A Deductive Approach for Fault Localization in ATL Model Transformations

Zheng Cheng, **Massimo Tisi**
AtlanMod team (Inria, IMT Atlantique, LS2N)
Nantes, France

20th International Conference on Fundamental Approaches to Software Engineering
April 28, 2017 - Uppsala, Sweden

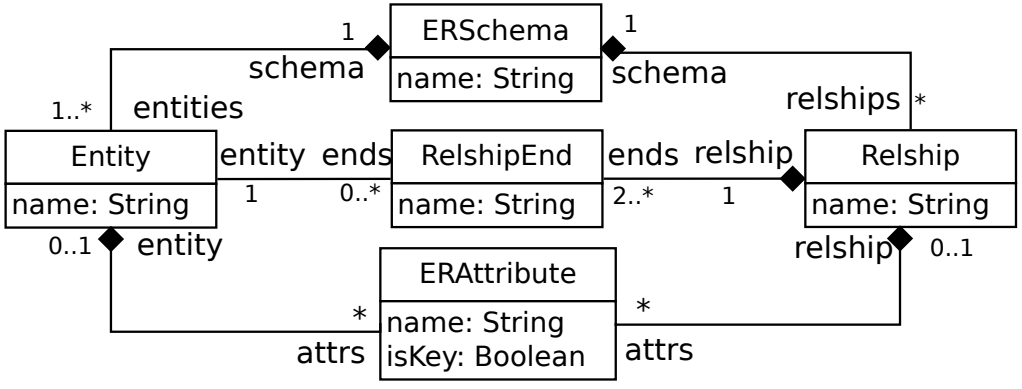
Contract-based development of MT

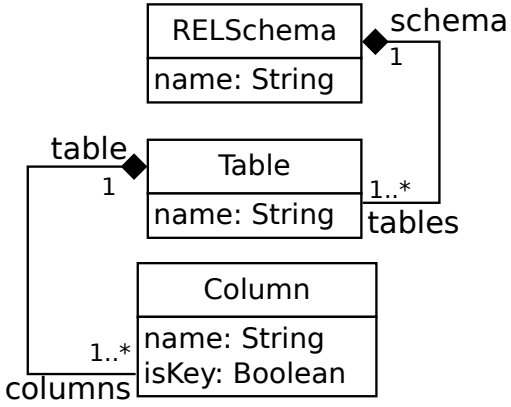
- Model transformation (MT) at the center of Model-Driven Engineering (MDE)
  - Dedicated MT languages, e.g. ATL
  - Source $\leftrightarrow$ target pattern mappings
- In critical systems, contract-based development of MTs
  - E.g., in aviation [Berry, 2008], automotive [Selim et al., 2012], medical [Wagelaar, 2014]
  - Pre/postconditions for the MT to be considered as correct
  - Often expressed in OCL

Example

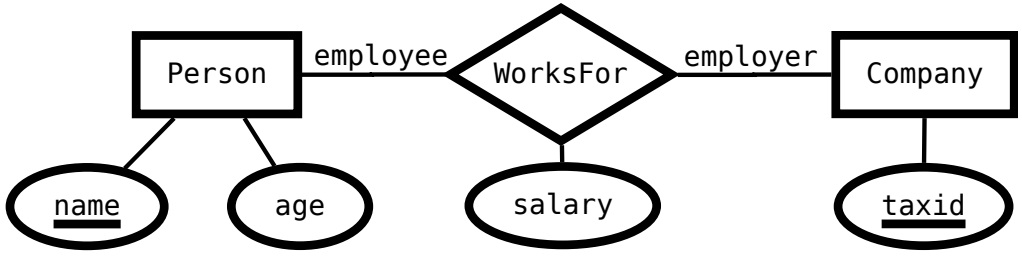
Entity-Relationship to Relational Schema (from [Büttner et al., 2012b])

Metamodels:





Sample models:



Person(<u>name</u>, age)

Company(<u>taxid</u>)

WorksFor(<u>name</u>, <u>taxid</u>, salary)

