# $\text{OCL}_{\text{UNIV}}$: Expressive UML/OCL Conceptual Schemas for Finite Reasoning

Xavier Oriol and Ernest Teniente

Universitat Politècnica de Catalunya, Barcelona, Spain
{xoriol,teniente}@essi.upc.edu

**Abstract.** Full UML/OCL is so expressive that most reasoning tasks are known to be undecidable in schemas defined with these languages. To tackle this situation, literature has proposed mainly three decidable fragments of UML/OCL: UML with no OCL, UML with limited OCL and no maximum cardinality constraints (OCL-Lite), and UML with limited OCL with no minimum cardinality constraints ($\text{OCL}_{\text{UNIV}}$). Since most conceptual schemas make use of OCL together with min and max cardinalities, this poses a strong limitation to current proposals. In this paper, we go beyond these limits by showing that $\text{OCL}_{\text{UNIV}}$ with *acyclic* min cardinality constraints and path acyclicity constraints also preserves decidability. In this way, we establish a language that can deal with most of UML/OCL identified constraint patterns. We also empirically test the expressiveness of this language through different UML/OCL case studies.

**Keywords:** UML, OCL, Decidability, Reasoning

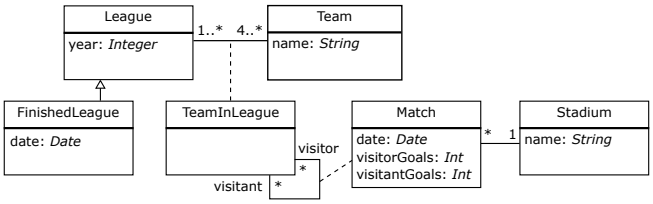
## 1 Introduction

Reasoning on UML/OCL conceptual schemas is aimed at answering questions regarding what kind of instances does a UML/OCL conceptual schema admit. This is known to be crucial in the specification stage of software development. Indeed, reasoning about what kind of instances does a UML/OCL schema admit allows to assess whether the UML/OCL schema is correct or not. In this way, we can avoid the propagation of conceptual errors to the other stages of software development [1].

For instance, consider the UML/OCL schema in Figure 1 specifying a soccer league competition. This domain includes *Leagues*, identified by year, and *Teams* enrolled in these leagues. Teams play *Matches* during a league, for which we store the goals made and the *Stadium* in which they took place. The UML schema is complemented with some OCL constraints that describe the *primary key attributes* of each class, ensure that no team has a match with itself, ensure that a league is *finished* when all teams have played against all other teams, and ensure that the unique unfinished league is the last one, whose finishing date is later than any date of its matches.

By taking a closer look at this UML/OCL schema, we may realize that it accepts an instance of a match among two teams of different leagues. This clearly indicates that there is an error in the UML/OCL schema and, thus, an OCL constraint preventing such situation should be incorporated to it. It is worth noting that, incorporating new constraints in the conceptual schema means propagating such changes into the other

Oriol, X., Teniente, E. OCLuniv: Expressive UML/OCL conceptual schemas for finite reasoning. A: International Conference on Conceptual Modeling. "Conceptual Modeling: 36th International Conference, ER 2017: Valencia, Spain, November 6–9, 2017: proceedings". València: Springer, 2017, p. 354-369.
The final publication is available at https://link.springer.com/chapter/10.1007%2F978-3-319-69904-2_28




```
context League inv LeagueID: League.allInstances()->isUnique(year)
context Team inv TeamID: Team.allInstances()->isUnique(name)
context Stadium inv StadiumID: Stadium.allInstances()->isUnique(name)
context Match inv NoSelfMattch: self.visitor <> self.visitant
context FinishedLeague inv EveryMatchPlayed:
     self.teamInLeague->forAll(t1,t2|t1.visitor->includes(t2)))
context FinishedLeague inv OnlyLastOneIsNotFinished:
     League.allInstances()->forAll(l|l=self or l.year < self.year)
context FinishedLeague inv FinishesAfterAllMatches:
     self.teamInLeague.match[visitor]->forAll(m|m.date < self.date)
```

**Fig. 1.** UML schema and OCL constraints

software artifacts (in particular, operation design-contracts and code), thus, it is crucial to reason that no constraint is missing/lacking in the conceptual schema in order to avoid the propagation of this mistake to the rest of artifacts.

Unfortunately, it is well-known that reasoning on UML/OCL schemas is undecidable [2]. That is, there is no algorithm that can reason over a UML/OCL schema ensuring termination and a correct output. The cause of the undecidability relies on the high expressiveness of UML/OCL schemas. Indeed, UML schemas with general OCL constraints have an expressive power beyond first-order logics. Thus, since reasoning about the satisfiability of first-order logic theories is undecidable and a UML schema with general OCL constraints can encode a first-order theory, the undecidability result of first-order logic reasoning is inherited by reasoning on UML/OCL schemas.

To mitigate this issue, one option is to reduce the expressiveness of UML/OCL into some fragment whose reasoning problems are decidable. To the best of our knowledge, three such fragments have been identified in the literature:

- UML schemas alone: that is full UML features with no OCL constraints [2].
- OCL-Lite: a fragment of UML/OCL which, in essence, forbids the usage of maximum cardinalities in the UML schema [3].
- $\text{OCL}_{\text{UNIV}}$: a fragment of UML/OCL which, in essence, forbids the usage of minimum cardinalities in the UML schema [4].

Clearly, all three proposed languages lack critical features making them strongly limited. Indeed, UML/OCL schemas tend to make use of OCL constraints and minimum and maximum cardinality constraints together, as required by our running example. Thus, although all such fragments are decidable, none of them is expressive enough for actual UML/OCL schemas.

