An Algebraic Semantics for UML+OCL Class Diagrams *

Manuel Clavel$^1$ and Marina Egea$^1$

Universidad Complutense de Madrid, Spain

Abstract. In this paper we propose an algebraic semantics for UML+OCL class diagrams that provides a formal foundation for automatically validating UML object diagrams with respect to OCL constraints. Based on this semantics we are developing a tool, named ITP/OCL, that will provide automatic generation and validation of object diagrams with respect to OCL constraints.

1 Introduction

The Unified Modeling Language (UML) [11] is a general-purpose visual modeling language that is used to specify, visualize, construct, and document the artifacts of a software system. The UML notation is largely based on diagrams. However, for certain aspects of a design, diagrams often do not provide the level of conciseness and expressiveness that a textual language can offer. The Object Constraint Language (OCL) [10] is a textual constraint language with a notational style similar to common object oriented languages. OCL came to provide help on precise information specification in UML models. Although designed to be a formal language, experience with OCL has shown that the language definition is not precise enough. Various authors have pointed out language issues related to ambiguities, inconsistencies or open interpretations [13,17,3,9].

Validation and testing in object-oriented software development has been recognized of key importance for long. A number of CASE tools exists which facilitate drawing and documenting UML diagrams. However, there is only little support for validating models during the design stage. Also, there is generally no substantial support for constraints written in OCL. We propose an algebraic semantics for UML+OCL class diagrams that provides a formal foundation for validating UML object diagrams with respect to OCL constraints. We use Membership Equational Logic (MEL) [1], an expressive version of equational logic, to specify class and object diagrams, and we formalize OCL constraints as formulas over those theories. As a result, validating UML object diagrams with respect to OCL constraints can be reduced to proving invariant-formulas over the standard models of invariant-object diagram theories. Based on this semantics, we are developing a tool, named ITP/OCL, that will provide automatic generation and validation of object diagrams with respect to OCL constraints.

* Research supported by Spanish MEC Projects TIC2002-01167 and TIC2003-01000.
The tool is intended as a lightweight formal method: it will help to find flaws in UML class diagrams in the early phases of the software development process. The latest version of the tool, with the available documentation and examples can be found at http://maude.sip.ucm.es/itp/ocl/.

Organization In Section 2 we summarize background material on membership equational logic. In Sections 3 and 4 we propose an algebraic specification of class and object diagrams as theories in MEL. In Section 5 we propose a formalization of invariants over class diagrams as formulas over extensions of the theories specifying them. In Section 6 we briefly introduce the ITP/OCL tool. Finally, in Sections 7 and 8 we report on related work and draw conclusions. In the Appendix we include examples of a class diagram and its corresponding class-theory; of an object diagram and its corresponding object-theory; and of an invariant and its corresponding invariant-theory and invariant-formula.

2 Membership Equational Logic (MEL)

Membership equational logic (MEL) is an expressive version of equational logic; a full account of its syntax and semantics can be found in [1].

A signature in MEL is a triple $\Omega = (K, \Sigma, S)$, with $K$ a set of kinds, $\Sigma = \{\Sigma_{k_1, \ldots, k_n, k}\}_{(k_1, \ldots, k_n, k) \in K \times K}$ a many-kinded signature, and $S = \{S_k\}_{k \in K}$ a pairwise disjoint $K$-kinded family of sets of sorts. The basic intuition is that correct or well-behaved terms are those that can be proved to have a sort, whereas error or undefined terms are terms that have a kind but do not have a sort. For example, assuming addition (+) and integer division (/) operators with the appropriate declarations, $3+4$ is a term of sort $\text{Int}$, but $7/0$ is a term of the kind of the sort $\text{Int}$ with no sort. We write $T_{\Sigma, k}$ and $T_{\Sigma, k}(x)$ to denote respectively the set of ground $\Sigma$-terms with kind $k$ and of $\Sigma$-terms with kind $k$ over variables in $x$, where $x = \{x_1 : k_1, \ldots, x_n : k_n\}$ is a set of $K$-kinded variables. Sometimes we use the notation $t(x)$ to make explicit the set of variables that appear in the term $t$.

The atomic formulas of MEL are either equations $t = t'$, where $t$ and $t'$ are terms of the same kind, or membership assertions of the form $t : s$, where the term $t$ has kind $k$ and $s \in S_k$. Sentences are Horn clauses on these atomic formulas, i.e., sentences of the form

$$\forall \{x\} \ A_1 \land \ldots \land A_n \Rightarrow A_0,$$

where each $A_i$ is either an equation or a membership assertion and $x$ is a set of $K$-kinded variables. A theory in MEL is a pair $(\Omega, E)$, where $E$ is a finite set of sentences in MEL over the signature $\Omega$. An $\Omega$-algebra $A$, with $\Omega = (K, \Sigma, S)$, consists of a set $A_k$ for each $k \in K$, a function $A_f : A_{k_1} \times \ldots \times A_{k_n} \rightarrow A_k$ for each operator $f \in \Sigma_{k_1, \ldots, k_n, k}$, and a subset $A_s \subseteq A_k$ for each sort $s \in S_k$. An algebra $A$ and a valuation $\sigma$, assigning to each variable $x : k$ in $x$ a value in $A_k$, satisfy an equation $\forall \{x\} t = t'$ if and only if $\sigma(t) = \sigma(t')$, where we use the same notation $\sigma$ for the valuation and its homomorphic extension to terms. We write
A, \sigma \models \forall(x) t = t' to denote such a satisfaction. Similarly, \( A, \sigma \models \forall(x) t : s \) holds if and only if \( \sigma(t) \in A_s \). We write \( A \models \phi \) when the formula \( \phi \) is satisfied for all valuations \( \sigma \), and then say that \( A \) is a model of \( \phi \). As usual, we write \( (\Omega, E) \models \phi \) when all the models of the set \( E \) of sentences are also models of \( \phi \).