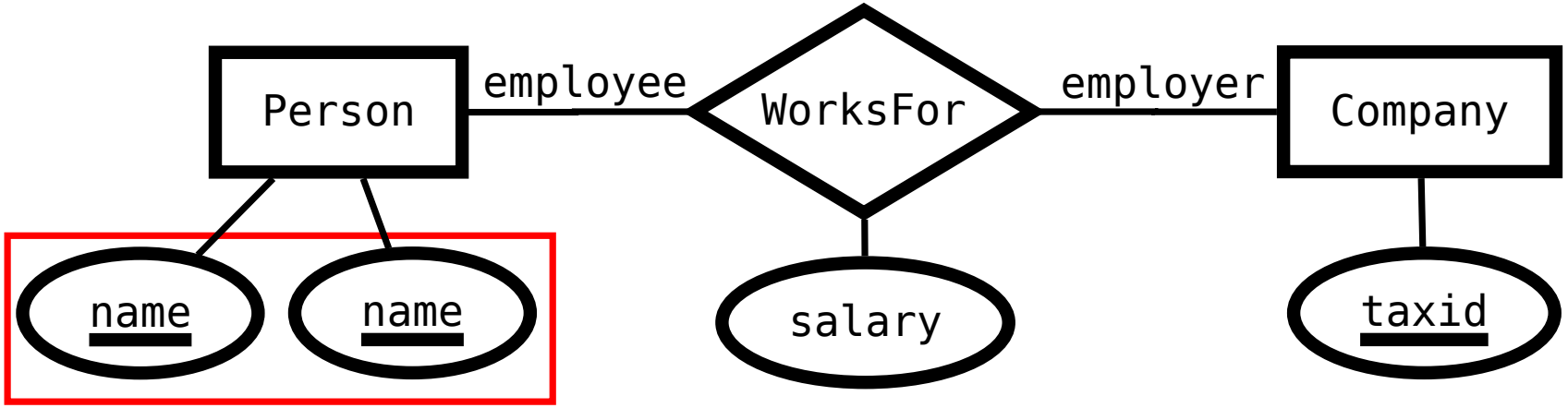
(in the paper we use an excerpt of HSM2FSM from [Büttner et al., 2012a])

Transformation Precondition 1

Name unicity on entity attributes

```
-- attribute names within the same entity are disjoint
context ER!EREntity def: Pre1():
  self.attrs->forAll(a1, a2 | a1<>a2 implies a1.name<>a2.name);
```

Sample violation:

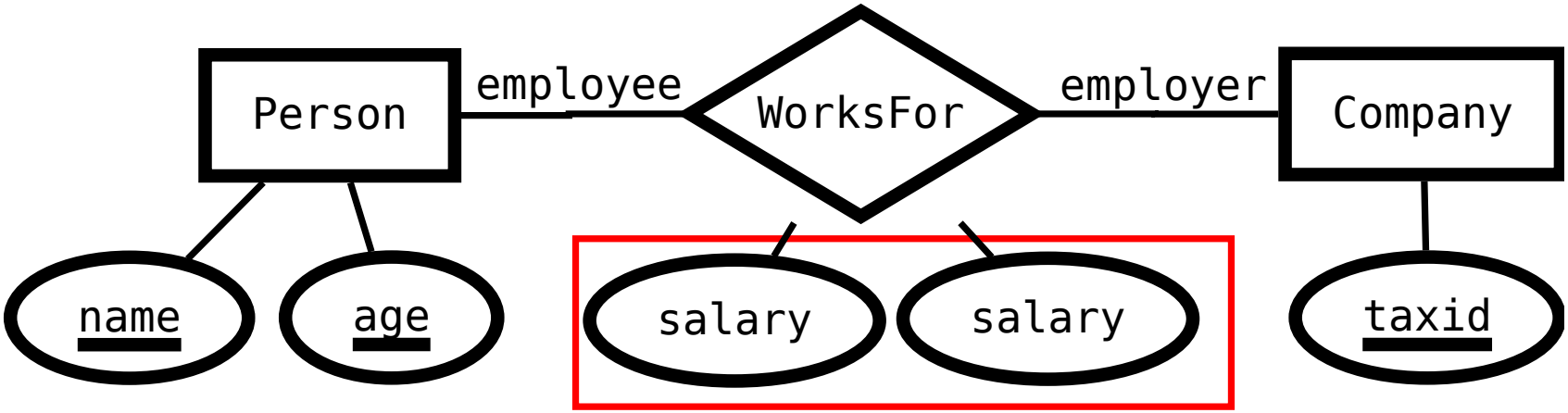


Transformation Precondition 2

Name unicity on relationship attributes

```
-- attribute names within the same relship are disjoint
context ER!ERRelship def: Pre2():
 self.attrs->forAll(a1, a2 | a1<>a2 implies a1.name<>a2.name);
```

Sample violation:

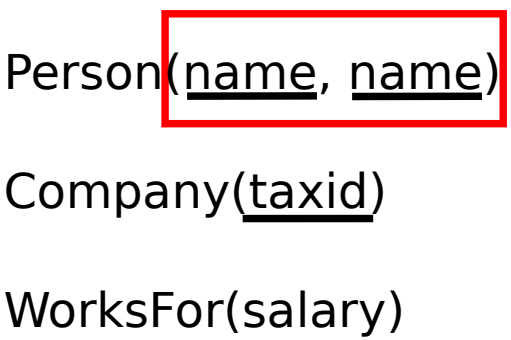


Transformation Postcondition

Name unicity on column names

```
-- column names within the same table are disjoint
context REL!Table def: Post1():
 self.columns->forAll(c1, c2 | c1<>c2 implies c1.name<>c2.name);
```

Sample violation:



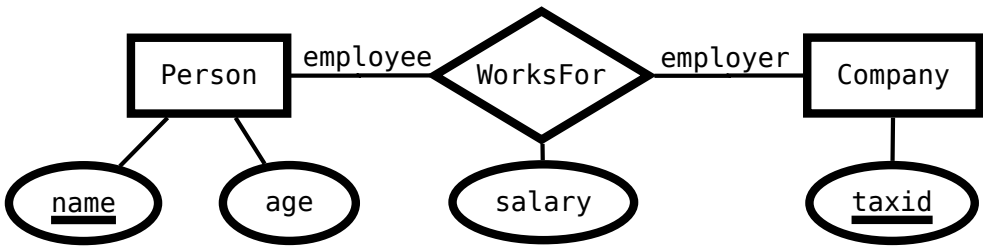
ATL Transformation

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

rule S2S {
from s: ER!ERSchema
to t: REL!RELSchema (name<—s.name, tables <— s.relships, tables <— s.entities)}

rule E2T {
from s: ER!Entity
to t: REL!Table (name<—s.name) }

rule R2T {
from s: ER!Relship
to t: REL!Table (name<—s.name) }

rule EA2C {
from att : ER!ERAttribute, ent : ER!Entity (att.entity = ent)
to t : REL!Column (name <— att.name, isKey <— att.isKey, table <— ent) }

rule RA2C {
from att : ER!ERAttribute, rs : ER!Relship ( att.relship = rs )
to t : REL!Column(name <— att.name, isKey <— att.isKey, table <— rs ) }

rule RA2CK {
from att : ER!ERAttribute, rse : ER!RelshipEnd ( att.entity = rse.entity and att.isKey = true )
to t : REL!Column ( name <— att.name, isKey <— att.isKey, table <— rse.relship )}
```

ATL Transformation

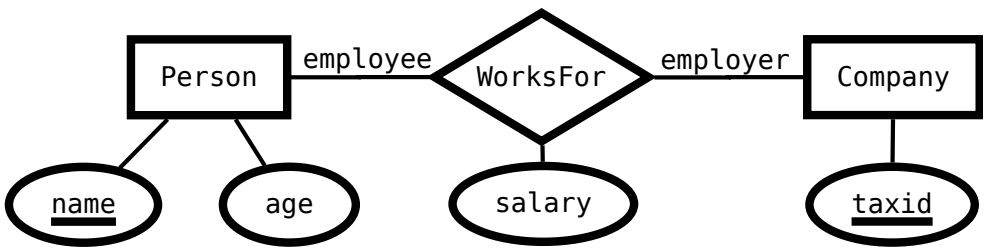
module ER2REL; **create** OUT : REL **from** IN : ER;

rule S2S {
from s: ER!ERSchema
to t: REL!RELSchema (name<—s.name, tables <— s.relships, tables <— s.entities)}



ATL Transformation

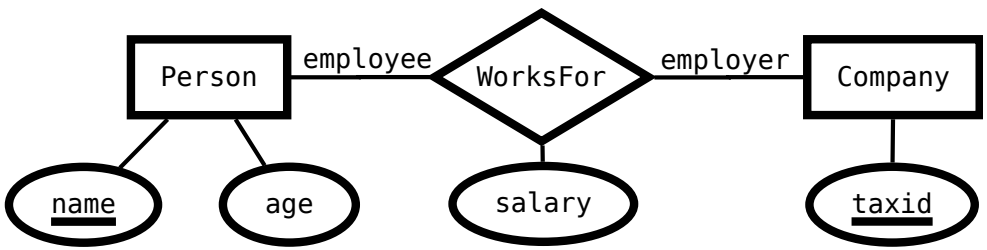
```
rule E2T {
from s: ER!Entity
to t: REL!Table (name<-s.name) }