Hence, the main goal of this paper is to identify an OCL subset that, combined with min and max cardinality constraints together, preserves decidability. In particualr, we prove that $\text{OCL}_{\text{UNIV}}$ combined with minimum cardinality constraints is still decidable, provided that these constraints do not form a cycle in the UML class diagram. This

decidability result is also preserved when extending OCL$_{\mathsf{UNIV}}$ with path *acyclicity* constraints (such as *no person can be his own ancestor*). As a result, we have that this extended language can deal with the constraint patterns most frequently used (as defined in [5]). We keep the name OCL$_{\mathsf{UNIV}}$ since the fragment of OCL handled by this new language is the same as the original OCL$_{\mathsf{UNIV}}$.

We also test the expressiveness of this decidable fragment by means of studying several UML/OCL case studies, and comparing how many textual constraints could we encode in the language proposed in this paper and how many in OCL-Lite. With this experiment, we show that, while OCL-Lite could cover 56% of the constraints of the case studies, our OCL$_{\mathsf{UNIV}}$ could handle the 82%. In addition, our OCL$_{\mathsf{UNIV}}$ required only deleting 1 minimum cardinality constraint to ensure decidability (i.e., acyclicity in the class diagrams associations) while OCL-Lite required removing all the maximum cardinality constraints in the schemas (a total of 69).

In conclusion, we can state that the language identified in this paper is currently the most expressive one for specifying UML/OCL schemas while ensuring finite reasoning on them. Thus, any complete UML/OCL schema reasoner (such as [6]) receiving as input a schema written in our language never hangs while checking its correctness.

## 2 Preliminaries

We start from a logic encoding of a UML class diagram into a logic schema based on [6], and the weak acyclicity result stating that any logic schema with no cycles involving existential variables can be reasoned in finite time [7, 8]. We summarize all these notions in the following:

*UML class diagram and compatible classes.* A UML class diagram is a diagram which contains a hierarchy of classes, n-ary associations among these classes (where some of them might be reified, i.e, association classes), and attributes inside the classes. In addition, a UML class diagram might be annotated with minimum/maximum cardinality constraints over its association-ends/attributes, and hierarchy constraints (that is, disjoint/complete constraints). In this paper, we say that two classes $C_1$ and $C_2$ are *compatible* if they have a common superclass $SC$ in the hierarchy.

*Terms, atoms, literals, and positions.* A *term* $t$ is either a variable or a constant. An *atom* is formed by a $n$-ary *predicate* $p$ together with $n$ terms, i.e., $p(t_1, ..., t_n)$. We may write $p(\overline{t})$ for short, and say that the position $p[i]$ is occupied by the term $t_i$. If all the terms of an atom are constants, we say that the atom is *ground*. A literal $l$ is either an atom $p(\overline{t})$, or a built-in literal $t_i\ \omega\ t_j$, where $\omega$ is an arithmetic comparison (i.e., $<$,$\leq$,$=$,$\neq$).

*Logic encoding of the UML class diagram.* We formalize each class *C* in a class diagram with attributes $\{A_1, \ldots, A_n\}$ by means of a base atom $c(Oid)$ together with $n$ atoms of the form $cA_i(Oid, A_i)$, each association *R* between classes $\{C_1, \ldots, C_k\}$ by means of a base atom $r(C_1, \ldots, C_k)$, and each association class *R* between classes $\{C_1, \ldots, C_k\}$ with attributes $\{A_1, \ldots, A_n\}$ by means of a base atom $r(C_1, \ldots, C_k)$ together with $n$ atoms $rA_i(C_1, \ldots, C_k, A_i)$.

*Dependencies.* A *Tuple-Generating Dependency (TGD)* is a formula of the form $\forall\, \overline{x}, \overline{z}.\ \varphi(\overline{x}, \overline{z}) \rightarrow \exists\, \overline{y}.\ \psi(\overline{x}, \overline{y})$ where $\varphi(\overline{x}, \overline{z})$ is a conjunction of literals, and $\psi(\overline{x}, \overline{y})$

a conjunction of atoms. A *Disjunctive Embedded Dependency (DED)* is a variation of TGDs where disjunctions are admitted in the conclusion of the rule. In particular, they follow the form: $\forall \overline{x}, \overline{z}.\ \phi(\overline{x}, \overline{z}) \rightarrow \bigvee \exists \overline{y}.\ \psi(\overline{x}, \overline{y})$. A *ded* is a *denial* if its right hand side is an empty disjunction: $\forall \overline{x}.\ \phi(\overline{x}) \rightarrow \bot$. From now on, we omit the logic quantifiers since they can be understood by context.

*Dependency graph and weak acyclicity.* Given a set of *deds*, its *dependency graph* is a directed graph obtained as follows. There is a vertex for each position of all *ded* predicates, and for each *ded* of the form $\phi(\overline{x}, \overline{y}) \rightarrow \bigvee \psi(\overline{x}, \overline{z})$, there is a *universal edge* from a position $p[i]$ of $\phi$ to a position $r[j]$ of $\psi$ iff: there is a variable $x \in \overline{x}$ occupying both $p[i]$ and $r[i]$. Moreover, there is an *existential edge* from a position $p[i]$ of $\phi$ to a position $r[j]$ of $\psi$ iff there is a variable $x \in \overline{x}$ occupying $p[i]$, and there is a variable $z \in \overline{z}$ occupying $r[i]$. A set of *deds* is said to be *weakly acyclic* iff its *dependency graph* does not contain any cycle involving an existential edge.

