R.C$) and by restricting the use of negation to concept names.*

Satisfiability of $\mathcal{EL}(\Sigma)$-concepts is decided by a tableau algorithm that tries to construct, for an input concept C_0, a model of C_0. To this purpose, it breaks down C_0 into subconcepts, hereby making explicit all constraints on individuals in this model. It first works on the abstract part of the model while collecting constraints on the concrete part. If the abstract part is successfully processed, it ends with a set of concrete constraints for which satisfiability must be decidable, and whose solution can be used to construct the missing concrete part of the model. The attempt to construct a model either fails (in the abstract or the concrete part) with obvious inconsistencies—in which case C_0 is unsatisfiable—or it succeeds and ends with a description of a model of C_0.

In contrast to the algorithm in [2] for $\mathcal{ALC}(\mathcal{D})$, constraints now also involve variables for multisets over the concrete domain—besides individual variables for elements in the abstract and in the concrete domain. To capture the relation between individual and multiset variables, new constraints will be introduced to make explicit that an individual variable stands for an element of a multiset. Then, besides concrete individual variables, aggregated multiset variables can occur in predicate restrictions.

Definition 13 (Constraint Systems) *Let $\tau = \tau_A \cup \tau_\Sigma = \{a, b, c, \ldots\} \cup \{x, y, z, \ldots\}$ be an infinite set of abstract and concrete individual variables, and let $\sigma = \{X, Y, Z, \ldots\}$ be an infinite set of multiset variables. We assume that τ_A, τ_Σ, and σ are disjoint. The set of* aggregated variables, *$\{\Gamma(X) \mid \Gamma \in \mathsf{agg}(\Sigma)$ and $X \in \sigma\}$, is denoted by $\mathsf{agg}(\sigma)$. Constraints are of the form:*

$$
\begin{aligned}
a\!:\!C &\quad \text{for } a \in \tau_A, C \text{ an } \mathcal{EL}(\Sigma)\text{-concept},\\
(a,b)\!:\!R &\quad \text{for } a, b \in \tau_A, R \in N_R,\\
(a,\ell)\!:\!f &\quad \text{for } a \in \tau_A, \ell \in \tau, f \in N_F,\\
(a,Y)\!:\!(R \circ f) &\quad \text{for } a \in \tau_A, R \in N_R, f \in N_F, Y \in \sigma,\\
P(\alpha_1, \ldots, \alpha_n) &\quad \text{for } \alpha_i \in \tau_\Sigma \cup \mathsf{agg}(\sigma), \text{ and}\\
x\!:\!Y &\quad \text{for } x \in \tau_\Sigma, Y \in \sigma.
\end{aligned}
$$

Constraints of the form $P(\alpha_1, \ldots, \alpha_n)$ or $x : Y$ are called Σ-constraints. A constraint system *is a finite set of constraints. A variable ℓ is said to be an R-successor (resp. an $f_1 \ldots f_n$-successor) of an abstract variable a in a constraint system S iff $(a, \ell) : R \in S$ (resp. $(a, y_1) : f_1, (y_1, y_2) : f_2, \ldots, (y_{n-1}, \ell) : f_n \in S$ for some $y_1, \ldots, y_{n-1} \in \tau_A$). An aggregated variable $\Gamma(Y)$ is said to be an $f_1 \ldots f_n\,\Gamma(R \circ f)$-successor of a in S iff there is an $f_1 \ldots f_n$-successor b of a in S and $(b, Y) : (R \circ f) \in S$.*

Next, the semantics of constraint systems is defined. Since we want to decide satisfiability of $\mathcal{EL}(\Sigma)$-constraints where Σ involves the aggregation function

count (which returns the number of elements in a multiset), it will turn out to be crucial that no two abstract variables are interpreted by the same individual. Hence we will restrict our attention to so-called m-models.

Definition 14 (Semantics of constraints) *We consider interpretations $\mathcal{I}$ that, additionally, map individual variables to individuals of the concrete or the abstract domain, and multiset variables to finite multisets over the concrete domain, i.e.,*

$$\begin{aligned} a^{\mathcal{I}} &\in \Delta^{\mathcal{I}} && \text{for } a \in \tau_A, \\ x^{\mathcal{I}} &\in \mathsf{dom}(\Sigma) && \text{for } x \in \tau_\Sigma, \\ X^{\mathcal{I}} &\in \mathsf{MS}(\mathsf{dom}(\Sigma)) && \text{for } X \in \sigma. \end{aligned}$$

An interpretation $\mathcal{I}$ satisfies a constraint of the form

$$\begin{aligned} a{:}C \quad & \textit{iff} \quad a^{\mathcal{I}} \in C^{\mathcal{I}}, \\ (a,b){:}R \quad & \textit{iff} \quad (a^{\mathcal{I}}, b^{\mathcal{I}}) \in R^{\mathcal{I}}, \\ (a,\ell){:}f \quad & \textit{iff} \quad f^{\mathcal{I}}(a^{\mathcal{I}}) = \ell^{\mathcal{I}}, \\ (a,Y){:}(R \circ f) \quad & \textit{iff} \quad M^{R\circ f}_{a^{\mathcal{I}}} = Y^{\mathcal{I}}, \\ P(\alpha_1,\ldots,\alpha_n) \quad & \textit{iff} \quad P^\Sigma(\alpha_1^{\mathcal{I}},\ldots,\alpha_n^{\mathcal{I}}), \qquad (17) \\ x{:}Y \quad & \textit{iff} \quad x^{\mathcal{I}} \in Y^{\mathcal{I}}, \qquad (18) \end{aligned}$$

where, for $\alpha_i = \Gamma(X)$, we define $\Gamma(X)^{\mathcal{I}} := \Gamma^\Sigma(X^{\mathcal{I}})$.

A constraint system S is satisfiable *iff there exists an interpretation satisfying all constraints in S such that $b^{\mathcal{I}} \neq c^{\mathcal{I}}$ for all $b, c \in \tau_A$ with $b \neq c$ and $\{(a,b){:}\, R, (a,c) : R\} \subseteq S$ for some $a \in \tau_A$ and $R \in N_R$. Such an interpretation is called an* m-model *of S.*

For a constraint system S, the conjunction S_Σ is defined as follows:

$$S_\Sigma := \bigwedge_{P(\alpha_1,\ldots,\alpha_n)\in S} P(\alpha_1,\ldots,\alpha_n) \wedge \bigwedge_{Y \text{ occurs in } S} \{\{x_i \mid x_i{:}Y \in S\}\} \subseteq Y.$$

A solution *of S_Σ in Σ is a mapping $\hat{\cdot}$ that maps each individual variable x in S_Σ to an element $\hat{x} \in \mathsf{dom}(\Sigma)$ and each multiset variable Y in S_Σ to a finite multiset $\hat{Y}$ over $\mathsf{dom}(\Sigma)$ such that*

- *if $\Gamma(Y)$ is an aggregated variable in S_Σ, then $\Gamma^\Sigma(\hat{Y})$ is defined*[2] *and*
- *the result of applying $\hat{\cdot}$ to (all variables in) S_Σ is true in Σ, where $\subseteq$ is interpreted as multiset inclusion, each predicate name P as P^Σ, and each aggregation function Γ as Γ^Σ.*

[2] For example, we do not admit the empty set as a solution for Y in $P(\mathsf{min}(Y))$ since $\mathsf{min}(\emptyset)$ is undefined.

A constraint system S is Σ-consistent *iff S_Σ has a solution.*

A constraint system S contains a clash *iff*

- $\{a\!:\!C, a\!:\!\neg C\} \subseteq S$ *for some concept C, or*
- $\{(a,x) : f, (a,b) : f\} \subseteq S$ *for a concrete variable $x \in \tau_\Sigma$ and an abstract variable $b \in \tau_A$.*

A constraint system S contains a fork *iff, for $a \in \tau_A$ and a feature name $f \in N_F$, we have*

- $\{(a,\ell)\!:\!f, (a,\ell')\!:\!f\} \in S$ *for two distinct variables $\ell, \ell' \in \tau_A$ or $\ell, \ell' \in \tau_\Sigma$, or*
- $\{(a,Y)\!:\!(R \circ f), (a,Z)\!:\!(R \circ f)\} \in S$ *for two distinct variables $Y, Z \in \sigma$.*

If a constraint system S contains a fork $\{(x,\ell) : f, (x,\ell') : f\}$ (resp. $\{(a,Y) : (R \circ f), (a,Z) : (R \circ f)\}$), then we say that S' is obtained by fork elimination *from S iff S' is obtained from S by replacing each occurrence of ℓ by ℓ' (resp. Y by Z).*

The *tableau algorithm* for deciding satisfiability of $\mathcal{EL}(\Sigma)$-concepts works on a tree where each node is labelled with a constraint system. It starts with the tree consisting of a single leaf, the root, labelled with $S = \{a_0 : C_0\}$, where C_0 is the $\mathcal{EL}(\Sigma)$-concept to be tested for satisfiability. The tableau algorithm applies the *completion rules* introduced in Figure 5 to constraint systems. For Rule 4, recall the definition of u-successors for aggregated features u in Definition 13. A rule can only be applied to a leaf labelled with a clash-free constraint system. Applying a rule $S \to S_j$, for $1 \leq j \leq n$, to such a leaf leads to the creation of n new successors of this node, where the j-th successor is labelled with S_j. The algorithm terminates if none of the rules can be applied to any of the leaves.

A constraint system S is *complete* if none of the completion rules can be applied to S. The tableau algorithm answers "C_0 is satisfiable" iff after its termination one of the leaves is labelled with a complete, clash-free, and Σ-consistent constraint system.