rule R2T {
from s: ER!Relship
to t: REL!Table (name<-s.name) }
```



Person ()
Company ()
WorksFor ()

ATL Transformation

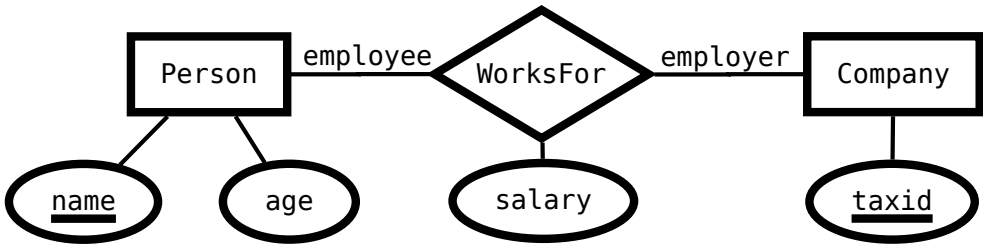
```
rule EA2C {
from att : ER!ERAttribute, ent : ER!Entity (att.entity = ent)
to t : REL!Column (name <- att.name, isKey <- att.isKey, table <- ent) }

rule RA2C {
from att : ER!ERAttribute, rs : ER!Relship ( att.relship = rs )
to t : REL!Column(name <- att.name, isKey <- att.isKey, table <- rs ) }
```



Person (name, age)
Company (taxid)
WorksFor (salary)

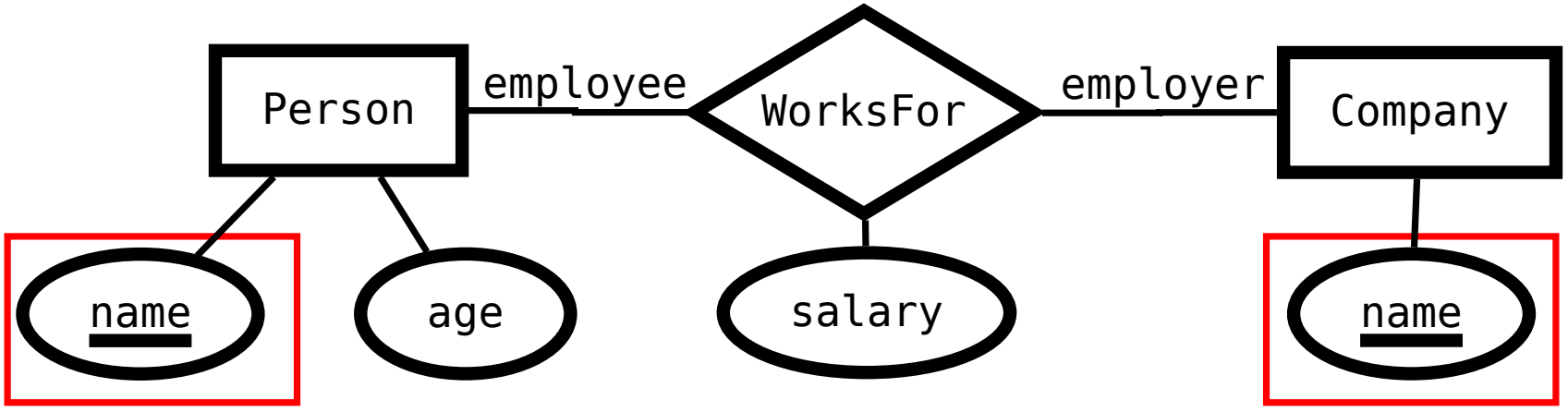
ATL Transformation