## 3 The OCL$_{\text{UNIV}}$ Language

OCL$_{\text{UNIV}}$ is a fragment of OCL which does not make use of existential variables (i.e., it does not have the *exists* OCL construct and it limits all the other constructs to avoid emulating it). Under the point of view of first-order logics, it is the fragment of OCL that can be described by means of the first-order constructs $\vee$, $\wedge$ and $\forall$; avoiding $\exists$ and limiting the usage of $\neg$ accordingly. The OCL$_{\text{UNIV}}$ language was firstly described as a fragment of OCL which can be efficiently checked by means of SQL queries with no need of subqueries [9], and as a fragment whose constraints can be mantained (i.e. repaired) in finite time [4].

We reproduce its grammar here for the sake of self-containment of the paper:

```
ExpBool ::= ExpBool and ExpBool     | ExpBool or ExpBool
          | ExpOp
ExpOp   ::= Path->excludesAll(Path) | Var.Member->includesAll(Path)
          | Path->excludes(Path)    | Var.Member->includes(Var)
          | Path->isEmpty()         | Path->forAll(Var| ExpBool)
          | Path OpComp Constant    | not Path.oclIsKindOf(Class)
          | Path OpComp Path        | Path.oclIsKindOf(Class)
Path    ::= Var.Nav                 | Class.allInstances().Nav
          | Var                     Class.allInstances()
Nav     ::= Role.Nav                | oclAsType(Class).Nav
          | Role                    | Attribute
          | oclAsType(Class)
```

With regard to semantics, the basic property of OCL$_{\text{UNIV}}$ is that its logic encoding provides as a result *deds* of the form: $\phi(\overline{x}, \overline{y}) \rightarrow \bigvee \psi(\overline{x})$ or denials $\phi(\overline{x}) \rightarrow \bot$ [4]. In any case, note the absence of existentials variables $\overline{z}$ of typical deds.

For instance, the OCL$_{\text{UNIV}}$ constraints in Figure 1 would be encoded as follows:

```
1) LeagueYear(l, y), LeagueYear(l2, y), l≠l2 → ⊥
2) TeamName(t, n), TeamName(t2, n), t≠t2 → ⊥
3) StadiumName(s, n), StadiumName(s2, n), s≠s2 → ⊥
4) Match(tl, tl) → ⊥
5) TeamInLeage(t, l), TeamInLeage(t2, l) → Match(t, l, t2, l)
```

```
6) FinishedLeage(f),LeagueName(l,n),l≠f,LeagueYear(f,y),LeagueYear(l,z),z<y → ⊥
7) FinishedLeage(f), LeagueDate(f, d), MatchDate(tl, l, tl2, l2, d2), d < d2 → ⊥
```

Moreover, since the OCL$_{\mathsf{UNIV}}$ language is strongly-typed, all its valid expressions satisfy the type conformance specified in the OCL standard. This fact implies that, for each variable $x$ in some *ded* codifying some OCL$_{\mathsf{UNIV}}$ constraint, the UML class of two different positions occupied by $x$ are compatible. Intuitively, this is because each variable $x$ stands for some UML instance of class $C$ obtained when evaluating an OCL$_{\mathsf{UNIV}}$ expressions, and UML instances of $C$ can only be obtained in OCL$_{\mathsf{UNIV}}$ expressions of a type compatible with $C$.

For instance, in the ded 4 above, we see that the variable *t1* holds two different positions in the *Match* predicate. This implies that both positions codify two compatible types. Indeed, if we take a look at the diagram, we see that *Match* is a recursive association and thus, both positions have the same type.

## 4 Decidability of OCL$_{\mathsf{UNIV}}$ with min cards and path acyclicity

Reasoning whether a UML/OCL schema satisfies a given property is aimed at checking whether the schema admits a consistent sample instantiation witnessing the property. For instance, checking whether the schema in our example satisfies that *Match* is *lively*, requires identifying whether this schema admits an instantiation containing, at least, one instance of *Match* without violating any integrity constraint.

The cause for undecidability is always the new objects that must be created to repair the constraints violated by the sample instantiation being built. Indeed, such new objects might violate other constraints that require creating new objects, thus, potentially stacking into an endless process of creating new objects.

For instance, assume that we have a UML class diagram with classes *Employee*, *Department*, and *Project* such that *each employee should be assigned to at least one department*, *each department should be assigned to at least one project*, and that *each project should have at least two employees*. Clearly, if we want to check whether *Employee* is *lively*, we need to instantiate one employee in the schema. Then, to satisfy the constraints, we will have to instantiate one department for this employee. Similarly, we will need to instantiate one project for the department and, then, we need a new employee for the project, thus potentially entering into an infinite loop.

Intuitively, a UML class diagram with acyclic minimum cardinality constraints avoids this kind of loops. Moreover, OCL$_{\mathsf{UNIV}}$ constraints (and path acyclicity constraints) guarantee that no new object needs to be created to satisfy the constraints. Thus, the combination of OCL$_{\mathsf{UNIV}}$ constraints with path acyclicity constraints into UML class diagrams with no minimum cardinality constraints cycles is decidable.

In the following we formalize the proof of this statement. We start by proving that reasoning on UML class diagrams with no min cardinality cycles and no OCL constraints is decidable. To facilitate the proofs, we begin with the assumption that min cardinalities are 1 or 1..*, and that all hierarchies are *complete*. Then, we show that incorporating OCL$_{\mathsf{UNIV}}$ constraints preserves decidability. Finally, we incorporate path acyclicity constraints and generalize our results to min cardinality constraints with boundaries different than 1, and incomplete hierarchies.

