On verifying ATL transformations using 'off-the-shelf' SMT solvers

Fabian Büttner[1], Marina Egea[2], Jordi Cabot[1]

[1] AtlanMod Research Group, INRIA / Ecole des Mines de Nantes
{fabian.buettner,jordi.cabot}@inria.fr
[2] Atos, Madrid
marina.egea@atosresearch.eu

Abstract. MDE is a software development process where models constitute pivotal elements of the software to be built. If models are well-specified, transformations can be employed for various purposes, e.g., to produce final code. However, transformations are only meaningful when they are 'correct': they must produce valid models from valid input models. A valid model has conformance to its meta-model and fulfils its constraints, usually written in OCL. In this paper, we propose a novel methodology to perform automatic, unbounded verification of ATL transformations. Its main component is a novel first-order semantics for ATL transformations, based on the interpretation of the corresponding rules and their execution semantics as first-order predicates. Although, our semantics is not complete, it does cover a significant subset of the ATL language. Using this semantics, transformation correctness can be automatically verified with respect to non-trivial OCL pre- and postconditions by using SMT solvers, e.g. Z3 and Yices.

1 Introduction

In Model-Driven Engineering (MDE), models constitute pivotal elements of the software to be built. When they are sufficiently well specified, model transformations can be employed for different purposes, e.g., to produce actual code. However, it is essential that such transformations are *correct* if they are to play their key role. Otherwise, errors introduced by transformations will be propagated and may produce more errors in the subsequent MDE steps. Thus, well-founded and, at the same time, practical verification methods and tools are important to guarantee this correctness.

Our work focuses on checking partial correctness of *declarative, rule-based transformations* between *constrained metamodels*. More specifically, we regard the ATL transformation language [17] and MOF [22] style metamodels that employ OCL constraints to precisely describe their domain. These ingredients are very popular due to their sophisticated tool support, and because OCL is employed in almost all OMG specifications. Several notions of correctness apply to such model transformations, like termination or confluence (see, e.g., [19,14]). In this paper, we are interested in a Hoare-style notion of partial correctness, i.e., in the correctness of a transformation with respect to certain sets of pre- and postconditions. In other words, we are interested in whether the output model produced by an ATL transformation for any valid input model is valid, too. A valid model is one that has conformance to its metamodel and fulfils its

constraints, usually written in OCL [21], that is, the OMG standard constraint language for models. We present a novel approach that targets *automatic* and *unbounded* verification of this property for ATL transformations. Our aim is to provide a 'push button' technology that can be applied regularly in model transformation development by developers lacking of a formal background.

The key components of our approach are a novel first order semantics for a declarative subset of ATL, based on the interpretation of ATL rules as first-order functions and predicates, and the use of automatic decision procedures for Satisfiability Modulo Theories problems in SMT solvers. Such solvers, e.g. Z3 [12,28] and Yices [13,27], have been significantly improved in the past years and can automatically and efficiently decide several important fragments of first-order logic (FOL). Our approach combines the advantages of formal verification (in the sense that we aim to provide formal proofs) and automatic verification (in the sense that we do not require the transformation developer to operate, for example, interactive theorem provers). To our knowledge, we are the first ones in proposing a first order semantics for a declarative subset of ATL which, in particular, allows the automatic unbounded verification of transformations between metamodels that may be constrained using OCL (more precisely, the subset of OCL that is considered in [11]).

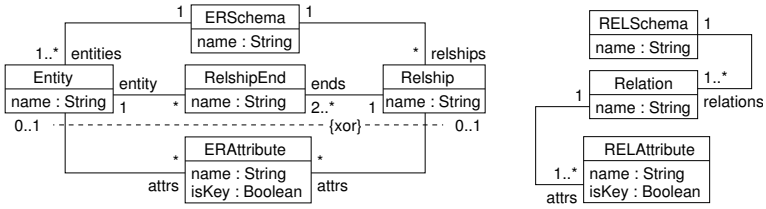
In addition to the running example that we use in this paper, we have made also available at [8], for the interested reader, the formalization, according to our FOL semantics, of a larger and more complex example. In this example we illustrate how we deal with type inheritance hierarchies and more complex and overlapping input patterns that have to be resolved by type checking and filtering conditions resolution. Also, we deal with more complex bindings statements in the output patterns for this transformation. This example was borrowed from [5].

Organization. In Sect. 2 we present our running example. Sect. 3 and Sect. 4 describe our FOL formalization for metamodels and ATL rules. In Sect. 5 we formalize our Hoare-style notion of partial correctness and how it can be checked using SMT solvers. We discuss related work in Sect. 6, and we conclude and outline future work in Sect. 7.

2 Running Example

Figure 1 depicts the ER and REL metamodels that are (resp.) the source and target metamodels for the ER2REL transformation, which is depicted in Fig. 2. In the ER metamodel, a schema may have entities and relationships (relships), both may contain attributes, and attributes may be keys; in the REL metamodel, a schema may have relations, which may have again attributes.[3] We only provide here an informal description of ER2REL, its precise semantics is discussed later in Sect. 4. Additional information on ATL can be found at [17,3]. In a nutshell, the ER2REL transformation takes an instance of the ER metamodel as input and produces an instance of the REL metamodel following the transformation in Fig. 2. This transformation is described by *matched rules*, which are the workhorse of ATL. Matched rules define a pattern of input types and possibly a filter

[3] For simplicity, we refer to the metamodel elements as schemas, entities, relationships, etc., instead of using schema type, entity type, etc..



(a) ER (b) REL

Fig. 1. ER and REL metamodels

```
module ER2REL; create OUT : REL from IN : ER;

rule S2S { from s : ER!ERSchema to t : REL!RELSchema (name <- s.name) }

rule E2R { from s : ER!Entity
           to t : REL!Relation (name<-s.name, schema<-s.schema) }

rule R2R { from s : ER!Relship
           to   t : REL!Relation (name <-s.name, schema<-s.schema) }

rule EA2A { from att : ER!ERAttribute, ent : ER!Entity (att.entity=ent)
            to   t   : REL!RELAttribute
                       (name<-att.name, isKey<-att.isKey, relation<-ent ) }

rule RA2A { from att : ER!ERAttribute, rs  : ER!Relship (att.relship=rs)
            to   t   : REL!RELAttribute
                       (name<-att.name, isKey<-att.isKey, relation<-rs) }

rule RA2AK { from att : ER!ERAttribute, rse : ER!RelshipEnd
                        (att.entity=rse.entity and att.isKey=true)
             to   t   : REL!RELAttribute
                        (name<-att.name, isKey<-att.isKey, relation<-rse.relship)}
```

Fig. 2. The ATL transformation ER2REL

expression (the `from`-clause). Each rule is applied to each matching set of objects in the input model to create the objects in the target model that are described in the `to`-clause, assigning values to their properties (typically) based on the input objects' properties.