A theory \( (\Omega, E) \) in MEL has an initial model [1], denoted by \( T_{\Omega\mid E} \), whose elements are equivalence classes \( [t]_E \) of ground terms. In the initial model, sorts are interpreted as the smallest sets satisfying the axioms in the theory, and equality is interpreted as the smallest congruence satisfying those axioms. We write \( (\Omega, E) \models \phi \) to denote that the initial model of the membership equational theory \( (\Omega, E) \) is also a model of \( \phi \), that is, that the satisfaction relation \( T_{\Omega\mid E} \models \phi \) holds.

3 UML Class Diagrams

In this section we first summarize the notion of class diagrams in UML. Then, we introduce a formal description of class diagrams as mathematical structures. Finally, we propose an algebraic specification of class diagrams structures as theories in MEL.

The static view models concepts in the application domain as well as internal concepts invented as part of the implementation of an application. It does not describe the time-dependent behavior of the system, which is described in other views. Key elements in the static view are classes and their relationships. A class is a modeling element that describes entities containing values. Classes may be related in different ways, including association, generalization, and various kinds of dependency. The static view is displayed in class diagrams. A class diagram has the following components: a set of classes, a set of attributes for each class, a set of operations for each class, a set of associations with role names and multiplicities, and a generalization hierarchy over classes.

3.1 Class Diagrams as Structures

We introduce a formal description of class diagrams as mathematical structures. In the next section we propose an algebraic specification of class diagrams directly based on their formal description as mathematical structures.\footnote{For the sake of simplicity, we only consider classes with binary associations and without operations, and we do not consider enumeration classes.}

In the following definition, let \( V \) be the set \{Boolean, Integer, String\} of UML basic types.

**Definition 1 (class diagram structures).** A class diagram is described by a structure

\[
CD = (C_{CD}, \prec_{CD}, ATT_{CD}, ASSOC_{CD}, ROL_{CD})
\]

where:

- \( C_{CD} = \{c \mid c \text{ is a class}\} \).
is a partial order over \( C_{CD} \), which describes the generalization hierarchy over classes.

- \( ATT_{CD} = \{ ATT_{(c,v)} \}_{(c,v) \in C_{CD} \times V} \), where
  \[ ATT_{(c,v)} = \{ at \mid at \text{ is an attribute of } c \text{ with type } v \} \].

- \( ASSOC_{CD} = \{ ASSOC_{(c,c')} \}_{(c,c') \subseteq C_{CD} \times C_{CD}} \), where
  \[ ASSOC_{(c,c')} = \{ as \mid as \text{ is an association between } c \text{ and } c' \} \].

- \( ROL_{CD} = \{ ROL_{(c,c')} \}_{(c,c') \subseteq C_{CD} \times C_{CD}} \), where
  \[ ROL_{(c,c')} = \{ p \mid p \text{ is the role played by } c \text{ in an association } as \in ASSOC_{(c,c')} \} \].

### 3.2 Class Diagrams as Theories in MEL

We propose an algebraic specification of class diagrams as membership equational theories directly based on their formal description as mathematical structures. Basically,

- classes are represented by sorts of the kind \( Class \);
- class collections are represented by sorts of the kind \( ClassCol \); and
- attributes and roles are represented by operators of the appropriate ranks.

Notice that in the algebraic specifications of class diagrams, the sorts representing the classes are empty and the operators representing the attributes and roles are undefined. In the next section we propose an algebraic specification of object diagrams as extensions—sorts representing classes are filled with elements representing objects, and operators representing attributes and roles are defined for those elements—of the algebraic specification of their class diagram.

#### Definition 2 (class diagram theories).

A class diagram

\[ CD = (C_{CD}, \prec_{CD}, ATT_{CD}, ASSOC_{CD}, ROL_{CD}) \]

is specified by a theory \( CD = (\Omega_{CD}, \Gamma_{CD}) \) in MEL, where\(^2\)

\[ \Omega_{CD} = (K_{CD}, \Sigma_{CD}, S_{CD}) \text{, where} \]

- \( K_{CD} = \{ Class, ClassCol, Value \} \);  
- \( S_{CD} = \{ S_{Class}, S_{ClassCol}, S_{Value} \} \), with
  - \( S_{Class} = \{ c \mid c \in C_{CD} \} \),
  - \( S_{ClassCol} = \{ \text{Col}(c) \mid c \in C_{CD} \} \), and
  - \( S_{Value} = \{ \text{Boolean}, \text{Integer}, \text{String} \} \);  
- \( \Sigma_{CD} = \{ \Sigma_{q,k} \}_{q \in K \cdot k \in K} \), where
  - \( \Sigma_{Class, Value} = \bigcup_{(c,v) \in C_{CD} \times V} \{ at \mid at \in ATT_{(c,v)} \} \), and
  - \( \Sigma_{Class, ClassCol} = \bigcup_{(c,c') \in C_{CD} \times C_{CD}} \{ p \mid p \in ROL_{(c,c')} \} \); and

\(^2\) We assume that the theory includes the theories specifying integers, strings and booleans.
* \( \Sigma_{\text{ClassCol}} = \{ \text{empty} \} \); and
* \( \Sigma_{\text{Class}, \text{ClassCol}} = \{ \text{col} \} \).