### 4.1 Reasoning on UML class diagrams with no min card cycles is decidable

We say that there is a *min cardinality cycle* in a UML class diagram if there is a directed cycle in the UML class diagram formed by association roles whose min cardinality is one and/or hierarchy constraints (taken upwards or downwards, i.e., in both directions of the hierarchy). Then, we say that a UML class diagram is weakly acyclic if it does not contain any min cardinality cycle. Formally:

**Definition 1.** *A* min cardinality cycle *on class $C$ is a sequence of association-role names $r_0$, ..., $r_n$ s.t.:*

1. *Forms a cycle, i.e., the UML class of $r_n$ is compatible with $C$.*
2. *Each association-role $r_i$ has a min cardinality 1 (or is a navigation from an association class to one of its members).*
3. *It is a valid path, i.e., $C$ can navigate to $r_0$, and the UML class of $r_{i-1}$ can navigate to a role/association-class $r_i$ for every $i > 1$. We consider that a UML class can navigate through a role $r_i$ if it has a role property called $r_i$, or some of its compatible classes have it.*

Equivalently, there is a *min cardinality cycle* if we can build an OCL path that starts from class $C$, navigates uniquely through roles whose min cardinality is 1 (or from association classes to its members since an instance of an association class always one member for each association-end), possibly using *oclAsType* to cast some (intermediate) path result to some other compatible classes into which continue navigating, and whose final result is (a collection) of a type compatible with $C$.

**Definition 2.** *A UML class diagram is* weakly acyclic *iff it does not contain any min cardinality cycle for any of its classes.*

In the following, we prove that reasoning on a *weakly acyclic* UML class diagram is decidable. We do so by showing that the logic encoding of a *weakly acyclic* UML schema results into a *weakly acyclic* set of *deds*, which are well-known to be decidable [7, 8].

**Theorem 1.** *Reasoning on a weakly acyclic UML class diagram is decidable.*

*Proof.* The proof starts from the logic encoding of the UML constraints present in the class diagram. In particular, we see that there are only 6 kinds of rules generated by the encoding of constraints in this language [1]:

- Integrity Reference Rules: Assoc(x, y, ...) $\rightarrow$ Class(x)
- Minimum cardinality rules: Class(x) $\rightarrow$ Assoc(x, y, ...)
- Maximum cardinality rules: Assoc(x, y, ...) $\wedge$ Assoc(x, y2, ...) $\rightarrow \bot$
- Hierarchy constraints: Subclass(x) $\rightarrow$ Class(x)
- Disjoint constraints: Subclass1(x) $\wedge$ Subclass2(x) $\rightarrow \bot$
- Complete constraints: Class(x) $\rightarrow$ Subclass1(x) $\vee$ ... $\vee$ SubclassN(x)

Now we see that, if the UML class diagram is weakly acyclic, these kinds of rules form a weakly acyclic set of deds. We do this by contraposition. That is, we show that if there is a cycle in the deds, we can find a min cardinality cycle in the class diagram.

If there is a cycle in the deds, there is, for sure, an existential edge. The unique rules that generate an existential edge are the min cardinality rules, so, there must be a min cardinality constraint in the UML class diagram. We use this class to generate a min cardinality cycle. Indeed, each edge that appears in the cycle in the deds can be seen either as a navigation through a min cardinality 1 role, or the navigation to the class of the association, a superclass or a subclass. We can generate the min cardinality cycle by simply picking the role names of the min cardinality constraint deds appearing in the deds cycle. Since the deds form a cycle, for sure, the navigations ends with the original class $C$. □

Up to here, we know that reasoning on weakly acyclic UML schemas is decidable. Now, we generalize this condition in order to ensure good expressiveness. In particular, it is quite frequent to find a binary association in a UML class diagram with a min cardinality 1 in both association-ends. This forms a trivial cycle in the UML class diagram, and thus, a cycle in its logic encoding.

Assume, for instance, that the (reified) association *TeamInLeague* in our example has a min cardinality 1 in both association-ends: *Team* and *League*. In such case, the logic encoding contains the following cycle:

```
8) Team(t), → TeamInLeague(t, l)
9) TeamInLeague(t, l), → League(l)
10) League(l) → TeamInLeague(t, l)
11) TeamInLeague(t, l) → Team(t)
```

This kind of cycles do not affect the decidability of the schema. Intuitively, this is because everytime we *repair* the ded 9 creating a new league $l$ for some team $t$ we are actually repairing the ded 10 that says that each league (such as $l$) should have at least one team. Thus, ded 10 is not triggered, and the repairing process does not loop forever.

Formally, this kind of cycles satisfy the third decidability theorem identified in [1], which states that when creating new instances to repair such kind of constraints, those instances do not get stacked into an infinite loop.

Therefore, cycles involving only one (non-recursive) association with min cardinalities in both association ends do not break decidability.

### 4.2 Incorporating OCL$_{\mathsf{UNIV}}$

We assume now that the UML class diagram is complemented with OCL$_{\mathsf{UNIV}}$ constraints and show that decidability is still preserved. The basic idea is that these constraints do not create new objects to be repaired, but reclassify them among compatible classes or add new associations to them. Thus, no new cycles with new existential edges are added.

It might seem that an OCL$_{\mathsf{UNIV}}$ constraint can create a new *universal edge* that is involved within a cycle with some existential edge (given by some min card constraint). However, we show that, if this is the case, then, there is a min cardinality cycle in the UML class diagram alone (and thus, the UML class diagram would not be weakly

acyclic). The intuition behind this fact is that, given a ded cycle involving an $\text{OCL}_{\text{UNIV}}$ constraint, we can build a UML min card cycle by simply bypassing $\text{OCL}_{\text{UNIV}}$ constraints. This is because $\text{OCL}_{\text{UNIV}}$ cannot make an existing object become an instance of an arbitrary class, but can only reclassify objects into compatible classes (whose propagations are already taken in account when searching for min cardinality cycles through hierarchies).