The first rule in Fig. 2, S2S, maps ER schemas to REL schemas, the second rule E2R maps each entity to a relation, and the third rule R2R maps each relationship to a relation. The remaining three rules generate attributes for the relations. Both, entity and relationship attributes are mapped to relation attributes (rules EA2A and RA2A). Furthermore, the key attributes of the participating entities are mapped to relation attributes as well (rule RA2AK). Notice that in the property assignment, a so-called *implicit resolution* step is needed to resolve source objects to target objects: For example the binding `schema<-s.schema` in E2R and R2R 'silently' replaces the `ERschema` value of `s.schema` by the `RELSchema` object that is created for `s.schema` by S2S. Fig. 3 shows a list with the OCL constraints for the source and target metamodels. The constraints require the expected uniqueness of names within their scopes (e.g., for the entities in a schema), and the existence of key attributes in entities and relations. In addition to the constraints in Fig. 3, multiplicity constraints are encoded as OCL constraints for

both metamodels, too. For every lower bound different from 0 and each upper bound different from * we add one OCL constraint named ⟨qualified rolename⟩.lo (for lower) resp. ⟨qualified rolename⟩.up (for upper). E.g., the constraint `Entity::schema.lo` is `context Entity inv: self.schema->size()>=1`, and the constraint for the upper bound is `Entity::schema.up` is `context Entity inv: self.schema->size()<=1`.

```
context ERSchema inv pre1:       -- unique schema names
ERSchema.allInstances()->forall(s1,s2| s1<>s2 implies s1.name<>s2.name)

context ERSchema inv pre2:       -- relship names are unique in schema
self.relships->forAll(r1,r2 | r1<>r2  implies  r1.name<>r2.name)

context ERSchema inv pre3:       -- entity names are unique in schema
self.entities->forAll(e1,e2 | e1<>e2 implies e1.name<>e2.name)

context ERSchema inv pre4:       -- disjoint entity and relship names
self.relships->forAll(r | self.entities->forAll(e |  r.name<>e.name))

context EREntity inv pre5:       -- attr names are unique in entity
self.attrs->forAll(a1,a2 | a1.name=a2.name implies a1=a2)

context ERRelship inv pre6:      -- attr names are unique in relship
self.attrs->forAll(a1,a2 | a1.name = a2.name implies a1=a2)

context Entity inv pre7:         -- entities have a key
self.attrs->exists(a | a.isKey)
-- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

context RELSchema inv post1:     -- unique schema names
RELSchema.allInstances()->forall(s1,s2| s1<>s2 implies s1.name<>s2.name)

context RELSchema inv post2:     -- relation names are unique in schema
relations->forall(r1,r2| r1<>r2 implies r1.name<>r2.name)

context Relation inv post3:      -- attribute names  unique in relation
self.attrs->forAll(a1,a2 | a1.name=a2.name implies a1=a2)

context Relation inv post4:       -- relations have a key
self.attrs->exists(a | a.isKey)
```

Fig. 3. OCL constraints for ER and REL

Problem Statement. A developer who is designing a model transformation typically wonders the following question several times during the designing process: *Do the constraints imposed on the source model plus the transformation specification guarantee that these other constraints are fulfilled by the target models?* When the answer to this question is *'yes'* for certain properties, we would say that the transformation which is being designed is correct with respect to the given sets of pre and postconditions. Namely, in our view, a model transformation is *correct* if and only if executing it using a constrained-valid input model as argument always results on a constrained-valid output model, where a constrained-valid input model is a model that satisfies the model transformation's *preconditions* and a constrained-valid output model is a model that satisfies the model transformation's *postconditions*. Notice that the ATL model transformations that we consider here are always executed to populate target models which are initially empty. For our approach to be practical, we are implementing a tool that automatically maps ATL matched rules to first order logic files and employs the Z3 solver

to check whether the implications between pre and postconditions hold. A tool that automatically maps the OCL constrained metamodels to FOL is already implemented.

We can read each row in Table 2 (left hand side) as follows: For each input model which satisfies the named preconditions, the respective postcondition will hold for the output model. These implications were proven automatically to hold for the ER2REL transformation by Z3 and Yices.[4] Moreover, the SMT solvers can also determine the minimal set of preconditions that are required to prove a given postcondition. The table shows that every target postcondition of the REL metamodel can be inferred except post3 – for which Z3 can find a counter example even if all preconditions are assumed, e.g. an input model containing reflexive relationships on the source side. In the following sections, we will explain our FOL semantics for ATL matched rules that allow these implications to be automatically proven by the SMT solvers.

Table 1. Implications that hold for ER2REL (and that can be proven automatically using Z3 and Yices* using our translation into first-order logic). (QI=Quantifier Instantiations). (Class names abbreviations: E = Entity, RS = Relship, RSE = RelshipEnd, RE= Relation, RA = RELAttribute)

Proofs found automatically by Z3 and Yices*					
Preconditions	Postcondition	Unsat core total = 69	QI (C)	QI (U)	QI (P)
pre1	post1	4	22	18	22
E::schema.lo, RS::schema.lo	R::schema.lo	9	186	30	60
E::schema.up, RS::schema.up	R::schema.up	9	310	118	200
pre2, pre3, pre4, E::schema.lo+up, RS::schema.lo+up	post2	16	10274	888	423
RSE:relship.lo	RA::relation.lo	11	359	50	105
RSE::relship.lo+up	RA::relation.up	11	2864	72	247
pre4, RSE::type.lo, RS::ends.lo	post4	14	493	141	235

3 Mapping Metamodels and OCL constraints to First Order Logic

Since ATL transformations are always defined from a source to a target metamodel, we will first explain how we map metamodels' elements to first order logic. We had already used this first order formalization in [11].

- *Type-predicates:* Metamodels' classes are mapped to unary boolean functions. E.g., the class ERSchema is mapped to a unary boolean function ERSchema: $int \rightarrow bool$;
- *Objects variables* are mapped to integer variables, e.g, an object variable cl of type Entity is mapped to an integer variable cl, such that Entity(cl) holds;

[4] We put a '$*$' to Yices in the table since it requires to assume some previously proven postconditions as lemmas in order to find a proof for R::schema.lo and RA::relation.lo

- *Attribute-functions:* Attributes are mapped to either boolean or integer functions, e.g., the attribute `name` is mapped to a function `name`: $int \rightarrow int$;[5]
- *Association-predicates:* Association-ends of a given association, e.g. `erschema` are mapped to binary boolean functions, e.g., `erschema`: $int\ int \rightarrow bool$.