- \( \Gamma_{\text{CD}} = \Gamma_{\prec_{\text{CD}}} \cup \Gamma_{\mathit{ATT}_{\text{CD}}} \cup \Gamma_{\mathit{ROL}_{\text{CD}}} \), where
  - \( \Gamma_{\prec_{\text{CD}}} = \{ \forall x \{ x : c \Rightarrow x : c' \} \mid (c, c') \in \prec_{\text{CD}} \} \), where \( x \) is a variable of kind Class;
  - \( \Gamma_{\mathit{ATT}_{\text{CD}}} = \bigcup_{(c, c') \in \text{Class} \times \text{Class}} \{ \forall x \{ x : c \Rightarrow \text{at}(x) : v \} \mid \text{at} \in \mathit{ATT}_{(c, c')} \} \), where \( x \) is a variable of kind Class;
  - \( \Gamma_{\mathit{ROL}_{\text{CD}}} = \bigcup_{(c, c') \in \text{Class} \times \text{Class}} \{ \forall x \{ x : c \Rightarrow p(x) : \text{Col}(c) \} \mid p \in \mathit{ROL}_{(c, c')} \} \), where \( x \) is a variable of kind Class.

4 UML Object Diagrams

In this section we first summarize the notion of object diagrams in UML. Then, we introduce a formal description of object diagrams as mathematical structures. Finally, we propose an algebraic specification of object diagrams structures as theories in MEL.

A system may be in different states as it changes over time. An object diagram models the objects and links that represent the state of a system at a particular moment. An object is an instance of a class. A link is an instance of an association. An object diagram is primarily a tool for research and testing. It can be used to understand a problem by documenting examples from the problem domain. It can also be used during analysis and design to verify the accuracy of class diagrams.

4.1 Object Diagrams as Structures

We introduce a formal description of object diagrams as mathematical structures. In the next section we propose an algebraic specification of object diagrams directly based on their formal description as mathematical structures.

**Definition 3 (object diagram structure).** An object diagram for a class diagram

\[
\text{CD} = (\text{C}_{\text{CD}}, \prec_{\text{CD}}, \mathit{ATT}_{\text{CD}}, \text{ASSOC}_{\text{CD}}, \text{ROL}_{\text{CD}})
\]

is described by a structure

\[
\mathcal{O}_{\text{CD}} = (O_{\text{CD}}, O\mathit{ATT}_{\text{CD}}, O\mathit{ROL}_{\text{CD}}),
\]

where:

- \( O_{\text{CD}} = \{ O_c \}_{c \in \text{C}_{\text{CD}}} \), where
  \( O_c = \{ o \mid o\text{ is an instance of class } c \in \text{C}_{\text{CD}} \} \).
- \( O\mathit{ATT}_{\text{CD}} = \{ O\mathit{ATT}_{(c, v)} \}_{(c, v) \in \text{C}_{\text{CD}} \times \text{V}} \), where
  \( O\mathit{ATT}_{(c, v)} = \{ \text{at} : O_c \Rightarrow O_v \mid \text{at} \in O\mathit{ATT}_{(c, v)} \} \).
- \( O\mathit{ROL}_{\text{CD}} = \{ O\mathit{ROL}_{(c, c')} \}_{(c, c') \in \text{C}_{\text{CD}} \times \text{C}_{\text{CD}}} \), where
  \( O\mathit{ROL}_{(c, c')} = \{ p : O_{c'} \Rightarrow \mathcal{P}(O_c) \mid p \in O\mathit{ROL}_{(c, c')} \} \),

where \( \mathcal{P}(O_c) \) is the powerset of \( O_c \).
4.2 Object Diagrams as Theories in MEL

The relation between class diagrams and object diagrams is reflected by the fact that object diagram theories extend their corresponding class diagram theories by instantiating the sorts representing their classes and the operators representing their attributes and roles. Basically,

- objects are represented by constants of the kind \texttt{Class}, which are declared to belong to the sorts representing their classes;
- values are declared for object attributes by defining the value of the operators representing the attributes when applied to the constants representing the objects; and
- collections are associated to object roles by defining the value of the operators representing the roles when applied to the constants representing the objects.

**Definition 4 (object diagram theories).** Let \( CD = (\Omega_{CD}, \Gamma_{CD}) \) be a class diagram theory. An object diagram \( O_{CD} = (O_{CD}, OATT_{CD}, OROL_{CD}) \) is specified by a theory \( O_{CD} = (\Omega_{CD} \cup \Omega_{OCD}, \Gamma_{CD} \cup \Gamma_{OCD}) \) in MEL, where