Lemma 15 *Let C_0 be an $\mathcal{EL}(\Sigma)$-concept, and let S be a constraint system obtained by applying the completion rules to $\{a_0\!:\!C_0\}$.*

(1) If C_0 is satisfiable, then $\{a_0\!:\!C_0\}$ has an m-model.
(2) Let $\mathcal{R}$ be a completion rule that can be applied to S. Then S is satisfiable iff one of the systems S_i obtained by applying $\mathcal{R}$ to S is satisfiable.
(3) If S is a complete, Σ-consistent, and clash-free constraint system, then S has an m-model.
(4) If S contains a clash or is not Σ-consistent, then S does not have an m-model.

1. Conjunction: If $a\!:\!(C_1 \sqcap C_2) \in S$ and $a\!:\!C_1 \notin S$ or $a\!:\!C_2 \notin S$, then
$S \to \;\; S \cup \{a\!:\!C_1, a\!:\!C_2\}$.

2. Disjunction: If $a\!:\!(C_1 \sqcup C_2) \in S$ and $a\!:\!C_1 \notin S$ and $a\!:\!C_2 \notin S$, then
$S \to \;\; S_1 = S \cup \{a\!:\!C_1\}$,
$S \to \;\; S_2 = S \cup \{a\!:\!C_2\}$.

3.a. Existential restriction on roles: If $a\!:\!(\exists R.C) \in S$ for a role name R, $\{b_1, \ldots, b_n\}$ are all R-successors of a, and $b_i : C \notin S$ for all b_i, then
$S \;\; \to S_i \;\;\; = S \cup \{b_i : C\}$
$S \;\; \to S_{n+1} = S \cup \{(a, b)\!:\!R, b : C\}$
for a new variable $b \in \tau_A$.

3.b. Existential restriction on features: If $a\!:\!(\exists f.C) \in S$ for a feature name f and if there is no f-successor b of a with $b\!:\!C \in S$, then
$S \;\; \to S \cup \{(a, b)\!:\!f, b : C\}$.
for a new variable $b \in \tau_A$. If forks were created, then eliminate these forks.

4. Concrete predicates: If $a\!:\!P(u_1, \ldots, u_n) \in S$ and a does not have u_i-successors α_i with $P(\alpha_1, \ldots, \alpha_n) \in S$, then, for each u_i let

$$S_i := \begin{cases} \{(a, b_{i1})\!:\!f_{i1}, (b_{i1}, b_{i2})\!:\!f_{i2}, \ldots, (b_{im_i-1}, y_i)\!:\!f_{im_i}\} \\ \qquad\qquad \text{if } u_i = f_{i1} f_{i2} \ldots f_{im_i} \\ \{(a, b_{i1})\!:\!f_{i1}, (b_{i1}, b_{i2})\!:\!f_{i2}, \ldots, (b_{im_i-1}, b_{im_i})\!:\!f_{im_i}, (b_{im_i}, Y_i)\!:\!(R_i \circ f_i)\} \\ \qquad\qquad \text{if } u_i = f_{i1} f_{i2} \ldots f_{im_i} \Gamma_i (R_i \circ f_i) \end{cases}$$

for new variables $b_{ij} \in \tau_A, y_i \in \tau_\Sigma$, $Y_i \in \sigma$. Let α_i be the u_i-successor of a in S_i. Then
$S \to \;\; S \cup \{P(\alpha_1, \ldots, \alpha_n)\} \cup \bigcup_{1 \le i \le n} S_i$.
If forks were created, then eliminate these forks.

5. Element assertions: If $\{(a, b)\!:\!R, (b, z)\!:\!f, (a, Y)\!:\!(R \circ f)\} \subseteq S$ for $z \in \tau_\Sigma$ and $z\!:\!Y \notin S$ then
$S \to \;\; S \cup \{z\!:\!Y\}$.

Fig. 5. The completion rules for $\mathcal{EL}(\Sigma)$.

(5) The tableau algorithm terminates when applied to $\{a_0\!:\!C_0\}$.

Before proving Lemma 15, let us comment on Rule 3.a, which is non-standard. Consider, for example, the following C:

$$C := (\exists R.\, P_{\ge 2}(f)) \sqcap (\exists R.\, P_{=2}(f)) \sqcap P_{\le 1}(\mathsf{count}(R \circ f))$$

This concept contains the aggregation function count, which returns the number of (not necessarily distinct) elements of a multiset, and is satisfiable. However, a tableau algorithm that generates, for each constraint of the form

$a:\exists R.C$, a new R-successor of a would either not detect that C is satisfiable (i.e., it would be incomplete), or it would need to switch back to abstract reasoning (and identify both R-successors of a) after having tested the consistency of the concrete constraints. The latter alternative can easily be seen to necessitate alternation between Σ-consistency checks and tableau rule applications, and thus makes termination of the whole algorithm problematic. To design a complete tableau algorithm that switches only once from "abstract" to "concrete" reasoning, existential restrictions are handled by trying all possibilities to generate as few R-successors as possible. This is realised by trying to reuse, for a constraint $a:(\exists R.C)$, already existing R-successor of a. For the case that this reuse is not possible, a new R-successor is also introduced. As a consequence, we can restrict our attention to those models that interpret different R-successors as different individuals, i.e., to m-models.

PROOF. (Lemma 15.1:) Each model of C_0 is obviously an m-model of $\{a_0 : C_0\}$.

(Lemma 15.2:) (ii)⇒(i) is obvious because each S_i obtained by applying the completion rules to S is a superset of S where variables were possibly renamed due to fork elimination, and thus an m-model of S_i is also an m-model of S (modulo the mapping of renamed variables).

(i)⇒(ii): We only consider Rules 3.a, 4, and 5 because Rules 1, 2, and 3.b are obvious and similar to those used in other tableau-based algorithms; see, for example, [15,2].

Let $\mathcal{I}$ be an m-model of S as defined in the precondition of Rule 3.a. Hence there is some $c \in \Delta^{\mathcal{I}}$ with $(a^{\mathcal{I}}, c) \in R^{\mathcal{I}}$ and $c \in C^{\mathcal{I}}$. If $b_i^{\mathcal{I}} = c$ for some R-successor b_i of a, then $\mathcal{I}$ is an m-model of S_i. Otherwise, $\mathcal{I}$ extended with $b^{\mathcal{I}} = c$ is an m-model of S_{n+1}.

Let $\mathcal{I}$ be an m-model of S as defined in the precondition of Rule 4 and let S' be obtained by applying Rule 4 to S. Then $a:P(u_1, \ldots, u_n) \in S$ and, for each u_i with $1 \leq i \leq n$, if

- u_i is a feature chain $f_{i1} f_{i2} \ldots f_{im_i}$, then $a^{\mathcal{I}}$ has $f_{i1} \ldots f_{ij}$-successors $c_{ij} \in \Delta^{\mathcal{I}}$ for $1 \leq j < m_i$, and an $f_{i1} f_{i2} \ldots f_{im_i}$-successor $z_{im_i} \in \mathsf{dom}(\Sigma)$. If we define $b_{ij}^{\mathcal{I}} = c_{ij}$ and $y_{im_i}^{\mathcal{I}} = z_{im_i}$, then $\mathcal{I}$ satisfies S_i as defined in Rule 4.
- u_i is an aggregated feature $f_{i1} f_{i2} \ldots f_{im_i} \Gamma_i(R_i \circ f_i)$, then $a^{\mathcal{I}}$ has $f_{i1} \ldots f_{ij}$-successors $c_{ij} \in \Delta^{\mathcal{I}}$ for $1 \leq j \leq m_i$. If we define $b_{ij}^{\mathcal{I}} = c_{ij}$ and $Y^{\mathcal{I}} = M^{R_i \circ f_i}_{c_{im_i}}$, then $Y^{\mathcal{I}}$ is by definition the appropriate multiset, and $\mathcal{I}$ satisfies S_i as defined in Rule 4.

Given $\mathcal{I}$ as extended above to the newly introduced variables and α_i as defined in Rule 4, we have that α_i is indeed interpreted as the u_i-successor of a, namely

$u_i^{\mathcal{I}}(a^{\mathcal{I}}) = \alpha_i^{\mathcal{I}}$ for all $1 \leq i \leq n$. Since $\mathcal{I}$ satisfies $a\!:\!P(u_1, \ldots, u_n)$, we thus have that $\mathcal{I}$ satisfies $P(\alpha_1, \ldots, \alpha_n)$.

Let $\mathcal{I}$ be an m-model of S, and let S' be obtained by applying Rule 5 to S. Then $\{(a,b) : R, (b,z) : f, (a,Y) : (R \circ f)\} \subseteq S$ and $z \in \tau_\Sigma$. Thus $z^{\mathcal{I}}$ is an f-successor of an R-successor of $a^{\mathcal{I}}$ in $\mathsf{dom}(\Sigma)$. By definition, $z^{\mathcal{I}} \in M_{a^{\mathcal{I}}}^{R \circ f}$, and, since $\mathcal{I}$ is an m-model of S, $Y^{\mathcal{I}} = M_{a^{\mathcal{I}}}^{R \circ f}$. Hence $z^{\mathcal{I}} \in Y^{\mathcal{I}}$, and thus $\mathcal{I}$ satisfies $S \cup \{z\!:\!Y\} = S'$.