Notice that we assume that a function $\lceil . \rceil$ exists to generates unique function symbols for metamodel elements, and that metamodel elements are uniquely named (without losing generality). Next, we outline how we can ensure disjointness of types in the type system, and how to consider inheritance relationships, that were not covered before in our approach. Our extension includes formulas

- to ensure that those type-predicates mapping classes that are not subclasses in any inheritance relationship are pairwise disjointly interpreted. I.e., if c and c' are disjoint classes, we include a formula $\forall(x)\ \neg(\lceil c \rceil(x) \wedge \lceil c' \rceil(x))$ to ensure that their corresponding predicates are disjointly interpreted;
- to map class inheritance relationships. Namely, for each direct subclass relations between c' and c, i.e., c' is a (direct) subclass of c, we include a formula $\forall(x)(\lceil c' \rceil(x) \Rightarrow \lceil c \rceil(x))$. For each abstract superclass c, the following formula would also be included: $\forall(x)(\lceil c \rceil(x) \Rightarrow \bigvee_{1 \leq h \leq k} \lceil c_h \rceil(x))$ for all c_h subclass of c.

Notice that we need to add these axioms explicitly because we do not use sorted logic as, e.g., [25]. In the case of inheritance, only for the immediate subtypes below a superclass in different branches of the tree, the formulas to guarantee the pairwise disjoint interpretation of these types are included.

Remark 1. Neither superclass attributes nor associations ends are specified explicitly for the subclasses, but they are mapped and used according to how they are specified in the metamodel for the superclass. We do not consider multiple inheritance relationships.

From OCL to First-Order Logic. As we mentioned before, this work focuses on the correctness of transformations defined between OCL constrained metamodels. We use our previous work [11] to translate OCL constraints into FOL. The operators that are listed in the examples below are those covered in [11]. [6] Our mapping is both simple, in the sense that the resulting FOL formulas closely mirror the original OCL constraints; and practical, in the sense that, using this mapping we can also employ automated theorem provers and/or SMT solvers (e.g., Z3 and Yices) to automatically perform unsatisfiability checks on non-trivial sets of OCL constraints. In a nutshell, our mapping is defined recursively over the structure of OCL expressions. Attributes, classes and association ends that may be part of OCL expressions are mapped as we explained for metamodel elements.

[5] We do not considered attributes' values of type object or collection. Also, strings are currently treated as Integers. Thus, no string-specific operations are supported. The reason is that there are no such theories and decision procedures available yet for SMT solvers (although this is ongoing work).

[6] Our mapping is not yet complete but it does cover a sufficiently significant subset of the OCL language.

- `Boolean`-expressions are translated to formulas, which essentially mirror the logical structure of the OCL expressions, e.g., for the operations `or`, `and`, `implies`, `not`, `isEmpty()`, `notEmpty()`, `includes`, `excludes`, $<$, $>$, $\leq$, $\geq$, $=$, $\neq$;
- `Integer`-expressions are basically copied, e.g. $+$, $-$, $*$; currently, we do not cover `String`-expressions.
- `Collection`-expressions are translated to *fresh* predicates that augment the specification signature and whose meaning is defined by additional formulas generated by the mapping. E.g. to map `select`, `reject`, `collect`, `forAll`, `exists`, `including` or `excluding` operations.

Example 1. Mapping precondition 1 using our previous translation from OCL to FOL. Precondition 1 in the example presented in Sect. 2 is:

```
inv pre1:
ERSchema.allInstances()->forall(x,y| x<>y implies x.name<>y.name)
```

It is mapped to:

$$\forall(x)(\mathsf{ERSchema}(x) \Rightarrow \forall(y)(\mathsf{ERSchema}(y) \Rightarrow ((x \neq y) \Rightarrow (\mathsf{name}(x) \neq \mathsf{name}(y)))))$$

4 First-Order Semantics for ATL transformations

There are two things that need to be considered in order to understand the meaning of a model-to-model transformation and how it works. One is the language in which it is specified, the other is how the transformation definition is executed by a transformation engine. Therefore, to be able to reason about pre- and postconditions that may hold for an ATL transformation, our first order interpretation of ATL transformations is capturing in addition to the definition of the rules, how the ATL engine executes them. For the work presented here, we assume that the ATL transformation parses and type checks correctly regarding the source and target metamodels, and that its execution does not end in abortion or error, i.e., a valid output model is produced from any valid input model. Currently, we only regard matched rules, in a slightly restricted form that allows only one output pattern element per rule and three kinds of bindings, as it is captured in Fig. 4. But our mapping can be extended to deal with more than one output pattern elements and to cover OCL collection expressions that can be used on the right-hand side of binding statements. Lazy rules will be included (with some restrictions), in future work. Matched rules' patterns (up to name uniqueness) always compose ATL transformations that are terminating and confluent [18]. Furthermore, we only support the subset of OCL that is supported in [11]. In particular, we do not support recursively defined OCL operations that would be the only source of non-termination. Last, we do not allow that both ends of an association are used as target of bindings at the same time, because ATL does not guarantee confluence in this case. The structure of ATL matched rules was briefly explained in Sect. 2. Fig. 4 shows the pattern of matched rules that our mapping currently supports.

The object variables and the OCL expression appearing in the `from`-clause is called the rule's source pattern. The object variable and the binding statements appearing in the `to`-clause are called the rule's target pattern. Recall that the $oclexp$ in the source pattern is a boolean filtering condition (if there is not filtering condition, it is assumed

to be $true$). The expressions $bindstm_i$ are binding statements and s_j and o are object variables of types t_j (of the source metamodel) and t' (of the target metamodel), resp.. The properties $attname'_j$, $assocend'_k$ or $assocend'_l$ are (resp.) attribute's names and association ends that belong to t' objects according to the target metamodel definition. Analogously, $attname_{r_k}$ and $assocend_{f_p}$ are attribute's names and association ends that belong to t_r and t_f objects (resp.) according to the source metamodel definition. Moreover, attributes must be of integer or boolean types and their type must conform when they are bound by a rule, e.g. $attname'_j \leftarrow s_r.attname_{r_k}$. We assume that the function $\lceil . \rceil$ introduced in Sect. 3 also produces unique function symbols for ATL rules. These functions are declared with the arity that corresponds to the rules they mirror. Next we give a closer look on how the ATL engine executes these rules. We take advantage of this description to explain how the properties of the execution semantics of ATL rules are captured by our formalization in first order logic. As expected the ATL engine interprets ATL rules oriented from source to target.