- \( \Omega_{OCD} = (K_{CD}, (\Sigma_{CD} \cup \Sigma_{cdr}), \Sigma_{CD}) \), where
  - \( \Sigma_{OCD} = \{ \Sigma_{q,k} \}_{q \in K, k \in K} \), where
  - \( \Sigma_{\lambda, \text{class}} = \bigcup_{c \in C_{cdr}} O_{c} \).
- \( \Gamma_{OCD} = \Gamma_{OCD} \cup \Gamma_{OATT_{CD}} \cup \Gamma_{OROL_{CD}} \), where:
  - \( \Gamma_{OCD} = \bigcup_{c \in C_{cdr}} \{ o : c \mid o \in O_{c} \} \).
  - \( \Gamma_{OATT_{CD}} = \bigcup_{(c, v) \in C_{cdr} \times V} \{ at(o) = at^{OATT_{(c,v)}}(o) \mid o \in O_{c}, at \in ATT_{(c,v)} \} \), where \( at^{OATT_{(c,v)}}(o) \) is the value of \( at(o) \) in the structure \( O_{CD} \).
  - \( \Gamma_{OROL_{CD}} = \bigcup_{(c, c') \in C_{cdr} \times C_{cdr}} \{ p(o) = p^{OROL_{(c,c')}}(o) \mid o \in O_{c'}, p \in ROL_{(c,c')} \} \), where \( p^{OROL_{(c,c')}}(o) \) is the value of \( p(o) \) in the structure \( O_{CD} \), and \( (\underline{\_}) \) is a function that represents elements in \( \mathcal{P}(O_{c}) \), i.e., sets of objects in the class \( c \), as terms of sort \( \text{Col}[c] \).

The function \( (\underline{\_}) \) builds a list (using the list-constructors \( \text{col} \) and \( \text{empty} \)) with the objects in the given set, sorted by their names and without repetitions.

5 OCL Invariants

OCL is a pure specification language on top of UML. It is a textual language with a notational style similar to common object oriented languages. It can be used to specify constraints concerning the static structure and the behavior of a system. The most important uses of OCL expressions in UML diagrams are [18]: the specification of invariants on classes and types in the class diagram; the specification of constraints on operations and methods; the description of pre- and post-conditions on operations; the specification of initial values and
derivation rules for attributes; the specification of query operations; and the introduction of new attributes and operations.

In this section we propose a formalization of invariants over class diagrams as formulas over extensions of the theories specifying them. These extensions basically contain:

- the specification of the generic OCL operators over collections; and
- the specification of the instances of the generic OCL iterators over collections that occurs in the invariants under consideration.

At present, we do not consider invariants describing pre- and post-conditions on operations.

5.1 Invariants as Formulas

We first define the invariant theory extensions. Then, we define the representation of expressions occurring in invariants as terms over the signatures of the invariant theory extensions. Finally, we define the representation of invariants as formulas over the invariant theory extensions.

In the following definition, \( @ \) is an injective function from OCL expressions to strings of characters (which do not contain blank spaces).\(^3\)

Definition 5 (invariant-class diagram theories). Let \( \text{inv} \) be an OCL invariant over a class diagram structure \( CD \),

\[
CD = (C_{CD}, \sim_{CD}, ATT_{CD}, ASSOC_{CD}, ROL_{CD})
\]

Let \( CD \) be the class diagram theory,

\[
CD = (\Omega_{CD}, \Gamma_{CD})
\]

that specifies \( CD \) in MEL. We define the following theory extension \( inv_{CD} \) of \( CD \):

\[
inv_{CD} \triangleq (\Omega_{CD} \cup \Omega_{col} \cup \Omega_{inv_{CD}}, \Gamma_{CD} \cup \Gamma_{col} \cup \Gamma_{inv_{CD}}),
\]

where:

- The operators in \( \Omega_{col} \) represent OCL generic operators over collections, i.e., they do not depend on the particular invariant \( \text{inv} \). For the sake of space limitation, we omit them (see [6] for details).
- \( \Omega_{inv_{CD}} \) is an injective function from OCL expressions to strings of characters (which do not contain blank spaces).\(^3\)

\[3\] Consider, for example, a function that concatenates in a single string all the characters, different from blank spaces, appearing (from left to right) in the given expression.
The axioms in \( \Sigma_{\text{collect}} \) are the set containing an operator forAll@\( \exp' \) for each iterator expression forAll-(\( \text{id}' : \exp' \)) occurring in the invariant inv. Similarly for \( \Sigma_{\text{exists}} \).

\[ \Sigma_{\text{collect}} \subseteq \Sigma_{\text{collect}}, \]

* \( \Sigma_{\text{collect}} \subseteq \Sigma_{\text{collect}}, \quad \text{and} \quad \Sigma_{\text{coll}} \subseteq \Sigma_{\text{coll}}, \quad \text{collect@}\exp' \) for each iterator expression collect-(\( \text{id}' : \exp' \)) occurring in the invariant inv. Similarly for \( \Sigma_{\text{select}}, \Sigma_{\text{reject}} \).

\[ \Sigma_{\text{any}} \subseteq \Sigma_{\text{any}}, \quad \text{any@}\exp' \) for each iterator expression any-(\( \text{id}' : \exp' \)) occurring in the invariant inv.

- The axioms in \( \Gamma_{\text{col}} \) define the operators in \( \Sigma_{\text{col}} \). For the sake of space limitation, we omit them (see [6] for details).

\[ \Gamma_{\text{inv}} = \Gamma_{\text{forAll}} \cup \Gamma_{\text{exists}} \cup \Gamma_{\text{collect}} \cup \Gamma_{\text{reject}} \cup \Gamma_{\text{any}}. \]

The axioms in \( \Gamma_{\text{iter}} \) for iter any of the generic OCL iterators (i.e., forAll, exists, collect, select, reject and any), define the operators iter@\exp', representing OCL iterator expressions iter-(\( \text{id}' : \exp' \)). Basically,

- the first axiom declares that iter@\exp' is an operator of the expected sort when their arguments are of the appropriate sorts.