We start stating two intermediate Propositions that make the overall proof easier:

**Proposition 1.** *Given a set of* deds *encoding a UML class diagram, if a position $r[i]$ encodes objects compatible with class $C$, then, there is a universal-path (i.e., a path formed by* universal edges *in the dependency graph) from $r[i]$ to $C[0]$.*

*Proof.* There are two different cases: $r$ might encode either a class or an association (the case of association classes is treated similarly).

Assume $r$ is a predicate encoding a class $R$. If $R$ is compatible with $C$ we have that, in the UML class diagram, $R$ is connected to $C$ through hierarchies. This connection is represented in the dependency graph through *deds* encoding hierarchy constraints (intuitively, to go upwards a hierarchy), and *complete* constraints (intuitively, to go downwards a hierarchy). Both kinds of *deds* only generate universal edges and, so, there is a universal-path between them.

Assume $r$ is a predicate encoding an association $R$ and $r[i]$ encodes the i-th member of the association $R$, whose UML class is $RC$. If $RC$ is compatible with $C$, we have a universal path between the positions $rc[0]$ and $c[0]$ (as we have seen previously). Now, because of the ded encoding the integrity reference constraint, there is also a universal edge from $r[i]$ to $rc[0]$. So, there is a universal path between $r[i]$ and $c[0]$. □

With this result at hand, we can prove that, if there is a cycle in the *deds* encoding a UML schema with $\text{OCL}_{\text{UNIV}}$ constraints, then, there is a min cardinality cycle in the UML schema. The proof is based on showing that, if there is a cycle in the *deds*, and such cycle uses some universal edge generated from a *ded* encoding an $\text{OCL}_{\text{UNIV}}$ constraint, we can build a new cycle without using such *ded* by means of replacing such edge for the universal path stated in Proposition 1.

**Proposition 2.** *If there is a cycle in the* deds *encoding a UML schema with $\text{OCL}_{\text{UNIV}}$ constraints, then, there is a min cardinality cycle in the UML schema.*

*Proof.* Take a cycle in the *deds*. If such cycle does not use any edge resulting from a ded encoding an $\text{OCL}_{\text{UNIV}}$ constraint, then, there is a min cardinality cycle in the UML class diagram (see proof of Theorem 1).

If such cycle uses an edge coming from the *ded* of a $\text{OCL}_{\text{UNIV}}$ constraint, we are going to see that we can create a new cycle avoiding such *ded*. Assume that the edge is from positions $r[i]$ to position $s[j]$. First, because $\text{OCL}_{\text{UNIV}}$ does not create existential variables, such edge is going to be a universal edge. Then, because $\text{OCL}_{\text{UNIV}}$ is a typed language, we have that the classes represented in $r[i]$ and $s[j]$ are compatible. Thus, by Proposition 1 there is a universal path between $r[i]$ and $s[j]$ that only goes through deds encoding UML schema constraints. Thus, we can build a cycle with no ded encoding $\text{OCL}_{\text{UNIV}}$ by replacing such edges by their corresponding alternative universal-paths.

So, since there is a cycle using the deds encoding the UML schema, there is a min cardinality cycle in the UML class diagram (see proof of Theorem 1). □

Now, we can finally state that reasoning with weakly acyclic UML class diagrams with OCL$_{\mathsf{UNIV}}$ constraints is decidable.

**Theorem 2.** *Reasoning on a weakly acyclic UML class diagram with OCL$_{\mathsf{UNIV}}$ constraints is decidable.*

*Proof.* Taking the contraposition of Proposition 2, we have that a UML schema with OCL$_{\mathsf{UNIV}}$ constraints but no min cardinality cycle generates a weakly acyclic set of deds, which are known to be decidable [7, 8]. □

### 4.3 Incorporating acyclicity constraints, min cardinalities greater than 1, and incomplete hierarchies

Our goal now is to show that weakly acyclic UML schemas with OCL$_{\mathsf{UNIV}}$ constraints are still decidable when considering: (1) path acyclicity constraints, (2) minimum cardinalities greater than 1, and (3) incomplete hierarchies. In this way, the language we identify is able to deal with almost all identified frequent UML schema constraints [5].

Intuitively, a path is acyclic on a UML class diagram if and only if, given a class of the path, we can establish a stratification of the instances of such class. This can be emulated by means of considering a new fake attribute called *strata* in such class, and adding a new OCL$_{\mathsf{UNIV}}$ constraint forcing that the strata of some instance of the class should be less than the strata that can be obtained through navigation by the cyclic path.

**Theorem 3.** *Reasoning on a weakly acyclic UML class diagram with OCL$_{\mathsf{UNIV}}$ constraints and path acyclicity constraints is decidable.*

*Proof.* The proof is based on reducing the problem to reasoning on a UML class diagram with OCL$_{\mathsf{UNIV}}$ constraints and no path acyclicity constraint.