`rule` $rlname$
  `from` $s_1 : t_1, \ldots, s_n : t_n$ $(oclexp)$
  `to` $o : t'$ $(bindstm_1, \ldots, bindstm_m)$
where each $bindstm_i$ can have one of the following shapes:
  shape I: $attname'_j \leftarrow s_r.attname_{r_k}$,
  shape II: $assocend'_l \leftarrow s_f.assocend_{f_p}$
  shape III: $assocend'_k \leftarrow s_v$

Fig. 4. ATL matched rule's pattern currently supported by our mapping

1. Objects in the target metamodel exists only if they have been created by an ATL rule since the ATL transformations that we consider are always initially executed on an empty target model. Namely, the ATL transformation considered as an operation from a source to a target model is surjective on rules' target object variables' types. When an object type can be generated by the execution of more than one rule of an ATL transformation, then a disjunction considering all these possibilities is made in the consequent of the assertion. E.g., if an object o of type t' can be created by any of the rules $rlname_1$ of input parameters $s_{11} : t_{11}, \ldots, s_{1v_1} : t_{1v_1}$ and filtering expression $oclexp_1$, $rlname_2$ of input parameters $s_{21} : t_{21}, \ldots, s_{2v_2} : t_{2v_2}$ and filtering expression $oclexp_2$, ..., and $rlname_k$ of input parameters $s_{k1} : t_{k1}, \ldots, s_{kv_k} : t_{kv_k}$ and filtering expression $oclexp_k$, formulas shaped as shown in Fig. 5, pattern (i), are generated. This type of formulas is inserted for each target object type of rules in the transformation.[7] Corresponding formulas instantiated for the ER2REL example presented in Sect. 2 are shown in Figs. 7, 9, 10, assertions (i).

[7] The function ocl2fol represents the mapping from OCL to first order logic defined in [11] and described in Sect. 3. It will produce a conjunction of boolean formulas when applied to an OCL boolean expression.

$$
\begin{aligned}
(i)\ &\forall(o)\ (\lceil t' \rceil(o)) \Rightarrow \exists(s_{11},\ldots,s_{1v_1})\ (\lceil t_{11} \rceil(s_{11}) \wedge \ldots \wedge \lceil t_{1v_1} \rceil(s_{1v_1})\ \wedge \mathsf{ocl2fol}(oclexp_1) \wedge \\
&\quad (\lceil rlname_1 \rceil(s_{11},\ldots,s_{1v_1}) = o)) \vee \\
&\quad \exists(s_{21},\ldots,s_{2v_2})\ (\lceil t_{21} \rceil(s_{21}) \wedge \ldots \wedge \lceil t_{2v_2} \rceil(s_{2v_2})\ \wedge \mathsf{ocl2fol}(oclexp_2) \wedge \\
&\quad\quad (\lceil rlname_2 \rceil(s_{21},\ldots,s_{2v_2}) = o)) \vee \\
&\quad\quad \ldots\ldots\ldots\ldots \\
&\quad \exists(s_{k1},\ldots,s_{kv_k})\ (\lceil t_{k1} \rceil(s_{k1}) \wedge \ldots \wedge \lceil t_{kv_k} \rceil(s_{kv_k})\ \wedge \mathsf{ocl2fol}(oclexp_k) \wedge \\
&\quad\quad (\lceil rlname_k \rceil(s_{k1},\ldots,s_{kv_k}) = o))
\end{aligned}
$$

Fig. 5. Formulas to capture that ATL transformations are surjective on target object variables' types

2. A rule's source pattern is matched (taking into account the filtering condition) against the elements of the source model (see assertion pattern (ii) in Fig. 6). Elements in the target model are created by the execution of at most one rule using a tuple of input objects that cannot be matched by two different rules (see assertion pattern (iii) in Fig. 6). Namely, an ATL transformation from source to target is executed as a function and, in addition, it is globally injective. To ensure that a target object can only be produced by one rule on a fixed tuple of arguments, we introduce a function creation. It assigns to each target object the rule identifier (which is a constant defined exactly for this purpose) and the input object pattern that created it. In order to have an homogeneous signature of this function, we assume the maximum input pattern arity u (plus 1 to insert the rule identifier). For rules with a lesser arity $v < u$, the tuple is completed with arbitrary object variables that appear existentially quantified. This definition, shown in Fig. 6, represents a simple way for ensuring global injectivity for the transformation. These type of formulas are also inserted for every ATL rule in the transformation. These formulas instantiated for the ER2REL example presented in Sect. 2 are shown in Figs. 7, 9, 10, assertions (ii)-(iii).

$$
\begin{aligned}
(ii)\ &\forall(s_1,\ldots,s_n)(\lceil t_1 \rceil(s_1) \wedge \ldots \wedge \lceil t_n \rceil(s_n) \wedge \mathsf{ocl2fol}(oclexp)) \Rightarrow \\
&\quad \exists(o)\ (\lceil t' \rceil(o) \wedge (\lceil rlname \rceil(s_1,\ldots,s_n) = o)) \\
(iii)\ &\forall(s_1,\ldots,s_n,o)(\lceil t_1 \rceil(s_1) \wedge \ldots \wedge \lceil t_n \rceil(s_n) \wedge \mathsf{ocl2fol}(oclexp) \wedge \\
&\quad (\lceil rlname \rceil(s_1,\ldots,s_n) = o)) \Rightarrow \\
&\quad \exists(y_1,\ldots,y_d)(\lceil creation \rceil(o) = \langle idrlname, s_1,\ldots,s_n,y_1,\ldots,y_d \rangle),\ \text{with } d\text{=}(u\text{-}1)\text{-}n
\end{aligned}
$$

Fig. 6. Formulas to capture the rules' source patterns' matching and rule's inyectivity

3. The *bindings* of the target patterns are performed straight-forwardly for attribute's values (binding pattern shape I in Fig. 4) of primitive types. However, an implicit resolution strategy is applied by the ATL engine to resolve source objects to target objects. This mechanism is in place for shapes II and III of the binding statements given in Fig. 4). Recall that the transformations we assume as our subject of study can be successfully executed. In particular, this means that all the bindings declared in the target model are well defined, i.e., can be performed. To mirror the binding mechanism, auxiliar functions are defined. There will be as many of these functions as different rules' arities are

$(i)\ \forall(s)(\mathsf{RELSchema}(s) \Rightarrow \exists(p)(\mathsf{ERSchema}(p) \wedge (\mathsf{S2S}(p) = s)),$
$(ii)\ \forall(p)(\mathsf{ERSchema}(p) \Rightarrow \exists(s)\ (\mathsf{RELSchema}(s) \wedge (\mathsf{S2S}(p) = s)),$
$(iii)\forall(p,s)(\mathrm{ERSchema}(p) \wedge \mathrm{RELSchema}(s) \wedge (\mathrm{S2S}(p) = s)) \Rightarrow$
$\exists(y)(\mathrm{creation}(s) = \langle idS2S, p, y\rangle),$
(SI) $\forall(p,s)(\mathsf{ERSchema}(p) \wedge \mathsf{RELSchema}(s) \wedge (\mathsf{S2S}(p) = s)) \Rightarrow$
$(\mathsf{name}(s) = \mathsf{name}(p))$

Fig. 7. Formulas (i), (ii) and (iii) for the rule S2S

in the transformation. Fig. 8 illustrates how they are used, but we do not further explain here how they are defined for space reasons). Let us just say that we represent these functions with the symbol $\mathsf{resolve}_\mathsf{u}$, where u is the arity of the rules it is based on. It is defined as boolean function with $u + 1$ arity, and it helps to distinguish which is the rule resolving source to target objects. For the ATL transformation presented in Fig. 2, the function $\mathtt{resolve}_1$ defined as it is shown in Fig. 11 is used to resolve the binding patterns `erschema`←`er.relschema` of the rules `E2R` and `R2R`, and the binding patterns `relation`←`ent` of the rule `EA2A`, `relation`←`rs` and `relation`←`rse.relship` of the rules `RA2A` and `RA2AK` (resp.).