(Lemma 15.3:) Let S be a complete, Σ-consistent, and clash-free constraint system involving concrete and multiset variables $\{x_1, \ldots, x_m, X_1, \ldots, X_n\}$, and let $\hat{\cdot}$ be a solution for S_Σ. In particular, we have $\{\{\hat{x}_j \mid x_j\!:\!X_i \in S\}\} \subseteq \hat{X}_i$ for all multiset variables X_i occurring in S. To define an m-model for S, we first define a "quasi-model" $\mathcal{I}'$ as follows:

$$
\begin{aligned}
\Delta^{\mathcal{I}'} &:= \tau_A, \\
a^{\mathcal{I}'} &:= a \text{ for abstract variables } a \in \tau_A, \\
x^{\mathcal{I}'} &:= \hat{x} \text{ for concrete variables } x \in \tau_\Sigma, \\
X^{\mathcal{I}'} &:= \hat{X} \text{ for multiset variables } X \in \sigma, \\
A^{\mathcal{I}'} &:= \{b \in \Delta^{\mathcal{I}'} \mid b\!:\!A \in S\} \text{ for concept names } A \in N_C, \\
R^{\mathcal{I}'} &:= \{(a,b) \in \Delta^{\mathcal{I}'} \times \Delta^{\mathcal{I}'} \mid (a,b)\!:\!R \in S\} \text{ for role names } R \in N_R, \\
f^{\mathcal{I}'}(c) &:= \begin{cases} b & \text{if } (c,b)\!:\!f \in S \text{ for } b \in \tau_A, \\ \hat{x} & \text{if } (c,x)\!:\!f \in S \text{ for } x \in \tau_\Sigma, \\ \text{undefined else.} \end{cases} \\
& \qquad\qquad \text{for feature names } f \in N_F.
\end{aligned}
$$

The interpretation of feature names f is well-defined because S is clash-free and contains no forks. The only reason why $\mathcal{I}'$ might not be an m-model of S is the following. An abstract individual a may have less R-successors having an f-successor in $\mathsf{dom}(\Sigma)$ than required by the solution for the corresponding multiset variable X_i, that is, for constraints $(a, X_i)\!:\!(R \circ f) \in S$, we might have $M_{a^{\mathcal{I}'}}^{R \circ f} \subsetneq \hat{X}_i$. Due to the absence of universal value restrictions, an m-model $\mathcal{I}$ of S can be obtained from $\mathcal{I}'$ by simply adding R-successors d_a^{Rfj} and the lacking $R \circ f$-successors $\hat{y}_a^{Rfj}$. More precisely, for a multiset variable X with $(a, X)\!:\!(R \circ f) \in S$, let

$$\hat{X} \setminus M_a^{R \circ f} = \{\{\hat{y}_a^{Rf1}, \hat{y}_a^{Rf2}, \ldots\}\}.$$

Then

$$\Delta^{\mathcal{I}} := \Delta^{\mathcal{I}'} \uplus \bigcup_{(a,X):(R\circ f)\in S} \{d_a^{Rfj} \mid \hat{y}_a^{Rfj} \in \hat{X} \setminus M_a^{R\circ f}\},$$

$$A^{\mathcal{I}} := A^{\mathcal{I}'},$$

$$R^{\mathcal{I}} := R^{\mathcal{I}'} \cup \bigcup_{(a,X):(R\circ f)\in S} \{(a, d_a^{Rfj}) \mid \hat{y}_a^{Rfj} \in \hat{X} \setminus M_a^{R\circ f}\}$$

for role names $R \in N_R$,

$$f^{\mathcal{I}}(c) := \begin{cases} b & \text{if } (c,b) : f \in S, \\ \hat{x} & \text{if } (c,x) : f \in S, \\ \hat{y}_a^{Rfj} & \text{if } c = d_a^{Rfj} \\ \text{undefined else.} \end{cases} \qquad \text{for feature names } f \in N_F$$

Clearly, for two abstract variables $a \neq b$ we have $a^{\mathcal{I}} \neq b^{\mathcal{I}}$. Thus, it remains to show that $\mathcal{I}$ satisfies all constraints in S. This can be easily done by induction on the structure of concepts. By construction of $\mathcal{I}$, $\hat{X}_i = M_{a^{\mathcal{I}}}^{R\circ f}$ for all multiset variables with $(a, X_i) : (R \circ f) \in S$. Furthermore, $\hat{\cdot}$ being a solution for S_Σ implies that $\mathcal{I}$ satisfies all Σ-constraints in S. By definition, $\mathcal{I}$ satisfies all constraints of the form $(a,b) : R$, $(a,b) : f$, $(a,x) : f$, and $b : A$ for concept names A. Since S is clash-free, $\mathcal{I}$ satisfies all constraints of the form $b : \neg A$. By induction and because S is complete, $\mathcal{I}$ satisfies all constraints of the form $a : (C_1 \sqcap C_2)$, $a : (C_1 \sqcup C_2)$, and $a : (\exists R.C)$ for role or feature names R.

(Lemma 15.4:) Obviously, a constraint system containing a clash cannot have an m-model. For Σ-consistency, we show that an m-model $\mathcal{I}$ of S yields a solution of the conjunction S_Σ, which is defined in Definition 14 as follows:

$$\bigwedge_{P(\alpha_1,\ldots,\alpha_n)\in S} P(\alpha_1, \ldots, \alpha_n) \wedge \bigwedge_{Y \text{ occurs in } S} \{\{x_i \mid x_i : Y \in S\}\} \subseteq Y$$

Due to Line 17 of Definition 14, $\mathcal{I}$ satisfies all constraints of the form $P(\alpha_1, \ldots, \alpha_n) \in S$, and thus the first part of S_Σ. Line 18 of Definition 14 implies that $\mathcal{I}$ satisfies the inclusions in S_Σ when read with *set* semantics. Now, if Y occurs in S, then, by definition of the semantics, $M_{a^{\mathcal{I}}}^{R\circ f} = Y^{\mathcal{I}}$ for $(a, Y) : (R \circ f) \in S$. Since $\mathcal{I}$ is an m-model, all R-successors of a in S are interpreted as different objects, and thus $Y^{\mathcal{I}} = \{\{x_i^{\mathcal{I}} \mid x_i : Y \in S\}\}$. Thus, $\mathcal{I}$ also satisfies the second part of S_Σ.

(Lemma 15.5:) Termination is an immediate consequence of the fact that (i) the relational structure of the constraint systems generated by the tableau algorithm are trees, (ii) all concepts in constraints added by the completion rules are subconcepts of the concept C_0, whose number is linear in the length of C_0, (iii) these trees are of bounded width and breadth, and that (iv) these

trees are generated in a monotonic way, i.e., no constraints are removed. Properties (i), (ii), and (iv) are an immediate consequence of the definition of the completion rules. Property (iii) is due to the fact that (iii)′ the (maximum) length of concepts occurring in constraints on a successor node of a is less than the (maximum) length of concepts in constraints on a and (iii)″ the generation of new successors is triggered by constraints of the form $a:C$ (Rule 3 and 4) or $(a:Y):(R \circ f)$ (Rule 5). Each such constraint triggers at most once the generation of new variables. To see that Property (iv) holds, we eliminate forks in such a way that "old" variables are kept in the constraint system and, if replacement is necessary, new variables are replaced with "old" ones. □

Remark 16 In the proof of Lemma 15.3, the extension of $\mathcal{I}'$ to an m-model $\mathcal{I}$ of a complete and clash-free constraint system was only possible because we disallowed the use of universal value restriction: This enables us to add lacking $R \circ f$-successors for some a without the necessity to check again whether these new R-successors satisfy all universal value restrictions $a:\forall R.C$.

As an immediate consequence of Lemma 15, we have the following decidability result.

Theorem 17 *If Σ is a concrete domain such that Σ-consistency is decidable, then satisfiability of $\mathcal{EL}(\Sigma)$-concepts is decidable.*

In Section 5.3, we will show decidability of Σ-consistency for various concrete domains involving min, max, count, and comparisons (possibly with constants). Next, we will describe a decidable Description Logic with concrete domains and aggregation functions that is propositionally closed.

5.2 Decidability of $\mathcal{ALC}(\Sigma)^-$

So far, we have only proved decidability of *satisfiability* of $\mathcal{EL}(\Sigma)$-concepts. However, $\mathcal{EL}(\Sigma)$ is not closed under negation, and thus subsumption cannot be reduced to satisfiability.[3] Closing $\mathcal{EL}(\Sigma)$ under negation, one obtains $\mathcal{ALC}(\Sigma)$, and the key problem one encounters when trying to extend the tableau algorithm to decide satisfiability of $\mathcal{ALC}(\Sigma)$-concepts (and thus also subsumption of $\mathcal{ALC}(\Sigma)$) was already discussed in Remark 16, i.e., R-successors required by a solution of a multiset variable Y cannot be simply added to a quasi-model since they might be subject to universal value restrictions. More importantly, generating a "prophylactic" R-successor from which the missing ones could be copied does not even work for the aggregation functions min, max, and count.

[3] So far, it is unclear for which Σ subsumption of $\mathcal{EL}(\Sigma)$-concepts is decidable—the only exceptions are the domains mentioned in Theorem 26.

For example, consider the following concept

$$P_{\leq 1}(\mathsf{count}(R \circ f)) \sqcap \forall R.\top_{\Sigma}(f) \sqcap$$
$$P_{\geq 7}(\mathsf{max}(R \circ h)) \sqcap P_{\leq 3}(\mathsf{min}(R \circ h)) \sqcap \exists R.P_{=5}(h)$$

Let a be an instance of this concept. The first conjunct ensures that a has at most one R-successor with an f-successor in $\mathsf{dom}(\Sigma)$. The universal restriction implies that each R-successor of a has an f-successor in $\mathsf{dom}(\Sigma)$. Finally, the second line implies that a has at least three R-successors, and thus this concept is unsatisfiable. This interaction between universal value restrictions and concrete domain predicates seems to prohibit to do first reasoning on the abstract domain, then on the concrete domain, and then stop. In contrast, it seems to require various steps back and forth between abstract and concrete reasoning, for which one would need to guarantee termination while not corrupting correctness. The undecidability results in Section 4 imply that this is possible only for rather restricted concrete domains. Hence we consider $\mathcal{ALC}(\Sigma)^-$, a propositionally closed restriction of $\mathcal{ALC}(\Sigma)$ where this interaction cannot occur.