Indeed, remove the acyclicity constraint and add some fake attribute *strata* in some class belonging to the acyclic path. Then, add a OCL$_{\mathsf{UNIV}}$ constraint stating that each instance of such class should have a *strata* less or equal than the *strata* of the instances that can be obtained by navigating through the acyclic path. Clearly, the first schema is satisfiable iff the second one is satisfiable. Moreover, this transformation does not alter the existance/inexistance of a min cardinality cycle in the diagram (indeed, we are not altering the diagram), so, if the original UML diagram is weakly acyclic, the second one (the one with no acyclicity constraint) is weakly acyclic too, and thus, decidable. □

Consider now min cardinality constraints of n (n $>$ 1). We now show that they can be emulated by considering $n$ new associations of min cardinality one, and adding some constraints to ensure that each of these associations should retrieve a different object, and all $n$ should be included in the original association. Note that, indeed, all these constraints can be written in OCL$_{\mathsf{UNIV}}$. Formally:

**Theorem 4.** *Reasoning on a weakly acyclic UML class diagram with OCL$_{\mathsf{UNIV}}$ constraints and general min cardinality constraints is decidable.*

*Proof.* The proof is based on reducing the problem to reasoning in a UML class diagram with $\text{OCL}_{\text{UNIV}}$ constraints and where all min cardinalities are of the kind "1" or "1..*".

Indeed, remove the min cardinality $n$ and add $n$ new associations with the same members, and a min cardinality 1 in all of them. Then, add $n$ $\text{OCL}_{\text{UNIV}}$ constraints stating that each instance of such associations should be instance of the original association, and that all of them should be disjoint. Clearly, the first schema is satisfiable iff the second one is satisfiable. Moreover, this transformation does not alter the existance/inexistance of a min cardinality cycle in the UML class diagram (indeed, we are only adding min cardinalities between classes which already had min cardinalities), so, if the original UML class diagram is weakly acyclic, the second one (the one with no min cards greater than 1) is weakly acyclic too, and thus, decidable. □

Finally, we show that if a UML class diagram with $\text{OCL}_{\text{UNIV}}$ constraints and incomplete hierarchies is weakly acyclic, then, it is also decidable.

**Theorem 5.** *Reasoning on a weakly acyclic UML class diagram with $\text{OCL}_{\text{UNIV}}$ constraints and incomplete hierarchies is decidable.*

*Proof.* If the UML class diagram with $\text{OCL}_{\text{UNIV}}$ constraints is weakly acyclic, then, its set of deds is weakly acyclic. Hence, the set of deds after removing the deds encoding *complete* constraints is still weakly acyclic. Thus, reasoning on a weakly acyclic UML class diagram with $\text{OCL}_{\text{UNIV}}$ constraints and incomplete hierarchies is decidable. □

## 5 Expressiveness Study

$\text{OCL}_{\text{UNIV}}$ is expressible enough to deal with the typical constraint patterns used in conceptual modeling that were identified in [5]. The unique exception is the path inclusion pattern (i.e., a constraint stating that the instances that can be reached navigating through some path in the UML diagram should be a subset of the instances that can be obtained following another path), which in $\text{OCL}_{\text{UNIV}}$ is limited to paths of only one navigation step.

Such patterns are able to encode about the 60% of textual constraints in conceptual schemas [5]. Nevertheless, $\text{OCL}_{\text{UNIV}}$ is able to encode expressions beyond such patterns, thus, a better coverage is expected.

To show the expressiveness of our fragment, we have evaluated how many constraints could we encode in our $\text{OCL}_{\text{UNIV}}$ (i.e the original $\text{OCL}_{\text{UNIV}}$ with the extension we have proposed in this paper) in several typical UML/OCL case studies. In particular, we have used as case studies the schemas of osCommerce [10] (24 classes and 33 constraints), a Sudoku application [11] (10 classes, and 8 constraints), the DBLP schema [12] (17 classes, and 22 constraints), and the EU-rent fictional system [13] (33 classes, and 33 constraints). It is worth noting that two of them (osCommerce and DBLP) were obtained by reverse engineering of real systems.

For each case study, we have checked how many minimum cardinality constraints must be removed (if any) from the schema in order to ensure that the UML class diagram is *weakly acyclic*, and how many of their constraints may be encoded in $\text{OCL}_{\text{UNIV}}$. To be able to evaluate our results with regards to other decidable fragments of OCL, we

have decided to compare our proposal with OCL-Lite [3], which is, to our knowledge, the unique existing decidable and expressive fragment of OCL (appart form $OCL_{UNIV}$). To establish the comparison, we have counted how many cardinality constraints did we need to remove to ensure that the UML class diagram fulfilled the OCL-Lite requirements, and then, how many constraints could be written in OCL-Lite. Tables 1 and 2 summarize our results.

**Table 1.** Cardinalities removed to be compliant with $OCL_{UNIV}$ and OCL-Lite requirements

| | $OCL_{UNIV}$ | OCL-Lite |
|---|---|---|
| Sudoku | 1 | 17 |
| DBLP | 0 | 14 |
| EU-rent | 0 | 24 |
| osCommerce | 0 | 14 |
| TOTAL | 1 | 69 |

As shown in Table 1, we only had to remove 1 minimum cardinality constraint from the Sudoku schema to ensure that the UML class diagrams were *weakly acyclic* in all cases, and thus, decidable under our $OCL_{UNIV}$ constraints. This is because the unique cycle found was between the relations *Sudoku has Rows*, *Rows have Cells*, *Cells are in Columns*, *Columns are in Sudokus*, which form a cycle with min cardinalities 1 in each association end. To break this cycle, it is only necessary to remove one of such min cardinality constraints. However, this does not entail with our approach a decrease in expressivity since the cardinality between *Column* and *Sudoku* could be replaced with an $OCL_{UNIV}$ constraint saying that each column has the sudoku of its cells (which entails the min cardinality one).

On the other hand, OCL-Lite required removing a total of 69 cardinalities considering all the schemas. This is because OCL-Lite cannot handle maximum cardinalities (which are pretty common in UML diagrams) and, hence, all of them must be removed.