```
rule RA2CK {
from att : ER!ERAttribute, rse : ER!RelshipEnd
   ( att.entity = rse.entity and att.isKey = true )
to t : REL!Column ( name <– att.name, isKey <– att.isKey, table <– rse.relship )}
```



Person (<u>name</u>, age)
Company (<u>taxid</u>)
WorksFor (<u>name</u>, <u>taxid</u>, salary)

A counterexample



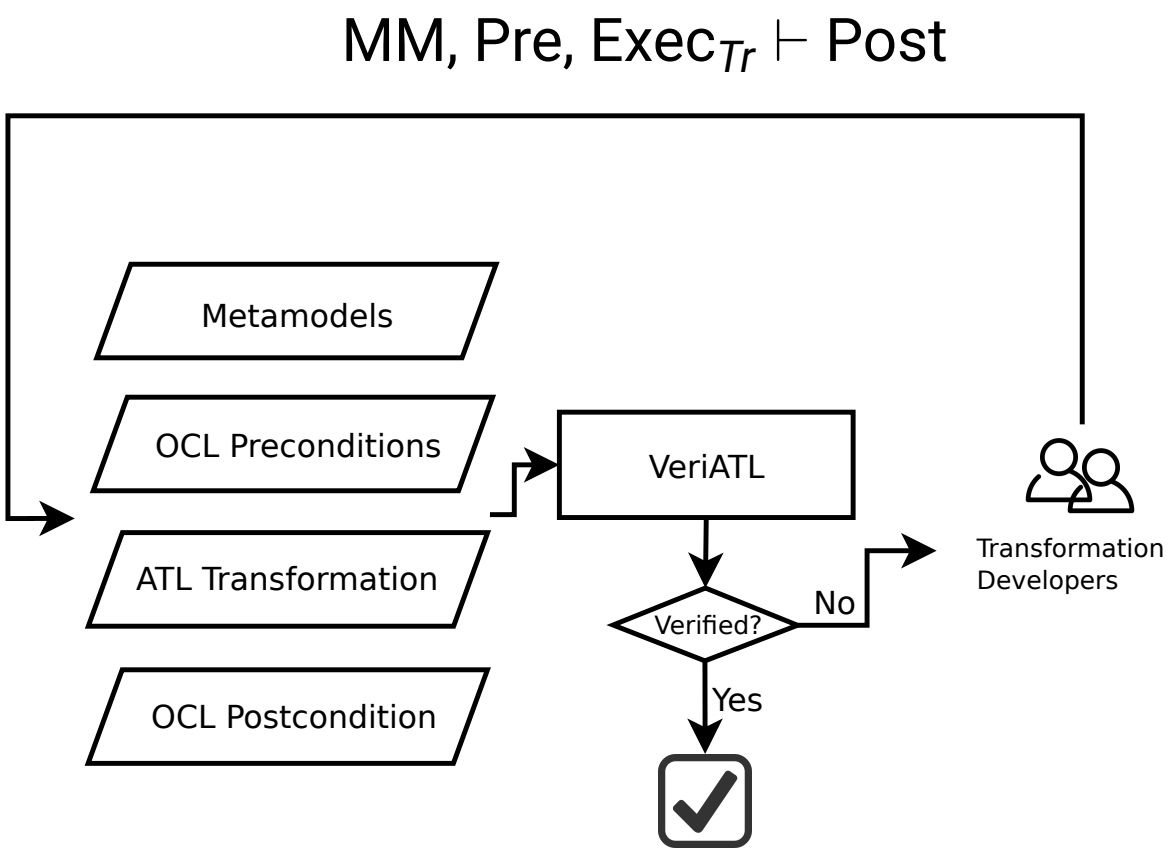
Person(<u>name</u>, age)

Company(<u>name</u>)

WorksFor(<u>name</u>, <u>name</u>, salary)

Overview of VeriATL [Cheng et al., 2015]

- Generates in Boogie:
  - axiomatic semantics of the ATL transformation
  - encoding of EMF metamodels and OCL contracts
- Hoare-style verification (by SMT on Z3)



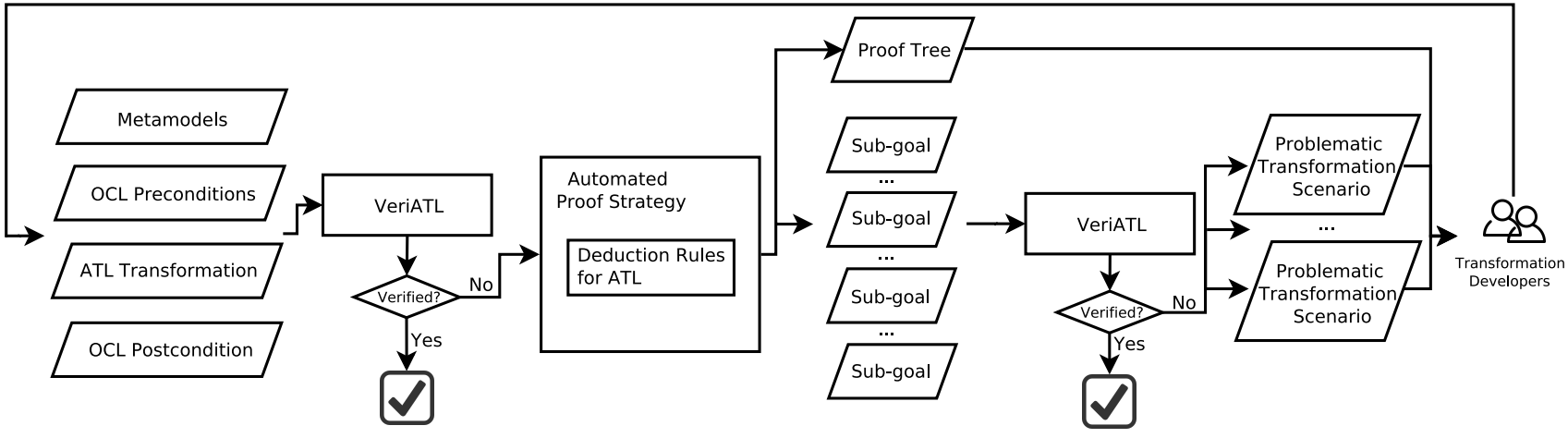
Deductive fault localization for ATL

- When a contract is not verified, no useful feedback from the deductive verification system
  - Time-consuming manual examination of the full MT and its contracts, e.g., reasoning on implicit rule interactions
- In related works, model-finding tools to generate counterexamples, e.g. [Büttner et al., 2012b]
  - How significant is the counterexample?
  - Do new counterexamples really add information?
- In [Burgueño et al., 2015] fault localization by exploiting ATL static trace information
  - i.e. target types $\leftrightarrow$ rules that potentially generate these types (+ corresponding source types)
  - by a syntactical and conservative approach
- We exploit ATL static trace information for fault localization by a semantic and deductive approach

Augmenting VeriATL with fault localization

- Fine-grained verification for fault localization by natural deduction and program slicing:

$$\text{MM, Pre, Exec}_{Tr} \vdash \text{Post}$$



$$\text{MM, Pre, Exec}_{sliced}\text{, Hypotheses} \vdash \text{Conclusion}$$

Postcondition decomposition (1)

proof tree:

Post1

$self_i$

I1

example:

```
-- columns names within table are disjoint
context REL!Table def: Post1():
*G*
  self.columns->forAll(c1, c2 |
    c1<>c2 implies c1.name<>c2.name);


context REL!Table def: I1():
*H* var self0: REL!Table
*H* REL!Table->allInstances()->includes(self0)
*G*
  self0.columns->forAll(c1, c2 |
    c1<>c2 implies c1.name<>c2.name)
```

applied rule:

Context T ...
G F[self]

$self_i$

H var self0: T
H T->allInstances()->includes(self0)
.
.
.
G F[self0/self]

Deduction rules

- 16 natural deduction rules for propositional and predicate logic
- 4 additional rules:

$$\frac{x.a : T \qquad T \in cl(MM_{\mathrm{T}})}{x.a \in All(T) \vee unDef(x.a)}\ \mathrm{TP}_{e1}$$

$$\frac{x.a : Seq\ T \qquad T \in cl(MM_{\mathrm{T}})}{(|x.a| > 0 \wedge \forall i \cdot (i \in x.a \Rightarrow i \in All(T) \vee unDef(i))) \vee |x.a| = 0}\ \mathrm{TP}_{e2}$$

$$\frac{T \in cl(MM_{\mathrm{T}}) \qquad trace(T) = \{R_1, ..., R_n\} \qquad i \in All(T)}{genBy(i, R_1) \vee ... \vee genBy(i, R_n)}\ \mathrm{TR}_{e1}$$

$$\frac{T \in cl(MM_{\mathrm{T}}) \qquad trace(T) = \{R_1, ..., R_n\} \qquad i : T \qquad unDef(i)}{\neg(genBy(i, R_1) \vee ... \vee genBy(i, R_n))}\ \mathrm{TR}_{e2}$$

Soundness proved based on the operational semantics of ATL

Postcondition decomposition (2)

example:

```
context REL!Table def: I1():
*H* var self0: REL!Table
*H* REL!Table->allInstances()->includes(self0)
*G*
  self0.columns->forAll(c1, c2 |
    c1<>c2 implies c1.name<>c2.name)
```