Definition 18 *Let C, D be $\mathcal{ALC}(\Sigma)$-concepts where C is a sub-expression of D and let $\mathbf{R} = R_1 \cdots R_n$ be a (possibly empty) chain of role or feature names. Then C is at* level $\mathbf{R}$ *in D iff*[4]

- $\mathbf{R}$ *is empty and $C = D$,*
- $D = \neg D_1$ *and C is at level $\mathbf{R}$ in D_1, or*
- $D = D_1 \sqcap D_2$ *or $D = D_1 \sqcup D_2$ and C is at level $\mathbf{R}$ in D_1 or D_2, or*
- $D = \exists R.D_1$ *or $D = \forall R.D_1$, $\mathbf{R} = R\mathbf{R}'$, and C is at level $\mathbf{R}'$ in D_1.*

An $\mathcal{ALC}(\Sigma)$-concept D is an $\mathcal{ALC}(\Sigma)^-$-concept if, whenever a concept of the form $P(\ldots, f_1 \ldots f_k\, \Gamma(R \circ f), \ldots)$ is at level $\mathbf{R}$ in D, then no concept of the form $\exists R.C$ or $\forall R.C$ is at level $\mathbf{R}f_1 \ldots f_k$ in D.

For example,

$$\exists R.(B \sqcap \forall S.P_{\geq}(h, f\mathsf{max}(T \circ g)))$$

is an $\mathcal{ALC}(\Sigma)^-$-concept, but

$$\exists R.(B \sqcap \forall S.P_{\geq}(h, f\mathsf{max}(T \circ g))) \sqcap \forall R.\exists S.\forall f.\exists T.A$$

is not an $\mathcal{ALC}(\Sigma)^-$-concept because $P_{\geq}(\cdot, f\mathsf{max}(T \circ g))$ occurs on level RS in it, and an existential restriction on T occurs at level RSf in it. Please note also that $\mathcal{ALC}(\Sigma)^-$ is not an extension of $\mathcal{EL}(\Sigma)$. For example, $\exists R.A \sqcap P(\Gamma(R \circ f))$

[4] Please note that a concept can be at several levels in another one.

is an $\mathcal{EL}(\Sigma)$-concept, but not an $\mathcal{ALC}(\Sigma)^-$-concept. Finally, it is easily verified that the set of $\mathcal{ALC}(\Sigma)^-$-concepts is closed under negation.

In the following, we will present an extended tableau algorithm that decides satisfiability (and thus also subsumption) of $\mathcal{ALC}(\Sigma)^-$-concepts, provided that Σ-consistency is decidable. The extended algorithm works on $\mathcal{ALC}(\Sigma)^-$-concepts in *negation normal form* (NNF), i.e., concepts where negation occurs in front of concept names only.

This normal form is more complex than usual, and it depends on the aggregation functions available. In the following, we assume that the concrete domain Σ is as defined in Corollary 25, i.e., the only aggregation functions considered are min, max, and count.

In the definition of the NNF, we use an abbreviation $\mathsf{NC}(u)$ that describes those individuals having no u-successors in the concrete domain, which is explained in detail after the definition. For aggregated features u, $\mathsf{NC}(u)$ depends on the aggregation function in u, and is defined differently for count and min or max because the former is defined on *all* finite multisets, whereas the latter are undefined on the empty multiset.

Definition 19 (NNF) *For a feature chain $u = f_1 \dots f_k$, define*

$$\lambda(u) = \top_\Sigma(f_1) \sqcup \top_\Sigma(f_1 f_2) \sqcup \dots \sqcup \top_\Sigma(f_1 \dots f_k),$$

where $\top_\Sigma$ denotes the unary concrete predicate for the concrete domain $\mathsf{dom}(\Sigma)$. Again, we use $\top_A$ as an abbreviation for $A \sqcup \neg A$, and we use $\bot_A$ as an abbreviation for $A \sqcap \neg A$. For a concrete feature u, $\mathsf{NC}(u)$ is defined as follows:

$$\mathsf{NC}(u) := \begin{cases} \lambda(f_1 \dots f_{k-1}) \sqcup \forall f_1.\forall f_2.\dots.\forall f_k.\top_A & \text{if } u = f_1 \dots f_k \\ \lambda(f_1 \dots f_k) \sqcup \forall f_1.\dots.\forall f_k.\forall R.\forall f.\top_A & \text{if } u = f_1 \dots f_k \mathsf{min}(R \circ f) \\ & \text{or } u = f_1 \dots f_k \mathsf{max}(R \circ f) \\ \lambda(f_1 \dots f_k) \sqcup \forall f_1.\dots.\forall f_k.\bot_A & \text{if } u = f_1 \dots f_k \mathsf{count}(R \circ f) \end{cases}$$

An $\mathcal{ALC}(\Sigma)^-$-concept is in negation normal form *(NNF) iff negation occurs only in front of concept names. If Σ is admissible,*[5] *each $\mathcal{ALC}(\Sigma)^-$-concept can be transformed into NNF by pushing negation inwards, making use of the*

[5] See Definition 4.

following equivalences:

$$\neg(C \sqcup D) \equiv \neg C \sqcap \neg D \qquad \neg(C \sqcap D) \equiv \neg C \sqcup \neg D \qquad \neg\neg C \equiv C$$
$$\neg(\exists R.C) \equiv (\forall R.\neg C) \qquad \neg(\forall R.C) \equiv (\exists R.\neg C)$$
$$\neg(P(u_1, \ldots, u_n)) \equiv \overline{P}(u_1, \ldots, u_n) \sqcup \bigsqcup_{1 \le i \le n} \mathsf{NC}(u_i) \quad ,$$

where $\overline{P}$ is the concrete predicate for the negation of P.

All but the last equivalence of the above definition are obvious. The last one is due to the fact that, for each interpretation $\mathcal{I}$ and concrete feature u, a is an instance of $\mathsf{NC}(u)$ iff a has no u-successor in $\mathsf{dom}(\Sigma)$.

$u = f_1 \ldots f_k$: For a having no u-successor in $\mathsf{dom}(\Sigma)$, there are two possibilities.

(i) The feature chain "goes too early" into the concrete domain, i.e, there is an $\ell < k$ such that a has an $f_1 \ldots f_\ell$-successor in the concrete domain. This case is covered by $\lambda(f_1 \ldots f_{k-1})$.

(ii) The feature chain "remains" in the abstract domain (including the case where it "breaks too early"). This case is covered by the second disjunct of $\mathsf{NC}(u)$.

$u = f_1 \ldots f_k \Gamma(R \circ f)$ for $\Gamma \in \{\mathsf{min}, \mathsf{max}\}$: Again, there are two possibilities for a having no u-successor in $\mathsf{dom}(\Sigma)$.

(i) a has no $f_1 \ldots f_k$-successor in $\Delta^{\mathcal{I}}$. This is the case if (i)$'$ a has, for some $\ell \le k$, an $f_1 \ldots f_\ell$-successor in $\mathsf{dom}(\Sigma)$, or if (i)$''$ a has, for some $1 \le \ell < k$, an $f_1 \ldots f_\ell$-successor in $\Delta^{\mathcal{I}}$ having no $f_{\ell+1}$-successor. Case (i)$'$ is covered by $\lambda(f_1 \ldots f_k)$, and case (i)$''$ by the second disjunct.

(ii) a has an $f_1 \ldots f_k$-successor $a' \in \Delta^{\mathcal{I}}$ and $\Gamma^{\Sigma}(M_{a'}^{R \circ f})$ is undefined. Since (on finite multisets) min and max are undefined only on the empty set, the second disjunct also covers this case.

$u = f_1 \ldots f_k \mathsf{count}(R \circ f)$: Analogously to the previous case, with the only difference that, since count is defined on *all* finite multisets, (ii) cannot occur and thus the second disjunct correctly covers case (i)$''$.

To obtain a sound and complete tableau algorithm for concepts involving universal value restrictions, we extend the tableau algorithm defined in Section 5.1 by Rule 6 in Figure 6.

6. Universal value restriction: If $a : (\forall R.C) \in S$ for a feature or role name R and if there is an R-successor b of a with $b : C \notin S$, then

$S \ \rightarrow S \cup \{b : C\}$

Fig. 6. The completion rule for universal value restrictions.

We use the same technical lemma as in Section 5.1 to show that the tableau algorithm decides satisfiability (and thus subsumption) of $\mathcal{ALC}(\Sigma)^-$-concepts.

Lemma 20 *Let Σ be an admissible concrete domain with aggregation functions* min, max, *and* count, *let C_0 be an $\mathcal{ALC}(\Sigma)^-$-concept in NNF, and let S be a constraint system obtained by applying the modified completion rules to $\{a_0 : C_0\}$.*

(1) If C_0 is satisfiable, then $\{a_0 : C_0\}$ has an m-model.
(2) Let $\mathcal{R}$ be a completion rule that can be applied to S. Then S is satisfiable iff one of the systems S_i obtained by applying $\mathcal{R}$ to S is satisfiable.
(3) If S is a complete, Σ-consistent, and clash-free constraint system, then S has an m-model.
(4) If S contains a clash or is not Σ-consistent, then S does not have an m-model.
(5) The tableau algorithm terminates when applied to $\{a_0 : C_0\}$.