**Table 2.** OCL constraints encodable in our $OCL_{UNIV}$ vs OCL-Lite for our case studies

| | | $OCL_{UNIV}$ | | OCL-Lite | |
|---|---|---|---|---|---|
| | OCL Constraints | Encodable | Non-Encodable | Encodable | Non-encodable |
| Sudoku | 8 | 8 | 0 | 7 | 1 |
| DBLP | 22 | 21 | 1 | 13 | 9 |
| EU-rent | 33 | 27 | 6 | 13 | 20 |
| osCommerce | 33 | 23 | 10 | 21 | 12 |
| TOTAL | 96 | 79 | 17 | 54 | 42 |

In Table 2 we can see that $OCL_{UNIV}$ can encode more constraints than OCL-Lite (82.3% against 56.3%). We believe that this is due to the lack of comparison operators ($=$, $<$, $\leq$, $<>$) in OCL-Lite, which made quite a lot of constraints encodable in $OCL_{UNIV}$ not encodable in OCL-Lite. Conversely, only very few cases were encodable in OCL-Lite but not in $OCL_{UNIV}$. These cases were related to constraints forcing

the existence of some object satisfying a particular condition (for instance, in the osCommerce, a constraint stating that there should exists an *enabled PaymentMethod* among the *PaymentMethods* in the system). Interestingly, neither $\text{OCL}_{\text{UNIV}}$ nor OCL-Lite could deal with some constraints involving path inclusions (such as *the shopping cart atttributes should be included in the attributes of the products of the shopping cart*). OCL-Lite could not encode any of them because of the absence of equalities, and $\text{OCL}_{\text{UNIV}}$ is limited to path inclusion constraints involving paths of only one navigation, which has been proven to be too limited in some of our case studies.

Given these results, we can state that the language we have identified in this paper is, up to now, the most expressive language for defining UML/OCL conceptual schemas while ensuring finite reasoning on them, with a substantial improvement with respect to the closest competitor found in the literature (OCL-Lite).

As a limitation of this experiment, we should point out that, due to the difficulties to find UML schemas with OCL constraints, only four schemas were used. In addition, it might be argued that evaluating expressiveness by means of counting encodable constraints might be insufficient since, subjectively, it could happen that OCL-Lite encoded constraints were more *interesting* than $\text{OCL}_{\text{UNIV}}$ constraints. However, this notion is subjective and thus, out of the scope of the controlled experiment we have carried out.

## 6 Related Work

We analyze languages and approaches related to $\text{OCL}_{\text{UNIV}}$. We distinguish between UML/ER based, and tgd-based.

**UML/ER based** Reasoning the satisfiability of an ER diagram considering only association cardinalities is polynomial [14]. When, considering UML schemas with all features in exception of OCL constraints, the problem becomes EXPTIME-complete [2].

Adding OCL-Lite constraints in UML class diagrams maintains the EXPTIME-complete complexity (and thus, decidability), although it requires removing the maximum cardinality constraints [3]. This requirement is, in our opinion, too strong since most realistic UML class diagrams always have some kind of maximum cardinality. In addition, we have seen during our expressiveness study that this inability to encode constraints involving equalities/inequalities represents also a drawback in comparison to $\text{OCL}_{\text{UNIV}}$. However, OCL-Lite is not subsumed by $\text{OCL}_{\text{UNIV}}$ since, for instance, OCL-Lite is capable of encoding the *exists* operator, which is forbidden in $\text{OCL}_{\text{UNIV}}$.

Another option consists in using general UML/OCL constraints, and then, analyze whether that particular UML/OCL schema is decidable or not [1, 6]. This approach subsumes $\text{OCL}_{\text{UNIV}}$. Indeed, $\text{OCL}_{\text{UNIV}}$ decidability relies in *weak acyclicity*, which is a condition subsumed by the previous decidability analysis [1]. However, we argue that it is quite difficult from the point of view of a conceptual modeller to write a UML/OCL schema that satisfies such decidability conditions. This is because such conditions are checked in the logic encoding of the UML/OCL schema, rather than the UML/OCL itself. In contrast, it is easy to check whether a UML/OCL schema satifies the decidabiilty requirements of our $\text{OCL}_{\text{UNIV}}$ (i.e., *weak acyclicity* and the syntax of $\text{OCL}_{\text{UNIV}}$).

Another interesting work is the one presented in [15]. In this work, UML/OCL constraints are written under the form of several constraint patterns, and such constraint patterns are analyzed to be consistent/inconsistent polynomially (through syntactic checks), which means that they not only guarantee decidability of reasoning, but also efficiency. However, this approach might bring false positives (i.e., it might say that some set of consistent constraints are inconsistent).

Finally, we have the approaches based in Armstrong tables [16]. An Armstrong table is, roughly speaking, an instantiation of the schema which exemplifies the satisfaction of the constraints entailed by the schema (and is a counterexample for anyone else). [16] shows how to build Armstrong tables for schemas considering min/max and not-null constraints. However, it does not target general constraints as we do.

**TGD based** Our approach is based on a logic encoding in deds of the UML/OCL schema, and the decidability of reasoning over such logic encoding. In particular, we have used the ded weak acyclicity property to ensure the termination of the chase algorithm to reason with such deds [7].

The weak acyclicity property of deds is subsumed by the stratification property stated in [8]. This means that we could potentially enlarge the subset of UML/OCL we can deal with if we based on stratification rather than weak acyclicity. However, we argue that this change is unfeasible. Indeed, checking whether a set of deds satisfy the weak acyclicity consists in a simple graph analysis, thus, we only needed to characterize which kind of UML/OCL expressions would bring an acyclic graph. In contrast, checking the stratification property requires solving a NP problem for each edge in the dependency graph. In our opinion, it is quite difficult to find some condition over the UML/OCL level that ensures that such NP condition is satisfied at the logic level.