Formulas mapping binding statements are inserted *on-demand* for every ATL rule in the transformation. We formalized the mapping of binding statements of shape I-III in Fig. 8. Although we do not provide in this paper the definition of the function `type` over the metamodel (and it is used in the definition shown in Fig. 8), notice that it simply returns the type of the association end it is applied to. These formulas instantiated for the `ER2REL` example presented in Sect. 2 are shown in Fig. 7, 9, 10, assertions headed by (SI) to (SIII) for (Shape I) to (Shape III).

(Shape I)
$\forall(s_1,\ldots,s_n,o)(\lceil t_1\rceil(s_1) \wedge \ldots \wedge \lceil t_n\rceil(s_n) \wedge \mathsf{ocl2fol}(oclexp) \wedge (\lceil rlname\rceil(s_1,\ldots,s_n) = o))$
$\Rightarrow (\lceil attname'_j\rceil(o) = \lceil attname_{r_k}\rceil(s_r))$
(Shape II)
$\forall(s_1,\ldots,s_n,o)(\lceil t_1\rceil(s_1) \wedge \ldots \wedge \lceil t_n\rceil(s_n) \wedge \mathsf{ocl2fol}(oclexp) \wedge (\lceil rlname\rceil(s_1,\ldots,s_n) = o))$
$\Rightarrow \forall(w)(\lceil \mathtt{type}(assocend_{f_p})\rceil(w) \wedge \lceil assocend_{f_p}\rceil(s_f,w)) \Rightarrow$
$\exists(w')(\lceil \mathtt{type}(assocend'_l)\rceil(w') \wedge \lceil assocend'_l\rceil(o,w') \wedge resolve_1(w,w'))$
$\forall(w')(\lceil \mathtt{type}(assocend'_l)\rceil(w') \wedge \lceil assocend'_l\rceil(o,w'))$
$\Rightarrow \exists(w)(\lceil \mathtt{type}(assocend_{f_p})\rceil(w) \wedge \lceil assocend_{f_p}\rceil(s_f,w) \wedge resolve_1(w,w'))$
(Shape III)
$\forall(s_1,\ldots,s_n,o)(\lceil t_1\rceil(s_1) \wedge \ldots \wedge \lceil t_n\rceil(s_n) \wedge \mathsf{ocl2fol}(oclexp) \wedge \lceil rlname\rceil(s_1,\ldots,s_n) = o)$
$\Rightarrow \exists(w')(\lceil \mathtt{type}(assocend'_k)\rceil(w') \wedge \lceil assocend'_k\rceil(o,w') \wedge resolve_1(s_v,w')) \wedge$
$\forall(w')((\lceil \mathtt{type}(assocend'_k)\rceil(w') \wedge \lceil assocend'_k\rceil(o,w')) \Rightarrow resolve_1(s_v,w')))$

Fig. 8. Formulas for the bindings of the rules

$(i)\ \forall(t)(\mathsf{Relation}(t) \Rightarrow$
$\quad \exists(e)(\mathsf{Entity}(e) \wedge (\mathsf{E2R}(e) = t))\ \vee\ \exists(rh)(\mathsf{Relship}(rh) \wedge (\mathsf{R2R}(rh) = t)),$
$(ii)\ \forall(e)(\mathsf{Entity}(e) \Rightarrow \exists(t)\ (\mathsf{Relation}(t) \wedge (\mathsf{E2R}(e) = t)),$
$(iii)\ \forall(t, e)(\mathsf{Entity}(e) \wedge \mathsf{Relation}(t) \wedge (\mathsf{E2R}(e) = t)) \Rightarrow$
$\quad \exists(y)(\mathrm{creation}(t) = \langle idE2R, e, y\rangle),$
(SI) $\forall(e, t)(\mathsf{Entity}(e) \wedge \mathsf{Relation}(t) \wedge (\mathsf{E2R}(e) = t)) \Rightarrow$
$\quad (\mathsf{name}(e) = \mathsf{name}(t)),$
(SII) $\forall(e, t)\ (\mathsf{Entity}(e) \wedge \mathsf{Relation}(t) \wedge (\mathsf{E2R}(e) = t)) \Rightarrow$
$\quad (\forall(p)\ (\mathsf{ERSchema}(p) \wedge \mathsf{erschema}(e, p)) \Rightarrow$
$\quad\quad \exists(s)\ (\mathsf{RELSchema}(s) \wedge \mathsf{relschema}(t, s) \wedge \mathsf{resolve}_1(p, s))) \wedge$
$\quad (\forall(s)\ (\mathsf{RELSchema}(s) \wedge \mathsf{relschema}(t, s)) \Rightarrow$
$\quad\quad \exists(p)\ (\mathsf{ERSchema}(p) \wedge \mathsf{erschema}(e, p) \wedge \mathsf{resolve}_1(p, s)))$

Fig. 9. Formulas (i), (ii) and (iii) for the rule E2R. Map of its binding statements of shape I-II.

5 Verifying Model Transformations

In this section we formalize the Hoare-style notion of correctness (i.e. Hoare triples) that we use to verify ATL model transformations. [8] In particular, Def. 1 follows standard Hoare logic in that it deals only with partial correctness, while termination would need to be proven separately. Notice however that the matched rules' patterns considered in this work (up to name uniqueness) always compose ATL transformations that are terminating and confluent since this type of rules do not contain any possible source of non-termination. Namely, they do not contain recursive calls, neither recursively defined OCL helper operations. This claim is further supported and explained in [18]. In addition, notice that for the ATL rules that we consider in this work, only two conditions can get an execution aborted: (a) Two rules match the same tuple of objects; (b) A binding of shape III (Fig. 4) cannot be resolved because the required object was not matched by any rule's source pattern. Neither (a) nor (b) happen in our examples.