In the case of collect,

- the second and third axioms define the value of iter@\exp', respectively, over empty collections and non-empty collections.

In the other cases,

- the second axiom defines the value of iter@\exp' over empty collections;

and

- the third and forth axioms define the value of iter@\exp' when \( \exp' \) is, respectively, true or false for an arbitrary element in non-empty collections.

In the definition of the axioms below, we use the representation function \( \overline{\cdot} \), introduced in Definition 6 below, which represents invariant expressions as terms of the appropriate sorts. Also, we use \( t[u/v] \) to denote the replacement of \( u \) by \( v \) in \( t \). Finally, we use operators over collections (isEmpty, excluding, getOne, and unionBag), which are declared in \( \Omega_{\text{col}} \) and are defined in \( \Gamma_{\text{col}} \).

- \( \Gamma_{\text{collect}} \) is the set containing instances of the following axioms for each collect@\exp' in \( \Sigma_{\text{collect}} \):

\[ (1) \forall \{ x, u \}; \quad \{ x : c \land u : \text{Col}[c'] \} \Rightarrow \text{collect@}\exp'(x, u) : \text{Col}[c']. \]

\[ (2) \forall \{ x, u \} ; \quad \{ x : c \land u : \text{Col}[c'] \} \land \text{isEmpty}(u) = \true \Rightarrow \text{collect@exp'}(x, u) = \text{empty}. \]

\[ (3) \forall \{ x, u \} ; \quad \{ x : c \land u : \text{Col}[c'] \} \land \text{isEmpty}(u) = \false \Rightarrow \text{collect@exp'}(x, u) = \text{unionBag}(\exp'[\overline{\text{id}'} : c'/ \text{getOne}(u)[\text{self}[x]], \text{collect@exp'}(x, \text{excluding}(\text{getOne}(u), u))). \]

- \( \Gamma_{\text{forAll}} \) is the set containing instances of the following axioms for each forAll@\exp' in \( \Sigma_{\text{forAll}} \):

\[ (1) \forall \{ x, u \} ; \quad \{ x : c \land u : \text{Col}[c'] \} \Rightarrow \text{forAll@}\exp'(x, u) : \text{Boolean}. \]

\[ (2) \forall \{ x, u \} ; \quad \{ x : c \land u : \text{Col}[c'] \} \land \text{isEmpty}(u) = \true \Rightarrow \text{forAll@exp'}(x, u) = \true. \]

\[ (3) \forall \{ x, u \} ; \quad \{ x : c \land u : \text{Col}[c'] \} \land \text{isEmpty}(u) = \false \]
\[
\forall \{x, u\} (x: c \land u: \text{Col}[c] \land \text{isEmpty}(u) = false \\
\text{false if } V = (V_1 \text{ and } V_2) \\
\text{false if } V = (V_1 \text{ or } V_2) \\
\text{true if } V = (V_1 \text{ implies } V_2) \\
\text{true if } V = (V_1 \text{ xor } V_2) \\
\text{false if } V = (E_1 = E_2) \\
\text{false if } V = (E_1 < E_2) \\
\text{false if } V = (E_1 > E_2) \\
\text{false if } V = (E_1 \leq E_2) \\
\text{false if } V = (E_1 \geq E_2) \\
\text{false if } V \equiv S.\text{isEmpty}()
\]

- Similarly for \(\Gamma_{\text{exists}_{\text{inv}}}, \Gamma_{\text{select}_{\text{inv}}}, \Gamma_{\text{reject}_{\text{inv}}}, \text{and } \Gamma_{\text{any}_{\text{inv}}}\).

Next, we define the function that represents expressions occurring in invariants as terms.

**Definition 6 (representation function for expressions).** Let \(\text{exp}\) be an OCL expression occurring in an invariant \(\text{inv}\) over a class diagram structure \(\text{CD}\). The function \(\langle \_ \rangle\) represents \(\text{exp}\) as a term of the appropriate sort in the theory \(\text{inv}_{\text{CD}}\).

- Let \(\text{exp} \equiv V\) be an OCL boolean expression. Then,

\[
\begin{align*}
\langle \text{true} \rangle & \quad \text{if } V \equiv \text{true} \\
\langle \text{false} \rangle & \quad \text{if } V \equiv \text{false} \\
\langle \text{V}_1 \text{ and } \text{V}_2 \rangle & \quad \text{if } V \equiv (\text{V}_1 \text{ and } \text{V}_2) \\
\langle \text{V}_1 \text{ or } \text{V}_2 \rangle & \quad \text{if } V \equiv (\text{V}_1 \text{ or } \text{V}_2) \\
\langle \text{V}_1 \text{ implies } \text{V}_2 \rangle & \quad \text{if } V \equiv (\text{V}_1 \text{ implies } \text{V}_2) \\
\langle \text{V}_1 \text{ xor } \text{V}_2 \rangle & \quad \text{if } V \equiv (\text{V}_1 \text{ xor } \text{V}_2) \\
\langle \text{equal(E}_1, \text{E}_2) \rangle & \quad \text{if } V \equiv (\text{E}_1 = \text{E}_2) \\
\langle \text{E}_1 < \text{E}_2 \rangle & \quad \text{if } V \equiv (\text{E}_1 < \text{E}_2) \\
\langle \text{E}_1 > \text{E}_2 \rangle & \quad \text{if } V \equiv (\text{E}_1 > \text{E}_2) \\
\langle \text{E}_1 \leq \text{E}_2 \rangle & \quad \text{if } V \equiv (\text{E}_1 \leq \text{E}_2) \\
\langle \text{E}_1 \geq \text{E}_2 \rangle & \quad \text{if } V \equiv (\text{E}_1 \geq \text{E}_2) \\
\langle \text{includes(S, E)} \rangle & \quad \text{if } V \equiv S.\text{includes}(E) \\
\langle \text{excludes(S, E)} \rangle & \quad \text{if } V \equiv S.\text{excludes}(E) \\
\langle \text{isEmpty(S)} \rangle & \quad \text{if } V \equiv S.\text{isEmpty}() \\
\langle \text{notEmpty(S)} \rangle & \quad \text{if } V \equiv S.\text{notEmpty}() \\
\langle \text{at(X)} \rangle & \quad \text{if } V \equiv X.\text{at} \\
\langle \text{exists@V'(self, S)} \rangle & \quad \text{if } V \equiv S.\text{exists}(X IV') \\
\langle \text{forall@V'(self, S)} \rangle & \quad \text{if } V \equiv S.\text{forall}(X IV')
\end{align*}
\]
Let $exp \equiv E$ be an OCL integer expression. Then,