PROOF. We prove only those parts of Lemma 20 that are different from those in Lemma 15.

(Lemma 20.2 (i)⇒(ii):) Rule 6 is standard; see, e.g., [2].

(Lemma 20.3:) An m-model $\mathcal{I}$ for a clash-free, complete, and Σ-consistent constraint system S can be defined in a way similar to the one in the proof of Lemma 15.3. Let S be a clash-free, complete, and Σ-consistent constraint system, $\hat{\cdot}$ a solution for S_Σ, and $\mathcal{I}'$ the "quasi-model" of S as defined in the proof of Lemma 15.3. Then $\mathcal{I}'$ can be extended in the same way to an m-model of S as in the proof of Lemma 15.3 since the additional elements d_a^{Rfj} are not subject to universal value restrictions due to the syntactic restriction of $\mathcal{ALC}(\Sigma)^-$.

The proof of Lemma 20.4 is similar to the proof of Lemma 15.4. Again, an m-model of S obviously yields a solution for S_Σ.

The proof of Lemma 20.5 is similar to the proof of Lemma 15.5 with the additional observation that also Rule 6 adds only constraints that are shorter than those that triggered the applicability of this rule. □

As a consequence, we have the following decidability result.

Theorem 21 *If Σ is an admissible concrete domain such that*

- $\mathsf{agg}(\Sigma) = \{\mathsf{min}, \mathsf{max}, \mathsf{count}\}$ *and*
- *Σ-consistency is decidable,*

then satisfiability and subsumption of $\mathcal{ALC}(\Sigma)^-$-concepts is decidable.

This result is not as generic as the decidability result for $\mathcal{EL}(\Sigma)$-concepts in that it is concerned with a fixed set of aggregation functions. It could have been formulated more generically by using an NNF with a predicate $\mathsf{undefined}(\Gamma(Y))$ on aggregated multiset variables. However, this would have been unnecessarily complicated and would not have given any new insights. Thus, to adapt the tableau algorithm to concrete domains with other aggregation functions, the NNF has to be modified appropriately. The result applies to all aggregation functions for which this is possible. For example, sum could be treated in the same way as count.

In the proof of the soundness of the algorithm, "copies" of abstract variables were used to generate abstract individuals that are required by the solution of the concrete constraints. This was only possible due to the syntactic restriction of $\mathcal{ALC}(\Sigma)^-$. In Section 5.4, we will see that this restriction can be removed for certain concrete domains involving only the aggregation functions min and max.

5.3 Concrete domains for which Σ-consistency is decidable

In Theorem 17, we have seen that decidability of Σ-consistency implies decidability of satisfiability of $\mathcal{EL}(\Sigma)$-concepts. The same condition implies that satisfiability and subsumption of $\mathcal{ALC}(\Sigma)^-$-concepts is decidable (Theorem 21). In this section, we will give examples for concrete domains for which Σ-consistency is indeed decidable. Basically, the behaviour of aggregation functions is axiomatised so that aggregated multiset variables $\Gamma(Y)$ can be replaced by individual variables y_Γ, a technique also used in [34]. For $\mathsf{dom}(\Sigma)$ the set of non-negative integers, integers, or rational numbers, the relations $<$, $>$, $=$, $\leq$, and $\geq$ are defined as usual. Furthermore, for $n \in \mathsf{dom}(\Sigma)$, the unary predicates $=_n, \leq_n, \geq_n, >_n, <_n$ are comparisons with n.

Lemma 22 *If Σ is a concrete domain such that*

- $\mathsf{dom}(\Sigma)$ *is the set of non-negative integers, integers, or rational numbers,*
- $\mathsf{pred}(\Sigma) = \{P_<, P_\geq, P_>, P_\leq, P_=\} \cup \bigcup_{n \in \mathsf{dom}(\Sigma)} \{P_{\leq_n}, P_{\geq_n}, P_{>_n}, P_{<_n}, P_{=_n}\}$, *and*
- $\mathsf{agg}(\Sigma) = \{\mathsf{min}, \mathsf{max}\}$,

then Σ-consistency is decidable.

Obviously, each Σ in Lemma 22 satisfies the first two properties in the definition of admissibility (Definition 4), and the third one follows from the proof of Lemma 22. Thus each Σ in Lemma 22 satisfies all conditions of Theorem 17 and 21.

PROOF. Let S be a constraint system, let Σ be defined as in the precondition of Lemma 22, and let S_Σ be the conjunction of Σ-constraints in S as defined in Definition 14.

Σ-consistency of S is equivalent to satisfiability of S_Σ and is decided by transforming S_Σ into a set D_S of linear (in)equalities without aggregation functions that is satisfiable iff S_Σ is satisfiable. Satisfiability of D_S can then easily be decided using—depending on the concrete domain—linear or integer programming [30]. To this purpose, for each term $\mathsf{max}(Y)$ (resp. $\mathsf{min}(Y)$) occurring in S_Σ, a new variable y_{max} (resp. y_{min}) is introduced in an intermediate set of constraints D'_S. More precisely, D'_S is the set of all concrete predicates $P(\alpha_1, \alpha_2)$ in S_Σ where each occurrence of $\mathsf{max}(Y)$ is replaced by y_{max}, and each occurrence of $\mathsf{min}(Y)$ is replaced by y_{min}. Then D_S is obtained from D'_S by replacing constraints by appropriate (in)equalities and adding axioms to capture the interaction between $\mathsf{min}(Y), \mathsf{max}(Y)$ and $z:Y$, i.e.,

$$\begin{aligned} D_S := \; & \{y_{\mathsf{min}} \leq y_{\mathsf{max}} \mid y_{\mathsf{min}} \text{ or } y_{\mathsf{max}} \text{ occurs in } D'_S\} \cup \\ & \{y_{\mathsf{min}} \leq z \mid y_{\mathsf{min}} \text{ occurs in } D'_S \text{ and } (z:Y) \in S\} \cup \\ & \{y_{\mathsf{max}} \geq z \mid y_{\mathsf{max}} \text{ occurs in } D'_S \text{ and } (z:Y) \in S\} \cup \\ & \{x \bowtie y \mid P_\bowtie \in \{P_\leq, P_\geq, P_>, P_<, P_=\} \text{ and } P_\bowtie(x,y) \in D'_S\} \cup \\ & \{x \bowtie n \mid P_{\bowtie_n} \in \{P_{\leq_n}, P_{\geq_n}, P_{>_n}, P_{<_n}, P_{=_n}\} \text{ and } P_{\bowtie_n}(x) \in D'_S\} \end{aligned}$$

Claim 23 *D_S is satisfiable iff S is Σ-consistent.*

The only constraints imposed on $\mathsf{min}(Y)$ (resp. $\mathsf{max}(Y)$) is that $\mathsf{min}(Y)$ is less than or equal to (resp. $\mathsf{max}(Y)$ is greater than or equal to) each element in Y. Moreover, the only elements that are required to be in Y are those x_i with $x_i : Y \in S$, the minimum, and the maximum of Y. Each solution of S_Σ is clearly also a solution of D_S. Now suppose we have a solution of D_S where $\hat{x} \in \mathsf{dom}(\Sigma)$ is the value for each variable x in D_S. Then we can define solutions $\hat{Y}$ for multiset variables Y in S_Σ by

$$\hat{Y} := \{\{\hat{x} \mid x:Y \in S\}\} \cup \{\{\hat{y}_{\mathsf{min}}, \hat{y}_{\mathsf{max}}\}\},$$

which clearly yields finite multisets. Since we started from a solution of D_S, this solution satisfies all predicate restrictions in S. Furthermore, the solution satisfies $\mathsf{max}(\hat{Y}) = \hat{y}_{\mathsf{max}}$ and $\mathsf{min}(\hat{Y}) = \hat{y}_{\mathsf{min}}$. By definition, this solution also satisfies the multiset inclusion conjuncts in S_Σ. $\square$

This axiomatisation of the behaviour of aggregation functions can also be extended to count.

Lemma 24 *If Σ is a concrete domain such that*

- $\mathsf{dom}(\Sigma)$ *is the set of non-negative integers, integers, or rational numbers,*
- $\mathsf{pred}(\Sigma) = \{P_<, P_\geq, P_>, P_\leq, P_=\} \cup \bigcup_{n \in \mathsf{dom}(\Sigma)} \{P_{\leq_n}, P_{\geq_n}, P_{>_n}, P_{<_n}, P_{=_n}\}$, *and*
- $\mathsf{agg}(\Sigma) = \{\mathsf{min}, \mathsf{max}, \mathsf{count}\}$,

then Σ*-consistency is decidable.*

PROOF. The decision procedure is similar to the one given in the proof of Lemma 22, with the only difference that, in addition, aggregated multiset variables involving count are also replaced by appropriate individual variables y_{count}, and that the behaviour of count is axiomatised. To this purpose, Boolean combinations of linear inequalities $D_{y_{\mathsf{count}}}$ are added to D_S for each aggregated multiset variable $\mathsf{count}(Y)$ occurring in S.