Assuming that *ocl2fol* represents our mapping from OCL to first-order logic [11] described in Sect. 3, and that *atl2fol* is the mapping from ATL to first-order logic that we partially described in Sect. 4, we are able to formalize our notion of ATL model transformations correctness in Def. 1 in two alternative shapes. The former definition of correctness is usually more convenient for using theorem provers to prove postconditions and the latter definition of correctness allows us to reduce the problem of checking the correctness of an ATL model transformation to the problem of checking the unsatisfiability of a set of first-order sentences, which can be checked using an SMT solver. In fact, all correctness checks shown in Table 2 were automatically proven by Z3 and Yices, two modern SMT solvers. However, let us remark again that Yices required some postconditions previously proven as lemmas to find a proof for postconditions `R::Schema.lo` and `RA::relation.lo`, whereas the decision procedures of Z3 were

[8] Let us recall, informally, that these triples, i.e. $\{\Phi\}\ \mathcal{Q}\ \{\Psi\}$, with $\{\Phi\}$ and $\{\Psi\}$ being formulas in first order logic, mean that if $\{\Phi\}$ holds before the execution of $\mathcal{Q}$ and, if $\mathcal{Q}$ terminates, then $\{\Psi\}$ will hold upon termination.

(i) $\forall(t)$ ($\mathsf{RELAttribute}(t) \Rightarrow$
$\exists(at, e)$ ($\mathsf{ERAttribute}(at) \wedge \mathsf{Entity}(e) \wedge \mathsf{entity}(at, e) \wedge (\mathsf{EA2A}(e, at) = t)) \vee$
$\exists(at, rh)$ ($\mathsf{ERAttribute}(at) \wedge \mathsf{Relship}(rh) \wedge \mathsf{relship}(at, rh) \wedge (\mathsf{RA2A}(rh, at) = t)) \vee$
$\exists(at, rhe, e)$ ($\mathsf{ERAttribute}(at) \wedge \mathsf{RelshipEnd}(rhe) \wedge \mathsf{Entity}(e) \wedge \mathsf{entity}(at, e) \wedge$
$\mathsf{type}(rhe, e) \wedge (\mathsf{isKey}(at) = true) \wedge (\mathsf{RA2AK}(rhe, at) = t))$,
(ii) $\forall(e, at)(\mathsf{Entity}(e) \wedge \mathsf{ERAttribute}(at) \wedge \mathsf{entity}(at, e)) \Rightarrow$
$\exists(t)(\mathsf{RELAttribute}(t) \wedge (\mathsf{EA2A}(e, at) = t))$,
(iii) $\forall(e, at, t)(\mathsf{Entity}(e) \wedge \mathsf{ERAttribute}(at) \wedge \mathsf{RELAttribute}(t) \wedge (\mathsf{EA2A}(e, at) = t)) \Rightarrow$
$(\mathrm{creation}(t) = \langle idEA2A, e, at\rangle)$,
(SI) $\forall(e, at, t)(\mathsf{Entity}(e) \wedge \mathsf{ERAttribute}(at) \wedge \mathsf{RELAttribute}(t) \wedge (\mathsf{EA2A}(e, at) = t)) \Rightarrow$
$(\mathsf{name}(at) = \mathsf{name}(t))$,
(SIII) $\forall(e, at, t)(\mathsf{Entity}(e) \wedge \mathsf{ERAttribute}(at) \wedge \mathsf{RELAttribute}(t) \wedge (\mathsf{EA2A}(e, at) = t)) \Rightarrow$
$\exists(w')$ ($\mathsf{Relation}(w') \wedge \mathsf{relation}(t, w') \wedge \mathsf{resolve}_1(e, w'))) \wedge$
$\forall(w')$ ($\mathsf{Relation}(w') \wedge \mathsf{relation}(t, w') \Rightarrow \mathsf{resolve}_1(e, w'))$)

Fig. 10. Formulas (i), (ii) and (iii) for the rule `EA2A`, taking into account that `RELAttributes` can also be created by the rules `RA2A` and `RA2AK`. Map of its binding statements of shape I and III.

$$\mathsf{resolve}_1(x, y) =^{def.} ((\mathsf{ERSchema}(x) \wedge \mathsf{RELSchema}(y) \wedge (\mathsf{S2S}(x) = y)) \vee (\mathsf{EREntity}(x) \wedge \mathsf{RELRelation}(y) \wedge (\mathsf{E2R}(x) = y)) \vee (\mathsf{ERRelship}(x) \wedge \mathsf{RELRelation}(y) \wedge (\mathsf{R2R}(x) = y)))$$

Fig. 11. Definition of the auxiliar function $\mathtt{resolve}_1$

able to fully handle these cases without further help (we further discuss this generality aspect below).

Definition 1. *Let $\mathcal{Q} = \{r_1, \ldots, r_n\}$ be an ATL model transformation composed of matched rules (and free from OCL recursive helper operators). Then, $\mathcal{Q}$ is correct with respect to preconditions $\{\varsigma_1 \ldots \varsigma_l\}$ and postconditions $\{\tau_1, \ldots, \tau_w\}$ if and only if, upon termination of $\mathcal{Q}$, for every τ_i, $i = 1, \ldots, w$, the following formula always hold:*

$$\left(\bigwedge_{j=1}^{l} ocl2fol(\varsigma_j)\right) \wedge \left(\bigwedge_{j=1}^{n} atl2fol(r_j)\right) \Rightarrow ocl2fol(\tau_i) \tag{1}$$

or, equivalently, the following formula is unsatisfiable

$$\left(\bigwedge_{j=1}^{l} ocl2fol(\varsigma_j)\right) \wedge \left(\bigwedge_{j=1}^{n} atl2fol(r_j)\right) \wedge \neg(ocl2fol(\tau_i)) \tag{2}$$

The correctness of our approach obviously depends on the correctness of the mappings from OCL to FOL and from ATL to FOL, that is, on whether they correctly capture the semantics of OCL constraints and of ATL rules and rules' execution. These are certainly two challenging theoretical problems, whose solutions will require, first of all, well-defined, commonly accepted, formal semantics for OCL and ATL: none of

these are currently available. Still notice that our translation yields very intuitive formulas for anyone familiar with ATL, and so they are suited for validation by humans against the expected behaviour of ATL.