$$E \triangleq \begin{cases} 
    i & \text{if } E \equiv i \in \mathcal{I} \\
    x: \text{Integer} & \text{if } E \equiv x: \text{Integer} \\
    -E_1 & \text{if } E \equiv (-E_1) \\
    E_1 + E_2 & \text{if } E \equiv (E_1+E_2) \\
    E_1 - E_2 & \text{if } E \equiv (E_1-E_2) \\
    E_1 \cdot E_2 & \text{if } E \equiv (E_1\cdot E_2) \\
    E_1 \div E_2 & \text{if } E \equiv (E_1\div E_2) \\
    \text{abs}(E_1) & \text{if } E \equiv E_1.\text{abs}() \\
    \text{mod}(E_1, E_2) & \text{if } E \equiv E_1.\text{mod}(E_2) \\
    \text{max}(E_1, E_2) & \text{if } E \equiv E_1.\text{max}(E_2) \\
    \text{min}(E_1, E_2) & \text{if } E \equiv E_1.\text{min}(E_2) \\
    \text{at}(X) & \text{if } E \equiv X.\text{at} \\
    \text{size}(S) & \text{if } E \equiv S->\text{size}() 
\end{cases}$$

Let $exp \equiv W$ be an OCL string expression. Then,

$$W \triangleq \begin{cases} 
    w & \text{if } W \equiv w \in \mathcal{S} \\
    \text{concat}(W_1, W_2) & \text{if } W \equiv W_1.\text{concat}(W_2) \\
    \text{substring}(W_1, E_1, E_2) & \text{if } W \equiv W_1.\text{substring}(W_2, E_1, E_2) \\
    \text{at}(X) & \text{if } W \equiv X.\text{at} 
\end{cases}$$

Let $exp \equiv S$ be an OCL collection expression. Then,

$$S \triangleq \begin{cases} 
    p(X) & \text{if } S \equiv X.p \\
    \text{collect}@S'(\text{self}, S) & \text{if } V \equiv S->\text{collect}(X \mid S') \\
    \text{select}@V(\text{self}, S) & \text{if } V \equiv S->\text{select}(X \mid V) \\
    \text{reject}@V(\text{self}, S) & \text{if } V \equiv S->\text{reject}(X \mid V) 
\end{cases}$$

Finally, we can define the function that represents invariants as formulas.

**Definition 7 (representation function for invariants).** Let $inv$ be an OCL invariant,

$$inv \equiv \text{context c inv id : exp},$$

over a class diagram structure $\mathcal{CD}$. The function $\overline{\text{inv}}$ represents $inv$ as a (universally quantified) formula over the theory $inv_{\mathcal{CD}}$,

$$\overline{\text{inv}} \triangleq \forall \{ \text{self} \} (\text{self}:c \implies \overline{\text{exp}} = \text{true}) ,$$

where self is a variable of kind Class.

### 5.2 Checking Invariants as Proofs

In this section, we first define the notion of invariant-object diagram theories as extensions of the invariant-class diagram theories: basically, sorts representing classes are filled with elements representing objects, and operators representing...
attributes and roles are defined for those elements. Then, we introduce a formal definition of checking invariants over object diagrams: it reduces to proving the formula representing the invariant in the initial model of the corresponding invariant-object diagram theory. Finally, we give a formal definition of test and validation cases.

**Definition 8 (invariant-object diagram theories).** Let $\mathcal{CD} = (\Omega_{\mathcal{CD}}, \Gamma_{\mathcal{CD}})$ be a theory specifying a class diagram structure $\mathcal{CD}$ and let $\mathcal{OCD} = (\Omega_{\mathcal{CD}} \cup \Omega_{\mathcal{OCD}}, \Gamma_{\mathcal{CD}} \cup \Gamma_{\mathcal{OCD}})$ be a theory specifying a $\mathcal{CD}$-object diagram structure. Also, let $\text{inv}$ be an OCL invariant over $\mathcal{CD}$. We define the following theory extension of $\text{inv}_{\mathcal{CD}}$

\[ \text{inv}_{\mathcal{OCD}} \equiv (\Omega_{\mathcal{OCD}} \cup \Omega_{\mathcal{col}} \cup \Omega_{\text{inv}_{\mathcal{CD}}}, \Gamma_{\mathcal{OCD}} \cup \Gamma_{\mathcal{col}} \cup \Gamma_{\text{inv}_{\mathcal{CD}}}). \]

We can now give a formal definition of test and validation cases.