```
context REL!Table def: I1_1():
*H* var self0: REL!Table
*H* REL!Table->allInstances()->includes(self0)
*H* genBy(self0,E2T) or genBy(self0,R2T)
*G*
  self0.columns->forAll(c1, c2 |
    c1<>c2 implies c1.name<>c2.name)
```

applied rule:

H T in MMt
H trace(T) = {R1...Rn}
H T.allInstances()->includes(x0)

TR_{e1} ▌

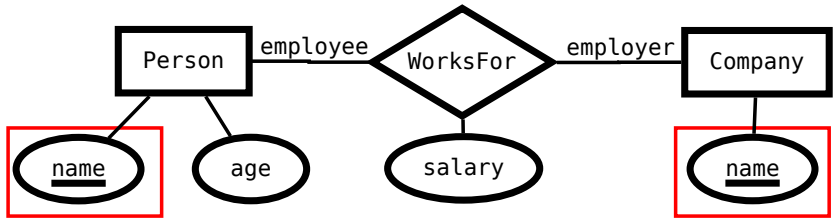
H T.allInstances()->includes(x0)
H genBy(x0,R1) or ... or genBy(x0, Rn)

proof tree:

Post1
$self_i$ ▌
I1
TR_{e1} ▌
I1_1

Automated proof strategy

- Simplify the original postcondition as much as possible
- As a by-product, deduce new hypotheses as debugging clues



context REL!Table **inv** S28:
H **var** self0: REL!Table, c1,c2: REL!Column
H genBy(self0,R2T)
H c1 <> c2
H self0.columns—>**includes**(c1)
H self0.columns—>**includes**(c2)
H genBy(c1,RA2CK)
H genBy(c2,RA2CK)
...
G c1.name <> c2.name

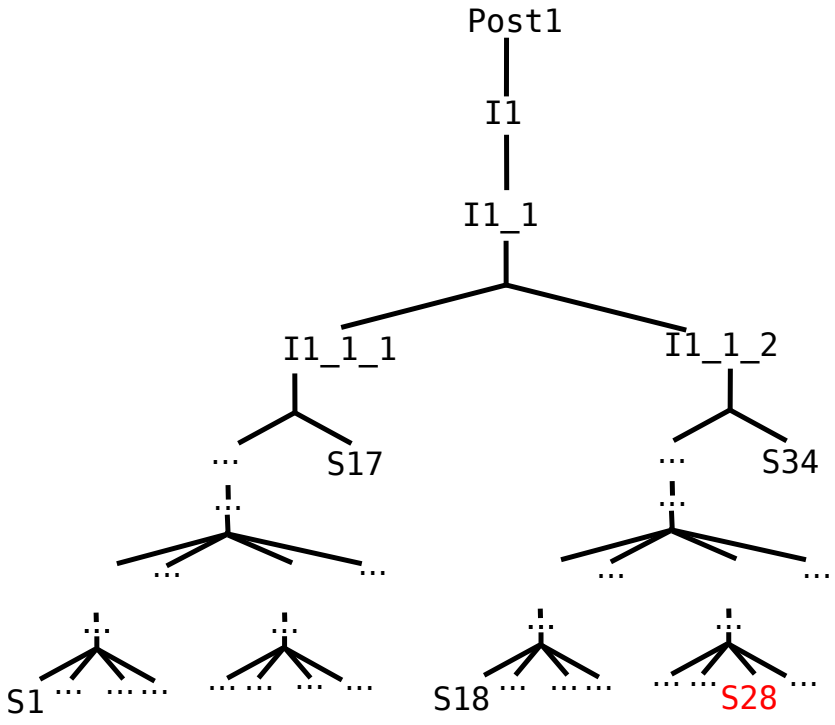
Person(name, age)

Company(name)

WorksFor(name, name, salary)

Sub-goal pruning

- We prune away Hoare triples that are automatically proved, because they represent:
  - impossible cases (e.g. incompatible rules $\rightarrow$ contradictory premises)
  - or cases that are not affected by the bug
- W.r.t. related work this:
  - improves the precision of fault localization
  - makes the whole process practical (few cases to manually analyze)



Transformation slicing

Only rules referred by genBy predicates can impact the verification result of each sub-goal.

```
context REL!Table inv S28:
*H* var self0: REL!Table, c1,c2: REL!Column
*H* c1 <> c2
*H* genBy(self0,R2T)
*H* self0.columns—>includes(c1)
*H* self0.columns—>includes(c2)
*H* genBy(c1,RA2CK)
*H* genBy(c2,RA2CK)
 ...
*G* c1.name <> c2.name
```