Automatic verification of transformation correctness. For the ER2REL example presented in Sect. 2, all implications summarized in Table 2 were automatically (and directly) proven by Z3 and (partly by) Yices in less than 1 second (in a standard 2.2 Ghz office laptop running Windows 7). Similarly, for the more complex example that we borrowed from [5], all postconditions that we required were proven automatically in less than a minute using Z3. Actually, they were proven in less than 10 seconds when previously proven postconditions (multiplicity constraints in the target metamodel) were used as lemmas. The preconditions that are required to find a proof can be automatically determined by the solvers as the *unsatisfiable cores*. The column '*unsat core*' in Table 2 shows for the ER2REL example the number of assertions (from the FOL specification that contains the ATL semantics and the mapped OCL constraints used as preconditions) that are required to prove a postcondition. The other columns show the number of *quantifier instantiations* required to perform the proof. These numbers are directly correlated to the numbers of ground terms created by Z3 by instantiating the universally quantified variables in our formulas. QI(C) is the number of instantiations when all assertions are enabled, QI(U) is the number of instantiation when we only leave active the assertions that belong to the unsatisfiable core (others are 'disabled'). Finally, QI(P) is the number of instantiations made when all assertions for the transformation semantics are considered together with only the required preconditions. The relation between the three columns shows that the solver benefits from reduced precondition sets. In total, the specification of the ER2REL example in FOL (accounting pre-conditions and ATL semantics assertions) consists of 69 formulas.

Z3 can also work as a counter example finder but, in general, its algorithms seem to be slower for that goal. We performed several experiments to test the efficiency of Z3 for counter example finding. For instance, since we knew that post3 for ER2REL did not hold from the assumed preconditions, we ask Z3 to find a counter example, however, Z3 took more than 2 hours for such task when we did not specify a maximum model extent. Nevertheless, we think that tools specially dedicated to finite model finding such as Alloy are better suited to perform exactly that task in less time. Our experience [9] leads us in this direction also, and we consider a matter of future work tailoring our semantics for these tools. They would provide the required complement to SMT solvers, for the satisfiable case.

Generality of our approach. Both examples (ER2REL and the one borrowed from [5]) can be automatically verified using Z3 and (partly) Yices (the interested reader can find the files containing the formalization of both examples ready to feed Z3 and Yices at [8]). But, as FOL is not decidable we cannot claim full generality for our approach. However, we expect our FOL mapping for ATL matched rules to fall in the scope of what can be solved by the model-based quantifier instantiation (MBQI) decision procedure of Z3 [15] that is refutationally complete for quantified formulas of uninterpreted and 'almost uninterpreted' functions. Yet, which part of the OCL language is decidable needs to be investigated.

6 Related Work

Several works address automated verification of model-to-model transformations for the same Hoare-style notion of partial correctness. Nevertheless, as it happens with any other modelization process, there are several possible translations from a given source language, e.g. from ATL, both to different formalisms and following different strategies depending on the correctness properties that we want to verify and the desired properties of the verification procedure itself (complete, automatic, etc.). Next, we will distinguish two groups of related works: 1) automatic unbounded verification approaches; 2) automatic bounded verification approaches.

In the first group, [16] type checks transformations with respect to a schema (i.e., a metamodel) by using the MONA solver. Only the typing of the graph can be checked in this approach. Other properties, e.g., the uniqueness of names, cannot be expressed in this approach while they can be checked using OCL. In [2], they propose novel deduction rules to be used for automatically deriving properties of model transformations. On the contrary, we do not propose new deduction rules but rely on the deduction systems implemented in the SMT solvers. In [24], unbounded model checking is used to check first-order linear temporal properties for graph transformation systems. Standard OCL does not capture temporal properties (nor does our mapping). In the same vein, [20] map transformations into the DSLTrans language, and pattern-based properties into a model-checking problem (using Prolog). To our knowledge, there is no approach in this group dealing with ATL or with a constraint language similar to OCL.

In the second group, [26], provides a rewriting logic semantics for ATL, and uses Maude to simulate and verify transformations. In the same logic, [7] formalizes QVT-like transformations. [1,4] translate pattern-based model-transformations into the relational logic of Alloy. In [6] they extend a verification technique capable of checking statically that graph based refactoring rule applications preserve consistency with regards to graph constraints by automatically performing counterexample finding. The consistency notion used in [6] is analogous to the partial correctness notion that we use in this paper when applied to 'in-place' transformations. The same notion is used also in [23] for the verification of graph programs. Finally, we have also translated ATL transformations into corresponding transformation models to capture its execution semantics by OCL constraints, and we have used Alloy to find counterexamples [9]. Notice that the generated transformation models, which have a nice intuitive interpretation as a trace model, could be further translated to FOL using [11]. However, the translation obtained is not adequate for SMT solvers since the resulting FOL assertions are overly complex for efficient e-pattern matching (neither Yices nor Z3 could perform any of the proofs in our examples using the resulting specification of this approach). In this sense, our works complement each other, i.e., [9] is well-suited for bounded counter example finding, whereas the approach presented in this paper can provide proofs for the unsatisfiable cases. In [10], we showed how TGG and QVT-R transformations can be translated into OCL-based transformation models, yielding a different kind of models than [9] due to the different execution semantics. We expect that a direct first-order semantics for QVT-R is also required in order to employ SMT solvers for verifying these transformations following the approach presented in this paper.

7 Conclusions and Future Work

We summarize our contributions in this paper as follows: (i) We provide a novel (and the only, so far) first-order semantics for a declarative subset of ATL; (ii) we propose an automatic, unbounded approach to formally verify a Hoare style notion of partial correctness for ATL transformations with respect to OCL pre- and postconditions; (iii) we have successfully used SMT solvers to perform that verification, i.e., to automatically prove constraints that will always hold on target models. Our approach is suited for 'black box' application by non-formal developers, because we do not require interaction with a theorem prover. For our approach to be practical, we are implementing a tool that automatically maps ATL matched rules to first order logic files and employs the Z3 solver to check whether the implications between pre and postconditions hold. A tool that automatically maps to FOL the OCL constrained metamodels is already implemented.

Our work complements those on bounded verification of model transformations (e.g., using SAT-based tools such as Alloy). Whereas bounded approaches are generally more efficient (and complete within the bounds) in finding counterexamples, our approach provides evidence in the cases when no counterexample could be found by the bounded approach. Furthermore, SMT solvers provide the information about which are the assertions producing unsatisfiability, i.e., 'what implies what', since they can extract the unsatisfiable core. This is particularly useful in terms of guaranteeing which preconditions imply which postconditions. For both the example presented in this paper and the larger one provided on-line, all assertions of interest could be proven in less than a few seconds. We expect our FOL mapping for ATL matched rules to fall in the scope of what can be handled by the model-based quantifier instantiation (MBQI) decision procedure of Z3 [15] that is refutationally complete for quantified formulas of uninterpreted and 'almost uninterpreted' functions (presuming that the OCL constraints of the metamodels fall into the same fragment). Yet, even if the generated first-order specification falls into a fragment for which the SMT decision procedure is refutationally complete, termination is not guaranteed for the case when a counterexample exists. Finally, we must say that most of the work in all examples that we considered so far, could be done by the incomplete (but more efficient) standard e-pattern matching procedure. In future work we will also generalize our translation to consider broader rule patterns, in particular, to deal with more than one output pattern elements and to cover OCL collection expressions that can be used on the right-hand side of binding statements. Lazy rules (in a restricted way) will be also considered.