More precisely, given a constraint system S and a concrete domain Σ as described in Lemma 24, D'_S contains all conjuncts in S_Σ, where each occurrence of $\mathsf{max}(Y)$ is replaced by y_{max}, each occurrence of $\mathsf{min}(Y)$ by y_{min}, and each occurrence of $\mathsf{count}(Y)$ by y_{count}. Then $D^\#_S$ is defined as follows:

$$D^\#_S := D_S \cup \bigcup_{y_{\mathsf{count}} \text{ occurs in } D'_S} D_{y_{\mathsf{count}}},$$

where D_S is defined as in the proof of Lemma 22 (now with the additional variables y_{count}), and $D_{y_{\mathsf{count}}}$ is defined as follows. For better readability, we use x_Y as a shorthand for those concrete variables known to belong to Y, i.e., $x_Y := \{x \in \tau_\Sigma \mid x : Y \in S\}$, and we use $\#x_Y$ for the cardinality of x_Y.

$$\begin{aligned} D_{y_{\mathsf{count}}} := \Big(& \Big(\#x_Y = y_{\mathsf{count}} \wedge \textstyle\bigvee_{x \in x_Y} x = y_{\mathsf{min}} \wedge \bigvee_{x \in x_Y} x = y_{\mathsf{max}}\Big) \vee \\ & \Big(\#x_Y = y_{\mathsf{count}} - 1 \wedge \textstyle\bigvee_{x \in x_Y} (x = y_{\mathsf{min}} \vee x = y_{\mathsf{max}})\Big) \vee \\ & (\#x_Y \leq y_{\mathsf{count}} - 2)\Big) \wedge \\ & y_{\mathsf{count}} \in \mathbb{Z} \wedge y_{\mathsf{count}} \geq 0 \end{aligned}$$

The disjunction is necessary because we have to distinguish between the case where some of the concrete variables known to belong to a multiset coincide with its minimum and/or maximum (in which case the cardinality can be equal to $\#x_Y$, resp. $\#x_Y + 1$), and the case where both the minimum and the maximum are distinct from values for concrete variables in x_Y. This distinction is necessary, for example, to capture that

$$x : Y \wedge =_4(x) \wedge \geq_6(\mathsf{max}(Y)) \wedge \leq_2(\mathsf{min}(Y))$$

implies that the cardinality of a solution for Y is greater than or equal to $\#x_Y + 2 = 3$.

Transforming $D_{y_{\mathsf{count}}}$ into disjunctive normal form, satisfiability of $D_S^{\#}$ can be decided by testing separately each disjunct together with the (in)equalities stemming from D_S. Thus we only need to decide satisfiability of a set of (in)equalities which, again, can be decided using linear, integer, or mixed programming techniques; see, for example, [30]. $\square$

Taking the results of this section together with the results of Section 5.1 and 5.2, we obtain the following decidability result.

Corollary 25 *If Σ is a concrete domain such that*

- $\mathsf{dom}(\Sigma)$ *is the set of non-negative integers, integers, or rational numbers,*
- $\mathsf{pred}(\Sigma) = \{P_<, P_\geq, P_>, P_\leq, P_=\} \cup \bigcup_{n \in \mathsf{dom}(\Sigma)} \{P_{\leq_n}, P_{\geq_n}, P_{>_n}, P_{<_n}, P_{=_n}\}$, *and*
- $\mathsf{agg}(\Sigma) = \{\mathsf{min}, \mathsf{max}, \mathsf{count}\}$,

then satisfiability of $\mathcal{EL}(\Sigma)$-concepts as well as satisfiability and subsumption of $\mathcal{ALC}(\Sigma)^-$-concepts is decidable.

5.4 Decidability of $\mathcal{ALC}(\mathcal{D}^{\mathsf{max}}_{\mathsf{min}})$

Finally, we present our last decidability result, namely the one for subsumption and satisfiability of $\mathcal{ALC}(\Sigma)$ for certain concrete domains Σ involving only the aggregation functions min, and max.

Theorem 26 *If Σ is a concrete domain such that*

- Σ *is admissible,*
- $\mathsf{pred}(\Sigma)$ *contains a binary relation symbol $P_=$ for equality in Σ, and a binary relation symbol $P_\leq$ for a linear ordering on $\mathsf{dom}(\Sigma)$, and*
- $\mathsf{agg}(\Sigma) = \{\mathsf{min}, \mathsf{max}\}$,

then satisfiability and subsumption of $\mathcal{ALC}(\Sigma)$-concepts is decidable.

We suppose that min and max have the standard semantics as defined in Remark 9 for $\leq = P_\leq$.

PROOF. In the following, a concrete domain as described in the preconditions of Theorem 26 is called $\mathcal{D}^{\mathsf{max}}_{\mathsf{min}}$. One possibility to prove Theorem 26 would be to further modify the tableau algorithm from Section 5.2. However, there is a shorter proof, namely by a translation to $\mathcal{ALCP}(\mathcal{D})$, a natural extension of $\mathcal{ALC}(\mathcal{D})$ introduced in [20]. More precisely, each $\mathcal{ALC}(\mathcal{D}^{\mathsf{max}}_{\mathsf{min}})$-concept D can be translated into an $\mathcal{ALCP}(\mathcal{D})$-concept $\phi(D)$ such that D is satisfiable

iff $\phi(D)$ is satisfiable. In [20], satisfiability of $\mathcal{ALCP}(\mathcal{D})$-concepts was shown to be decidable, provided that $\mathcal{D}$ is admissible. Thus admissibility of $\mathcal{D}^{\mathsf{max}}_{\mathsf{min}}$ also implies decidability of the satisfiability of $\mathcal{ALC}(\mathcal{D}^{\mathsf{max}}_{\mathsf{min}})$-concepts. Moreover, $\mathcal{ALC}(\mathcal{D}^{\mathsf{max}}_{\mathsf{min}})$ is closed under negation, hence subsumption can be reduced to (un)satisfiability. We start by introducing $\mathcal{ALCP}(\mathcal{D})$.

Definition 27 (Syntax and semantics of $\mathcal{ALCP}(\mathcal{D})$) *If, for $1 \leq i \leq m$, R_i is a role or a feature name, then $u = R_1 \dots R_m$ is called* role/feature chain. *For a role/feature chain u and $a \in \Delta^{\mathcal{I}}$, $b \in \Delta^{\mathcal{I}} \cup \mathsf{dom}(\Sigma)$, we have $(a,b) \in u^{\mathcal{I}}$ iff there are $a_1, \dots, a_{m-1}$ with*

$$(a, a_1) \in R_1^{\mathcal{I}}, (a_{m-1}, b) \in R_m^{\mathcal{I}}, \text{ and } (a_i, a_{i+1}) \in R_{i+1}^{\mathcal{I}} \text{ for all } 1 \leq i \leq m-2,$$

where, for a feature name f, $(w,z) \in f^{\mathcal{I}}$ iff $f^{\mathcal{I}}(w) = z$.

$\mathcal{ALCP}(\mathcal{D})$ is obtained from $\mathcal{ALC}(\mathcal{D})$ by adding concepts of the form

$$\forall u_1, \dots, u_n.P \quad \textit{(generalised value restriction) and}$$
$$\exists u_1, \dots, u_n.P \quad \textit{(generalised exists restriction).}$$

where P is a concrete predicate of arity n and $u_1, \dots, u_n$ are role/feature chains.

An $\mathcal{ALCP}(\mathcal{D})$ interpretation must satisfy, additionally,

$$(\forall u_1, \dots, u_n.P)^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} \mid \textit{For all } y_1, \dots, y_n \textit{ with } (x, y_i) \in u_i^{\mathcal{I}} \textit{ for all } 1 \leq i \leq n, \textit{ we have } (y_1, \dots, y_n) \in P^{\Sigma}\},$$
$$(\exists u_1, \dots, u_n.P)^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} \mid \textit{For all } 1 \leq i \leq n \textit{ there is } y_i \textit{ with } (x, y_i) \in u_i^{\mathcal{I}} \textit{ and } (y_1, \dots, y_n) \in P^{\Sigma}\}.$$

For pure feature chains $u_1, \dots, u_n$, the concept $\exists u_1, \dots, u_n.P$ is, by definition, equivalent to $P(u_1, \dots, u_n)$.

The idea of the translation from $\mathcal{ALC}(\mathcal{D}^{\mathsf{max}}_{\mathsf{min}})$ into $\mathcal{ALCP}(\mathcal{D})$ is to introduce new feature names $f_{\mathsf{min}(R \circ f)}$ and $f_{\mathsf{max}(R \circ f)}$ and to use the new generalised restrictions to make sure that $f_{\mathsf{min}(R \circ f)}(x)$ coincides with the minimum of x's $R \circ f$-successors.