```
context ER!ERSchema inv Pre1, Pre2

rule R2T {
from s: ER!Relship
to t: REL!Table (name<—s.name) }

rule RA2CK {
from att: ER!ERAttribute, rse: ER!RelshipEnd
       ( att.entity = rse.entity
         and att.isKey = true )
to t : REL!Column
       ( name <— att.name,
         isKey <— att.isKey,
        table <— rse.relship )}
```

Case study: Correctness

Mutations on HSM2FSM from [Büttner et al., 2012a]

1. The guilty constructs are included in the slice
   - and no inconclusive verification results

	Unverified Post.	$L_{faulty} \in PTS$
MT2	#5	T
DB1	#5	-
MB6	#4	T
AF2	#4	T
MF6	#2	N/A
	#4	T
DR1	#1	-
	#2	-
AR	#1	T
	#3	T

Case study: Practicality

❷ Sub-goals that need to be examined are usually few and meaningful

	Unverified Post.	Unverified / Total Sub-goals
MT2	#5	3 / 4
DB1	#5	1 / 1
MB6	#4	1 / 12
AF2	#4	2 / 12
MF6	#2	0 / 6
	#4	1 / 12
DR1	#1	3 / 6
	#2	3 / 6
AR	#1	1 / 8
	#3	6 / 16

Case study: Performance

❸ Verifying sub-goals takes less time than verifying postconditions (less rule interactions to axioms)

	Unverified Post.		Sub-goals	
	ID	Verif. Time (s)	Unverified / Total	Max Time (s)
MT2	#5	3.1	3 / 4	1.6
DB1	#5	2.9	1 / 1	1.5
MB6	#4	3.2	1 / 12	2.6
AF2	#4	3.4	2 / 12	2.6
MF6	#2	3.8	0 / 6	2.1
	#4	3.8	1 / 12	2.5
DR1	#1	2.1	3 / 6	1.6
	#2	2.2	3 / 6	1.8
AR	#1	3.9	1 / 8	1.8
	#3	4.1	6 / 16	1.9

Limitations

- Language coverage
  - ATL declarative matched rules, non-recursive helpers, first-order OCL contracts
- Completeness
  - No slicing in case of partial derivations
  - $\rightarrow$ we report to the user
- Scalability
  - We allow the user to specify a bound for the maximum number of unverified cases to generate
  - Future work: verification of intermediate nodes in the proof tree

Conclusion & Future Work

In summary:

- MT languages like ATL allow us to easily infer static trace information
- a deductive approach can use this information for improving the precision and automation of fault localization

In current/future work, we exploit contract decomposition for:

- fine-grained incremental verification of MT [Cheng and Tisi, 2017]
- increasing scalability for large MT verification tasks

References

Berry, G. (2008).

Synchronous design and verification of critical embedded systems using SCADE and Esterel.
In *12th International Workshop on Formal Methods for Industrial Critical Systems*, pages 2–2. Springer, Berlin, Germany.

Burgueño, L., Troya, J., Wimmer, M., and Vallecillo, A. (2015).

Static fault localization in model transformations.
IEEE Transactions on Software Engineering, 41(5):490–506.

Büttner, F., Egea, M., and Cabot, J. (2012a).

On verifying ATL transformations using 'off-the-shelf' SMT solvers.
In *15th International Conference on Model Driven Engineering Languages and Systems*, pages 198–213, Innsbruck, Austria. Springer.

Büttner, F., Egea, M., Cabot, J., and Gogolla, M. (2012b).

Verification of ATL transformations using transformation models and model finders.
In *14th International Conference on Formal Engineering Methods*, pages 198–213, Kyoto, Japan. Springer.

Cheng, Z., Monahan, R., and Power, J. F. (2015).

A sound execution semantics for ATL via translation validation.
In *8th International Conference on Model Transformation*, pages 133–148, L'Aquila, Italy. Springer.

Cheng, Z. and Tisi, M. (2017).

Incremental Deductive Verification for Relational Model Transformations.
In *ICST 2017 - 10th IEEE International Conference on Software Testing, Verification and Validation*, Tokyo, Japan.

References II

Selim, G., Wang, S., Cordy, J., and Dingel, J. (2012).

Model transformations for migrating legacy models: An industrial case study.
In *8th European Conference on Modelling Foundations and Applications*, pages 90–101, Lyngby, Denmark. Springer.

Wagelaar, D. (2014).

Using ATL/EMFTVM for import/export of medical data.
In *2nd Software Development Automation Conference*, Amsterdam, Netherlands.

Thank you

: github.com/veriatl/

AtlanMod Inria IMT Atlantique Bretagne-Pays de la Loire École Mines-Télécom LS2N

Zheng Cheng
zheng.cheng@inria.fr

Massimo Tisi
massimo.tisi@inria.fr