Acknowledgements. This research was partially funded by the EU project NESSoS (FP7 256890) and, by the Nouvelles Eq́uipes Program of the Pays de la Loire Region (France).

Moreover, we would like to thank Salvador Martinez Pérez (AtlanMod Research group-INRIA) for his helpful comments on previous versions of the work presented here.

References

1. K. Anastasakis, B. Bordbar, and J. M. Küster. Analysis of Model Transformations via Alloy. In *MoDeVVa'2007, Proceedings*, 2007. http://www.modeva.org/2007/modevva07.pdf.
2. M. Asztalos, L. Lengyel, and T. Levendovszky. Towards automated, formal verification of model transformations. In *3rd International Conference on Software Testing, Verification and Validation, ICST'2010, Proceedings*, pages 15–24. IEEE Computer Society, 2010.
3. ATL User Guide, 2012. Website, http://wiki.eclipse.org/ATL/User_Guide_-_The_ATL_Language.
4. L. Baresi and P. Spoletini. On the use of Alloy to analyze graph transformation systems. In A. Corradini, H. Ehrig, U. Montanari, L. Ribeiro, and G. Rozenberg, editors, *Third International Conference on Graph Transformations, ICGT 2006. Proceedings.*, volume 4178 of *LNCS*, pages 306–320. Springer, 2006.
5. B. Baudry, S. Ghosh, F. Fleurey, R. B. France, Y. L. Traon, and J.-M. Mottu. Barriers to systematic model transformation testing. *Communications of ACM*, 53(6), 2010.
6. B. Becker, L. Lambers, J. Dyck, S. Birth, and H. Giese. Iterative development of consistency-preserving rule-based refactorings. In *Theory and Practice of Model Transformations - 4th International Conference, ICMT 2011, Zurich, Switzerland, June 27-28, 2011. Proceedings*, volume 6707 of *LNCS*, pages 123–137. Springer, 2011.
7. A. Boronat, R. Heckel, and J.Meseguer. Rewriting logic semantics and verification of model transformations. In *FASE'09, Proceedings*, volume 5503 of *LNCS*. Springer, 2009.
8. F. Büttner, M. Egea, and J. Cabot. On verifying ATL transformations using 'off-the-shelf' SMT solvers: Examples. Website, http://www.emn.fr/z-info/atlanmod/index.php/MODELS_2012_SMT, 2012.
9. F. Büttner, M. Egea, J. Cabot, and M. Gogolla. Verification of ATL transformations using transformation models and model finders. In *14th International Conference on Formal Engineering Methods, ICFEM 2012, Kyoto, Japan, November 12-16, 2012. Proceedings*, Lecture Notes in Computer Science. Springer, 2012. In press.
10. J. Cabot, R. Clariso, E. Guerra, and J. Lara. Verification and validation of declarative model-to-model transformations through invariants. *Journal of Systems and Software*, 83(2), 2010.
11. M. Clavel, M. Egea, and M. A. G. de Dios. Checking unsatisfiability for OCL constraints. *Electronic Communications of the EASST*, 24, 2009.
12. L. M. de Moura and N. Bjørner. Satisfiability modulo theories: Introduction and applications. *Communications of ACM*, 54(9):69–77, 2011.
13. B. Dutertre and L. D. Moura. The Yices SMT solver. Technical report, Computer Science Laboratory, SRI International, 2006. http://yices.csl.sri.com/tool-paper.pdf.
14. H. Ehrig, K. Ehrig, U. Prange, and G. Taentzer. *Fundamentals of Algebraic Graph Transformation (Monographs in Theoretical Computer Science. An EATCS Series)*. Springer, 2006.
15. Y. Ge and L. M. de Moura. Complete instantiation for quantified formulas in satisfiabiliby modulo theories. In A. Bouajjani and O. Maler, editors, *Computer Aided Verification, CAV 2009, Proceedings*, volume 5643 of *LNCS*, pages 306–320, 2009.
16. K. Inaba, S. Hidaka, Z. Hu, H. Kato, and K. Nakano. Graph-transformation verification using monadic second-order logic. In P. Schneider-Kamp and M. Hanus, editors, *ACM SIGPLAN Conference on Principles and Practice of Declarative Programming – PPDP 2011, Proceedings*, pages 17–28. ACM, 2011.
17. F. Jouault, F. Allilaire, J. Bézivin, and I. Kurtev. ATL: A model transformation tool. *Science of Computer Programming*, 72(1-2), 2008.
18. F. Jouault and I. Kurtev. Transforming Models with ATL. In *Proceedings of the Model Transformations in Practice Workshop at MoDELS 2005*, 2005. online, http://sosym.dcs.kcl.ac.uk/events/mtip05/submissions/jouault_kurtev__transforming_models_with_atl.pdf.

19. K. Lano and S. K. Rahimi. Model-driven development of model transformations. In *Theory and Practice of Model Transformations - 4th International Conference, ICMT 2011, Zurich, Switzerland, June 27-28, 2011. Proceedings*, volume 6707 of *LNCS*, pages 47–61. Springer, 2011.
20. L. Lucio, B. Barroca, and V. Amaral. A Technique for Automatic Validation of Model Transformations. In D. C. Petriu, N. Rouquette, and O. Haugen, editors, *MODELS 2010, Part I*, volume 6394 of *LNCS*. Springer, 2010.
21. OMG. *The Object Constraint Language Specification v. 2.2 (Document formal/2010-02-01)*. Object Management Group, Inc., Internet: http://www.omg.org/spec/OCL/2.2/, 2010.
22. OMG. *Meta Object Facility (MOF) Core Specification 2.4.1 (Document formal/2011-08-07)*. Object Management Group, Inc., Internet: http://www.omg.org, 2011.
23. C. M. Poskitt and D. Plump. Hoare-style verification of graph programs. *Fundamenta Informaticae*, 118(1-2):135–175, 2012.
24. A. Rensink. Explicit state model checking for graph grammars. In P. Degano, R. D. Nicola, and J. Meseguer, editors, *Concurrency, Graphs and Models, Essays Dedicated to Ugo Montanari on the Occasion of His 65th Birthday*, volume 5065 of *LNCS*, pages 114–132. Springer, 2008.
25. M. Richters and M. Gogolla. On formalizing the UML object constraint language OCL. In *Conceptual Modeling - ER '98, 17th International Conference on Conceptual Modeling, Singapore, November 16-19, 1998, Proceedings*, volume 1507 of *LNCS*, pages 449–464. Springer, 1998.
26. J. Troya and A. Vallecillo. A Rewriting Logic Semantics for ATL. *Journal of Object Technology*, 10, 2011.
27. Yices. Website, http://yices.csl.sri.com/.
28. Z3. Website, http://research.microsoft.com/en-us/um/redmond/projects/z3/.