Another family of decidable languages based on tgds is Datalog$^{+/-}$ [17]. The basic notion in Datalog$^{+/-}$ is *guardedness*. A ded is said to be guarded if there is some atom containing all its universal variables in a single atom in the left-hand side (called guard). Under this situation, reasoning over such set of deds is decidable. It is easy to see that $\text{OCL}_{\text{UNIV}}$ is not subsumed by this language since ded 5 of our example is not guarded. It is also worth noting that $\text{OCL}_{\text{UNIV}}$ does not subsume Datalog$^{+/-}$ since it does not offer existential variables (appart from the special min cardinality 1 case). So, both are languages with different expressiveness. However, we argue that it is quite difficult to realize an expressive OCL subset that ensures that its logic encoding is *guarded*.

## 7 Conclusions

We have seen that reasoning with UML schemas with no minimum cardinality cycles, path acycliclity constraints and $\text{OCL}_{\text{UNIV}}$ constraints is decidable. This decidability result is guaranteed because, by construction, the logic encoding into deds of such schema is *weakly acyclic*, which guarantees that the chase algorithm terminates on such schema. Current UML/OCL reasoners such as [6] can benefit from this termination result.

We have compared the expressiveness of the decidable language we have identified with that of OCL-Lite [3], another decidable language based on OCL, and we have seen that $\text{OCL}_{\text{UNIV}}$ is more expressive in all different UML/OCL case studies we have taken

into account (while $\text{OCL}_{\text{UNIV}}$ could handle about the 82% of constraints appearing in these schemas, OCL-Lite could only deal with around 56%).

As future work, we would like to extend our proposal to be able to admit more constraints involving existential variables. We understand that special attention should be put to inclusion path constraints. In addition, $\text{OCL}_{\text{UNIV}}$ is designed only for ensuring decidability, so, a more sophisticated analysis should be done to bound its complexity.

*Acknowledgements:* this work is supported by the Ministerio de Economia y Competitividad, project TIN2014-52938-C2-2-R and by the Secretaria d'Universitats i Recerca de la Generalitat de Catalunya, project 2014 SGR 1534.

## References

1. Queralt, A., Teniente, E.: Verification and validation of UML conceptual schemas with OCL constraints. ACM Trans. Softw. Eng. Methodol. **21**(2) (2012) 13
2. Berardi, D., Calvanese, D., De Giacomo, G.: Reasoning on UML class diagrams. Artificial Intelligence **168**(1-2) (2005) 70–118
3. Queralt, A., Artale, A., Calvanese, D., Teniente, E.: OCL-lite: Finite reasoning on uml/ocl conceptual schemas. Data and Knowledge Engineering **73** (2012) 1 – 22
4. Oriol, X., Teniente, E., Tort, A.: Computing repairs for constraint violations in UML/OCL conceptual schemas. Data and Knowledge Engineering **99** (2015) 39–58
5. Costal, D., Gómez, C., Queralt, A., Raventós, R., Teniente, E.: Improving the definition of general constraints in UML. Software & Systems Modeling **7**(4) (2008) 469–486
6. Rull, G., Farré, C., Queralt, A., Teniente, E., Urpí, T.: AuRUS: explaining the validation of UML/OCL conceptual schemas. Software & Systems Modeling **14**(2) (2015) 953–980
7. Fagin, R., Kolaitis, P.G., Miller, R.J., Popa, L.: Data exchange: semantics and query answering. Theoretical Computer Science **336**(1) (2005) 89 – 124
8. Deutsch, A., Nash, A., Remmel, J.: The chase revisited. In: Proc. of the 27th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems. PODS '08, ACM (2008) 149–158
9. Oriol, X., Teniente, E.: Incremental checking of OCL constraints through SQL queries. In: Proc. of the 14th Int. Workshop on OCL and Textual Modelling. (2014) 23–32
10. Tort, A.: (The osCommerce case study) `http://www-pagines.fib.upc.es/~modeling/osCommerce_cs.pdf`.
11. Tort, A., Olivé, A.: (The sudoku case study) `http://www.essi.upc.edu/~atort/documents/Sudoku.pdf`.
12. Planas, E., Olivé, A.: The DBLP case study (2006) `http://www-pagines.fib.upc.es/~modeling/DBLP.pdf`.
13. Estañol, M., Queralt, A., Sancho, M.R., Teniente, E.: EU-rent car rentals specification. Technical report, Universitat Politècnica de Catalunya (2012) `http://www.essi.upc.edu/~estanyol/docs/artifacts_eu_rent.pdf`.
14. Hartmann, S.: On the consistency of int-cardinality constraints. In: Proc. of 17th International Conference on Conceptual Modeling. (1998) 150–163
15. Wahler, M., Basin, D., Brucker, A.D., Koehler, J.: Efficient analysis of pattern-based constraint specifications. Software & Systems Modeling **9**(2) (2010) 225–255
16. Hartmann, S., Köhler, H., Leck, U., Link, S., Thalheim, B., Wang, J.: Constructing armstrong tables for general cardinality constraints and not-null constraints. Ann. Math. Artif. Intell. **73**(1-2) (2015) 139–165
17. Calì, A., Gottlob, G., Lukasiewicz, T.: A general datalog-based framework for tractable query answering over ontologies. Web Semantics: Science, Services and Agents on the World Wide Web **14** (2012) 57–83