**Definition 9 (test and validation cases).** Let $\mathcal{CD} = (\Omega_{\mathcal{CD}}, \Gamma_{\mathcal{CD}})$ be a theory specifying a class diagram structure $\mathcal{CD}$. Let $\{\text{inv}_{\mathcal{CD}}^i\}_{1 \leq i \leq k}$ be a set of OCL invariants over $\mathcal{CD}$. Then, $\mathcal{OCD}$ is a test case if

\[ \bigcup_{i=1}^{k} \text{inv}_{\mathcal{OCD}}^i \models \bigwedge_{i=1}^{k} \text{inv}_{\mathcal{CD}}^i, \]

where $\bigcup_{i=1}^{k} \text{inv}_{\mathcal{OCD}}^i$ denotes the union of the $\text{inv}_{\mathcal{OCD}}^i$ theories, for $1 \leq i \leq k$.

Also, let $\text{inv}_{\mathcal{CD}}^0$ be an OCL invariant over $\mathcal{CD}$. Then, $\text{inv}_{\mathcal{CD}}^0$ is a validation case if

\[ \bigcup_{i=1}^{k} \text{inv}_{\mathcal{OCD}}^i \models \left( \bigwedge_{i=1}^{k} \text{inv}_{\mathcal{CD}}^i \right) \Rightarrow \text{inv}_{\mathcal{CD}}^0. \]

### 6 The ITP/OCL Tool

In this section we briefly present the ITP/OCL tool. The latest version of the tool, with the available documentation and examples can be found at [http://maude.sip.ucm.es/itp/ocl/](http://maude.sip.ucm.es/itp/ocl/).

The ITP/OCL tool is being developed as an extension of the ITP tool [2], an inductive theorem prover for MEL specifications. When completed, the ITP/OCL tool will allow to automatically:

- parse OCL invariants over UML class diagram;
- check OCL invariants over UML object diagrams;
- evaluate OCL expressions over UML object diagrams;
- generate UML object diagrams for UML class diagrams; and
- find test cases for UML class diagrams.

The ITP/OCL tool logic is directly based on the algebraic semantics for UML+OCL class diagrams presented in this paper:
class and object diagrams are specified as membership equational theories;
invariants are represented as formulas over the corresponding extensions of
those theories; and
checking invariants is reduced to proving the corresponding formulas in the
initial model of the corresponding invariant theory extensions.

Checking invariants over object diagrams can be automatized because:

the invariant theory extensions are Church-Rosser and terminating; and
the sorts representing classes in the invariant theory extensions only contain
a finite number of elements representing objects.

Finding test cases for class diagrams can be also automatized. However, note that
the (growing) complexity of the general problem suggests to limit in practice the
search space.

Complexity study Let $O_\mathbb{CD}$ be an object diagram structure, and let $p \in ROL_{(c,c')}$
be the role played by objects in a class $c$ in an association between classes $c$ and $c'$. Let $|c|$ and $|c'|$ be, respectively, the number of objects in the classes $c$ and $c'$.
Then, the number of all the possible snapshots that can be generated by just
changing the links between the objects for this association is:

$$2^{|c|} \times \cdots \times 2^{|c'|} = 2^{|c'| + |c|}.$$

Of course, if we consider all the roles $p_i \in ROL_{(c_i,c'_i)}$ played by the objects,
the number of all the possible snapshots that can be generated by just changing
the links between the objects, for $k$ different roles, is

$$\prod_{i=1}^{k} 2^{|c_i| + |c'_i|} = 2^{\sum_{i=1}^{k} |c_i| + |c'_i|}.$$

7 Related Work

In general, UML diagrams are carefully designed and represented using CASE
tools. Unfortunately, the UML static semantics expressed using OCL has many
syntactical, semantical and conceptual errors. This is mainly due to the lack of
appropriate OCL tool support. The following tools are trying to overcome this
problem.

- USE [12] is a system for the specification of information systems. A USE
specification contains a textual description of a model using features found in
UML class diagrams (classes, associations, etc.). Expressions written in OCL
are used to specify additional integrity constraints on the model. A model
can be animated to validate the specification against non-formal requirements. System states (snapshots of a running system) can be created and
manipulated during an animation. For each snapshot the OCL constraints are automatically checked. Information about a system state is given by graphical views. OCL expressions can be entered and evaluated to query detailed information about a system state. Due to the semi-formal definition of OCL there are some language constructs whose interpretation is ambiguous or unclear [8]. A formalization of OCL which attempts to provide a solution for most of the problems is proposed in [14, 15]. The USE approach for testing and validation is described in [16, 13].

- OCLE [4] is a CASE UML tool that allows checking UML models against the Well Formedness Rules, Methodological Rules, Profile Rules and Target Implementation Language Rules expressed in OCL. It has a graphical interface conceived and implemented with the aim of supporting the use of OCLE in a natural and intuitive manner.

- Oclarity [7] is an add-in for Rational Rose that offers comprehensive support for OCL. It provides integration with Rational Rose, supports constraints, initialization expressions, derivation rules and method-body definitions. It has editing capabilities with syntax highlighting, multi level undo/redo, search and replace and easily accessible code check.