The translation ϕ from $\mathcal{ALC}(\mathcal{D}^{\mathsf{max}}_{\mathsf{min}})$ to $\mathcal{ALCP}(\mathcal{D})$ is defined inductively on the structure of concepts and trivial for all concept forming operators (the exact definition is given below); the only changes it makes are for aggregated features: Whenever features of the form $f_1 \dots f_k \mathsf{min}(R \circ f)$ (resp. $f_1 \dots f_k \mathsf{max}(R \circ f)$) occur, new feature names $f_{\mathsf{min}(R \circ f)}$ (resp. $f_{\mathsf{max}(R \circ f)}$) are introduced. Then

these aggregated features are replaced by feature chains $f_1 \ldots f_k f_{\mathsf{min}(R \circ f)}$ (resp. $f_1 \ldots f_k f_{\mathsf{max}(R \circ f)}$). Finally, we make sure that the $f_1 \ldots f_k f_{\mathsf{min}(R \circ f)}$-successor is the minimum of all $f_1 \ldots f_k Rf$-successors. For this, we add concepts of the form [6]

$$(\exists\,(f_1 \ldots f_k Rf),\,(f_1 \ldots f_k f_{\mathsf{min}(R \circ f)}).\,P_{=}) \sqcap$$
$$(\neg\exists\,(f_1 \ldots f_k f),\,(f_1 \ldots f_k Rf_{\mathsf{min}(R \circ f)}).\,P_{<}).$$

The first conjunct makes sure that the $f_1 \ldots f_k f_{\mathsf{min}(R \circ f)}$-successor (exists and) coincides with one of the $f_1 \ldots f_k Rf$-successors. The second conjunct ensures that none of the $f_1 \ldots f_k Rf$-successors is smaller than the $f_1 \ldots f_k f_{\mathsf{min}(R \circ f)}$-successor. For max, we add similar concepts. Please note that we cannot replace the negated existential quantifier by a universal one since, in $\mathcal{ALCP}(\mathcal{D})$, the universal one quantifies over *all* role-successors, and not only over those in the concrete domain. Thus using the universal quantifier would require all $f_1 \ldots f_k Rf$-successors to be in the concrete domain—in contrast to the semantics of $\mathcal{ALC}(\Sigma)$. More precisely, ϕ is defined as follows:

$$\phi(C \sqcap D) = \phi(C) \sqcap \phi(D), \qquad \phi(C \sqcup D) = \phi(C) \sqcup \phi(D)$$
$$\phi(\exists R.C) \;= \exists R.\phi(C), \qquad \phi(\forall R.C) \;= \forall R.\phi(C)$$
$$\phi(P(u_1, \ldots, u_n)) = \exists \phi(u_1), \ldots, \phi(u_n).P \sqcap \bigsqcap_{1 \le i \le n} \psi(u_i),$$

where, for a concrete feature u and $\Gamma \in \{\mathsf{min}, \mathsf{max}\}$

$$\phi(u) = \begin{cases} u & \text{if } u \text{ is a feature chain} \\ f_1 \ldots f_k f_{\Gamma(R \circ f)} & \text{if } u = f_1 \ldots f_k\, \Gamma(R \circ f) \end{cases}$$

$$\psi(u) = \begin{cases} \top_A & \text{if } u \text{ is a feature chain} \\ \exists(f_1 \ldots f_k Rf), (f_1 \ldots f_k f_{\mathsf{max}(R \circ f)}).P_{=} \sqcap \\ \neg\exists(f_1 \ldots f_k Rf), (f_1 \ldots f_k f_{\mathsf{max}(R \circ f)}).P_{>} & \text{if } u = f_1 \ldots f_k \mathsf{max}(R \circ f) \\ \exists(f_1 \ldots f_k Rf), (f_1 \ldots f_k f_{\mathsf{min}(R \circ f)}).P_{=} \sqcap \\ \neg\exists(f_1 \ldots f_k f), (f_1 \ldots f_k Rf_{\mathsf{min}(R \circ f)}).P_{<} & \text{if } u = f_1 \ldots f_k \mathsf{min}(R \circ f) \end{cases}$$

By construction, each model of an $\mathcal{ALC}(\mathcal{D}^{\mathsf{max}}_{\mathsf{min}})$-concept D can be transformed into a model of $\phi(D)$ by defining $f^{\mathcal{I}}_{\Gamma(R \circ f)}(x) := \Gamma(R \circ f)^{\mathcal{I}}(x)$ for $\Gamma \in \{\mathsf{min}, \mathsf{max}\}$. Vice versa, the correctness proof in [20] implies that $\mathcal{ALCP}(\mathcal{D})$ has the finite model property. Hence each satisfiable $\mathcal{ALCP}(\mathcal{D})$-concept $\phi(D)$ has a finite model $\mathcal{I}$ which is, by construction, also a model of D. $\square$

[6] We use $\exists u, v.P_{<}$ as an abbreviation for $\exists v, u.\bar{P}_{\le}$, and $\exists u, v.P_{>}$ as an abbreviation for $\exists u, v.\bar{P}_{\le}$.

Intuitively, the reason for decidability of $\mathcal{ALC}(\mathcal{D}^{\mathsf{max}}_{\mathsf{min}})$ can be seen in the fact that $\mathsf{min}, \mathsf{max}$ only depend on the "boundaries" of a multiset and not on its "inside"—in contrast to all other standard aggregation functions such as sum or count.

6 Related Work and Conclusion

Reasoning with constraints involving aggregation functions is a crucial task for many advanced information systems like decision support and on-line-analytical processing systems, data warehouses, and (statistical) databases [34,18,28,14,26,36]. The more the amount of data that are processed by these systems grows, the more important aggregation functions become for summarising, consolidating, and analysing these large amounts of data. Hence, traditional techniques for query rewriting, query optimisation, view maintenance, and intensional reasoning must be extended such that they are able to cope with aggregation functions. Since Description Logics have been proved useful for these tasks, we have extended them with aggregation functions and investigated the effect of this extension on the decidability of the subsumption and the satisfiability problem.

The two undecidability results presented in this paper indicate that this task will be difficult. The aggregation functions min, max, and sum that suffice to obtain undecidability are among the "well-behaved" ones: aggregation functions like $\mathsf{average}$ are much more difficult to handle. For example, min and max are multiple-invariant (i.e., the multiplicity of an element of the multiset does not matter), and min, max, and sum are monotonic—in contrast to $\mathsf{average}$. Furthermore, min, max, and sum are distributive, i.e., for an aggregation function $\mathsf{agg} \in \{\mathsf{min}, \mathsf{max}, \mathsf{sum}\}$ and two disjoint multisets M, M', $\mathsf{agg}(M \cup M')$ can be computed using $\mathsf{agg}(M)$ and $\mathsf{agg}(M')$ only—in contrast to $\mathsf{average}$. Hence, our undecidability result cannot be said to be caused by using too powerful aggregation functions.

Arguing from another perspective, extending $\mathcal{ALC}(\mathcal{D})$ with aggregation functions yields a rather expressive family of Description Logics, and thus it might not be very surprising that a variety of these Description Logics is undecidable. In contrast, $\mathcal{FL}_0$ is, to our knowledge, the weakest Description Logic ever considered and thus the undecidability result of $\mathcal{FL}_0(\Sigma)$ with $\mathsf{min}, \mathsf{max}$, and sum only is rather surprising.

In [28], the expressive power of Datalog with constants, built-in predicates for comparisons (with constants), and aggregation functions is investigated. The undecidability results described there are orthogonal to those presented here since (1) our pre-requisites are weaker and (2) in contrast to Datalog, the

Description Logics described here do not provide any recursion mechanisms. For (1), for example, the results described in [28] concern fixed domains such as the non-negative integers, whereas our results involve domains *containing* the non-negative integers.

In [34], the authors investigate the complexity of the satisfiability problem of *aggregation constraints*, i.e., sets of equations over aggregated multiset variables and element variables. Besides some decidability results (with exact bounds), some undecidability results are presented. These do not imply those described here since in [34], all undecidability results either involve the aggregation functions sum *and* count, and possibly average.

Investigating these undecidability results more closely, we identify two sources of this complexity: the aggregation function sum and the interaction between (local) universal quantification in concepts of the form $\forall R.C$ and aggregation functions. Indeed, the decidability result for $\mathcal{ALC}(\mathcal{D}^{\mathsf{max}}_{\mathsf{min}})$ shows that min and max alone are far less expressive than in combination with sum—which is not too surprising. To obtain a generic decidability result, we further restricted the underlying Description Logic to $\mathcal{EL}$, presented a tableau algorithm that decides satisfiability of $\mathcal{EL}(\Sigma)$-concepts, and finally showed how this algorithm can be extended to decide satisfiability of $\mathcal{ALC}(\Sigma)^-$-concepts. The logic $\mathcal{ALC}(\Sigma)^-$ was designed such that the complex interaction between universal value restrictions and aggregation functions mentioned above do not arise. By construction, $\mathcal{ALC}(\Sigma)^-$ is closed under negation, and thus the tableau algorithm can also be used to decide subsumption of $\mathcal{ALC}(\Sigma)^-$-concepts.

This tableau algorithm is parameterised by a decision procedure for satisfiability of certain conjunctions of concrete predicates involving aggregation functions, i.e., Σ-consistency. Hence any concrete domain for which Σ-consistency is decidable can be used to form a logic $\mathcal{ALC}(\Sigma)^-$ for which intensional reasoning is decidable—provided that the negation normal form for concepts is adapted accordingly. In this paper, we showed that the (non-negative) integers or rational numbers with comparisons (possibly with constants) and aggregation functions min, max, and count are among those decidable concrete domains. However, we did not exhaustively classify all "standard" concrete domains, but believe that it is interesting to find other expressive concrete domains with aggregation functions for which Σ-consistency is decidable. For example, it would be interesting to see the consequence of replacing the aggregation function count in Lemma 22 by sum. It should be noted that *adding* sum to the concrete domain considered in the lemma makes Σ-consistency undecidable. This is as an easy consequence of one of Theorem 3.1 and Corollary 3.1 in [34].

These decidability results are orthogonal to the decidability results in [31] for containment of conjunctive queries with aggregation in the query head: we

have less powerful aggregation functions, but allow to use them in a more complex way. More precisely, we allow to build concepts in which aggregation occurs at various levels in nested concepts. The exact connection between our decidable DLs $\mathcal{EL}(\Sigma)$ and $\mathcal{ALC}(\Sigma)^-$ and conjunctive queries with aggregation is a topic for future research.

Finally, we would like to point out that, in the presence of aggregation functions and for data warehouse, OLAP, and similar applications, another inference problem plays a crucial role, which is unrelated to logical standard inference problems such as satisfiability and subsumption, namely *summarisability* [25]. Assume you have already summarised some base data up to a certain level of granularity using certain aggregation functions. Next, the same base data needs to be summarised again up to (possibly) a different level of granularity and (possibly) using different aggregation functions. In case this second summary can be computed from the first one, this fact can be exploited since (a) the summarised data is probably smaller and (b) the base data might no longer be available. Thus deciding this question of "what can be computed from what" can help in semantic query optimisation, and hence is subject to a variety of investigations. Various formalisms have been introduced that allow to specify *how* data can be summarised, i.e., formalisms to specify *dimensions* along which data can be summarised, and investigated w.r.t. the complexity of summarisability, see e.g., [23,33,24]. These formalisms vary w.r.t. their expressive power, and allow, roughly speaking, to populate a given, partially ordered, finite set of *categories*. For example, city, province, and state are categories that can be populated with Toronto, Alberta, and Canada, respectively. The partial order on the categories is then "transferred" in a appropriately restricted form to the instances. In [23], it is shown that, for distributive aggregation functions such as min, max, count, and sum and a given population of categories (i.e., a given model), it is co-NP-complete to decide whether the summary up to a certain category can be computed from other summaries to other categories . Please note that this result is restricted to summarisation along a single dimension and w.r.t. a single aggregation function.

Due to the tree model property of most description logics and the DAG-like structure of dimensions, the above mentioned frameworks cannot be directly mapped into description logics. Moreover, the standard description logic inference problems take into account *all* interpretations or *all* models of a knowledge base. In contrast, summarisability in the above mentioned frameworks takes into account a single one. To the best of our knowledge, there is no useful notion of summarisability in description logics. The introduction of this inference problem and the investigation of its complexity will be part of future investigations.

References

[1] F. Baader. Using automata theory for characterizing the semantics of terminological cycles. *Annals of Mathematics and Artificial Intelligence*, 18(2–4):175–219, 1996.

[2] F. Baader and P. Hanschke. A schema for integrating concrete domains into concept languages. In *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence (IJCAI-91)*, pages 452–457, Sydney, 1991. A long version appeared in the technical report RR-91-10, DFKI, Germany, 1991.

[3] F. Baader and U. Sattler Description Logics with Concrete Domains and Aggregation. In *Proceedings of the 13th European Conference on Artificial Intelligence (ECAI-98)*, pages 336-340. John Wiley & Sons Ltd, 1998.

[4] R. J. Brachman, D. McGuinness, P. Patel-Schneider, L. Resnick, and A. Borgida. Living with CLASSIC: When and how to use a KL-ONE-like language. In John F. Sowa, editor, *Principles of Semantic Networks*. Morgan Kaufmann, Los Altos, 1991.

[5] M. Buchheit, M. A. Jeusfeld, W. Nutt, and M. Staudt. Subsumption of queries over object-oriented databases. *Information Systems*, 19(1), 1994.

[6] D. Calvanese. Finite Model Reasoning in Description Logics. *Proceedings of the Fifth International Conference on the Principles of Knowledge Representation and Reasoning (KR-96)*, 1996. Morgan Kaufmann, Los Altos.

[7] D. Calvanese, G. De Giacomo, and M. Lenzerini. Conjunctive query containment in description logics with n-ary relations. In M.-C. Rousset, R. Brachmann, F. Donini, E. Franconi, I. Horrocks, and A. Levy, editors, *Proceedings of the International Workshop on Description Logics*, Gif sur Yvette, France, 1997. Université Paris-Sud.

[8] D. Calvanese, G. De Giacomo, and M. Lenzerini. On the decidability of query containment under constraints. In *Proc. of the 17th ACM SIGACT SIGMOD SIGART Sym. on Principles of Database Systems (PODS'98)*, pages 149–158, 1998.

[9] D. Calvanese, G. De Giacomo, and R. Rosati. Data integration and reconciliation in data warehousing: Conceptual modeling and reasoning support. *Network and Information Systems*, 4(2), 1999.

[10] D. Calvanese, M. Lenzerini, and D. Nardi. Description logics for conceptual data modeling. In Jan Chomicki and Gnter Saake, editors, *Logics for Databases and Information Systems*, pages 229–263. Kluwer Academic Publisher, 1998.

[11] D. Calvanese, M. Lenzerini, and D. Nardi. A unified framework for class based representation formalisms. In J. Doyle, E. Sandewall, and P. Torasso, editors, *Proceedings of the Fourth International Conference on the Principles of Knowledge Representation and Reasoning (KR-94)*, pages 109–120, Bonn, 1994. Morgan Kaufmann, Los Altos.

[12] T. Catarci, G. D'Angiolini, and M. Lenzerini. Conceptual language for statistical data modeling. *Data and Knowledge Engineering*, 17(2):93–125, 1995.

[13] M. Davis. Hilbert's tenth problem is unsolvable. *American Mathematical Monthly*, 80:233–269, 1973.

[14] G. De Giacomo and P. Naggar. Conceptual data model with structured objects for statistical databases. In *Proceedings of the Eighth International Conference on Statistical Database Management Systems (SSDBM'96)*, pages 168–175. IEEE Computer Society Press, 1996.

[15] F. Donini, M. Lenzerini, D. Nardi, and W. Nutt. The complexity of concept languages. In *Proceedings of the Second International Conference on the Principles of Knowledge Representation and Reasoning (KR-91)*, Boston, MA, USA, 1991.

[16] E. Franconi and G. Ng. The i.com tool for intelligent conceptual modelling. In *Working Notes of the ECAI2000 Workshop on Knowledge Representation Meets Databases (KRDB2000)*, 2000.

[17] E. Franconi and U. Sattler. A data warehouse conceptual data model for multidimensional aggregation: a preliminary report. *Italian Association for Artificial Intelligence AI*IA Notizie*, 1:9–21, 1999.

[18] A. Gupta, V. Harinarayan, and D. Quass. Aggregate-query processing in data warehousing environments. In *Proceedings of the 21. International Conference on Very Large Data Bases (VLDB-95)*, 1995.

[19] V. Haarslev and R. Möller. Expressive abox reasoning with number restrictions, role hierarchies, and transitively closed roles. In *Proceedings of the Seventh International Conference on the Principles of Knowledge Representation and Reasoning (KR-00)*, 2000.

[20] P. Hanschke. Specifying Role Interaction in Concept Languages. In *Proceedings of the Second International Conference on the Principles of Knowledge Representation and Reasoning (KR-91)*, 1992.

[21] B. Hollunder, W. Nutt, and M. Schmidt-Schauss. Subsumption algorithms for concept description languages. In *Proceedings of the European Conference on Artificial Intelligence (ECAI-90)*, Pitman Publishing, London, 1990.

[22] I. Horrocks. Using an Expressive Description Logic: FaCT or Fiction? In *Proceedings of the Sixth International Conference on the Principles of Knowledge Representation and Reasoning (KR-98)*, 1998.

[23] C. Hurtado and A. Mendelzon. OLAP Dimension Constraints. In *Proceedings of the ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS-02)*, ACM, 2002.

[24] W. Lehner, H. Albrecht, and H. Wedekind. Multidimensional normal forms. In *Proceedings of the Tenth International Conference on Statistical Database Management Systems (SSDBM'98)*, IEEE Computer Society, 1998.

[25] H. Lenz and A. Shoshani. Summarizability in OLAP and statistical databases. In *Proceedings of the Nineth International Conference on Statistical Database Management Systems (SSDBM'97)*, IEEE Computer Society, 1998.

[26] A. Y. Levy and I. S. Mumick. Reasoning with aggregation constraints. In *Proceedings of the International Conference on Extending Database Technology (EDBT-96)*, Avignon, France, 1996.

[27] C. Lutz. Reasoning with concrete domains. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI-99)*. Morgan Kaufmann, Los Altos, 1999.

[28] I. S. Mumick and O. Shmueli. How expressive is stratified aggregation. *Annals of Mathematics and Artificial Intelligence*, 15(3-4), 1995.

[29] B. Nebel. *Reasoning and Revision in Hybrid Representation Systems*. Lecture Notes in Artificial Intelligence. Springer-Verlag, 1990.

[30] G. L. Nemhauser and L. A. Wolsey *Integer and Combinatorial Optimization*. John Wiley & Sons, 1988.

[31] W. Nutt, Y. Sagiv, and S. Shurin. Deciding equivalences among aggregate queries. In *Proceedings of the Seventeenth ACM SIGACT SIGMOD Symposium on Principles of Database Systems (PODS-98)*, 1998.

[32] P. F. Patel-Schneider and I. Horrocks. DLP and FaCT. In *Proceedings of the International Conference on Automated Reasoning with Analytic Tableaux and Related Methods (TABLEAUX-99)*, volume 1397 of *Lecture Notes in Artificial Intelligence*, pages 19–23. Springer-Verlag, 1999.

[33] T. Pedersen and C. Jensen. Multidimensional data modeling for complex data. In *Proceedings of the 15th IEEE International Conference on Data Engineering (ICDE'99)*, IEEE Computer Science, 1999.

[34] K. A. Ross, D. Srivastava, P. J. Stuckey, and S. Sudarshan. Foundations of aggregation constraints. *Theoretical Computer Science*, 193(1–2):149–179, 1998.

[35] M. Schmidt-Schauß and G. Smolka. Attributive concept descriptions with complements. *Artificial Intelligence*, 48(1):1–26, 1991.

[36] D. Srivastava, S. Dar, H. V. Jagadish, and A. Y. Levy. Answering queries with aggregation using views. In *Proceedings of the 22. International Conference on Very Large Data Bases (VLDB-96)*, Bombay, India, 1996.