- The Dresden OCL Toolkit [5] consists of a Common OCL Metamodel, a Metadata Repository (MDR), a JMI-based (meta-) model API, an OCL Standard Library, and an OCL Parser. The OCL Toolkit has been integrated into the open-source CASE tool ArgoUML. Poseidon is the commercial successor of ArgoUML and also contains the integration of the Dresden OCL Toolkit just as its predecessor does.

8 Conclusions

We propose an algebraic semantics for UML+OCL class diagrams that provides a formal foundation for automatically validating UML object diagrams with respect to OCL constraints. We use an expressive version of equational logic to specify class and object diagrams, and we formalize OCL constraints as formulas over those theories. As a result, validating UML object diagrams with respect to OCL constraints is reduced to proving invariant formulas over the standard models of invariant object diagram theories. Based on this semantics, we are developing a tool, named ITP/OCL, that will provide automatic generation and validation of object diagrams with respect to OCL constraints. The tool is intended as a lightweight formal method: it will help to find defects in UML class diagrams in the early phases of the software developing process.

When validating a model, we can in many cases concentrate on a single system state given at a discrete point in time. For example, a system state provides the complete context for the evaluation of class invariants. For pre- and post-conditions, however, it is necessary to consider two consecutive states. We plan to extend our formalization to other UML diagrams, as statecharts, which capture behavioural constraints.
References

A Class Diagram Example

Consider the following class diagram $TW$. It shows an example from a railway context: train may own wagons, and wagons may be connected to other wagons (their predecessor and successor wagons).

The class diagram theory $TW = (\Omega_{TW}, \Gamma_{TW})$ specifies $TW$ as a theory in MEL.

An Object Diagram Example

Consider now the following object diagram $TW(1)$ for the class diagram $TW$:

The object diagram theory $TW(1) = (\Omega_{TW} \cup \Omega_{TW(1)}, \Gamma_{TW} \cup \Gamma_{TW(1)})$ specifies $TW(1)$ as a theory in MEL.
\[ \Sigma_{\text{TW}(1)} \] contains the following set:
- \( \Sigma_{\text{Train, Class}} = \{ \text{Train1, Wagon1, Wagon2} \}. \]

\( \Gamma_{\text{TW}(1)} \) contains the axioms in the following sets:
- \( \Gamma_{\text{TW}(1)} \), which contains the following axioms:
  * \( \text{Train1: Train} \),
  * \( \text{Wagon1: Wagon} \),
  * \( \text{Wagon2: Wagon} \).
- \( \Gamma_{\text{ATT TW}(1)} \), which contains the following axioms:
  * \( \text{identifier(Train1)} = \text{“HS7456SP”} \),
  * \( \text{smoking(Wagon1)} = \text{true} \),
  * \( \text{smoking(Wagon2)} = \text{false} \).
- \( \Gamma_{\text{ROL TW}(1)} \), which contains the following axioms:
  * \( \text{wagon(Train1)} = \text{col(Wagon1,\ col(Wagon2, nil))} \),
  * \( \text{train(Wagon1)} = \text{col(Train1, nil)} \),
  * \( \text{train(Wagon2)} = \text{col(Train1, nil)} \),
  * \( \text{pred(Wagon1)} = \text{nil} \),
  * \( \text{pred(Wagon2)} = \text{col(Wagon1, nil)} \),
  * \( \text{succ(Wagon1)} = \text{col(Wagon2, nil)} \).

An Invariant Example

Finally, consider the following constraint over the class diagram \( \text{TW} \): all the trains’ wagons are smoking. It can be expressed in OCL as an invariant over \( \text{TW} \):

\[
\text{context Train inv smokingWagons:}
\text{self.wagon->forall(w | w.smoking)}
\]

The invariant-theory for \( \text{smokingWagons} \) contains the following axioms defining the operator \( \text{forall}@\text{smoking} \) which represents the iterator expression \( \forall\forall\forall\forall\big( w \mid w.\text{smoking} \big) \).

(1) \( \forall\{x,u\}\{x: \text{Train} \land u: \text{Col[\text{Wagon}]} \Rightarrow \text{forall}@\text{smoking}(x, u): \text{Boolean} \}. \)

(2) \( \forall\{x,u\}\{x: \text{Train} \land u: \text{Col[\text{Wagon}]} \land \text{isEmpty}(u) = \text{true} \Rightarrow \text{forall}@\text{smoking}(x, u) = \text{true} \}. \)

(3) \( \forall\{x,u\}\{x: \text{Train} \land u: \text{Col[\text{Wagon}]} \land \text{isEmpty}(u) = \text{false} \land \text{smoking(getOne}(u)) = \text{true} \Rightarrow \text{forall}@\text{smoking}(x, u) = \text{false} \}. \)

(4) \( \forall\{x,u\}\{x: \text{Train} \land u: \text{Col[\text{Wagon}]} \land \text{isEmpty}(u) = \text{false} \land \text{smoking(getOne}(u)) = \text{false} \Rightarrow \text{forall}@\text{smoking}(x, u) = \text{false} \}. \)

Finally, the invariant \( \text{smokingWagons} \) can be represented by the following formula over the invariant-theory for \( \text{smokingWagons} \):

\[ \forall\{\text{self}\}\{\text{self: Train} \Rightarrow \text{forall}@\text{smoking}(\text{self, smoking(wagon(self))) = true}, \]

where \( \text{self} \) is a variable of kind \( \text{Class} \).