Testing Model Transformations in Model Driven Engineering

Benoit Baudry

INRIA – IRISA

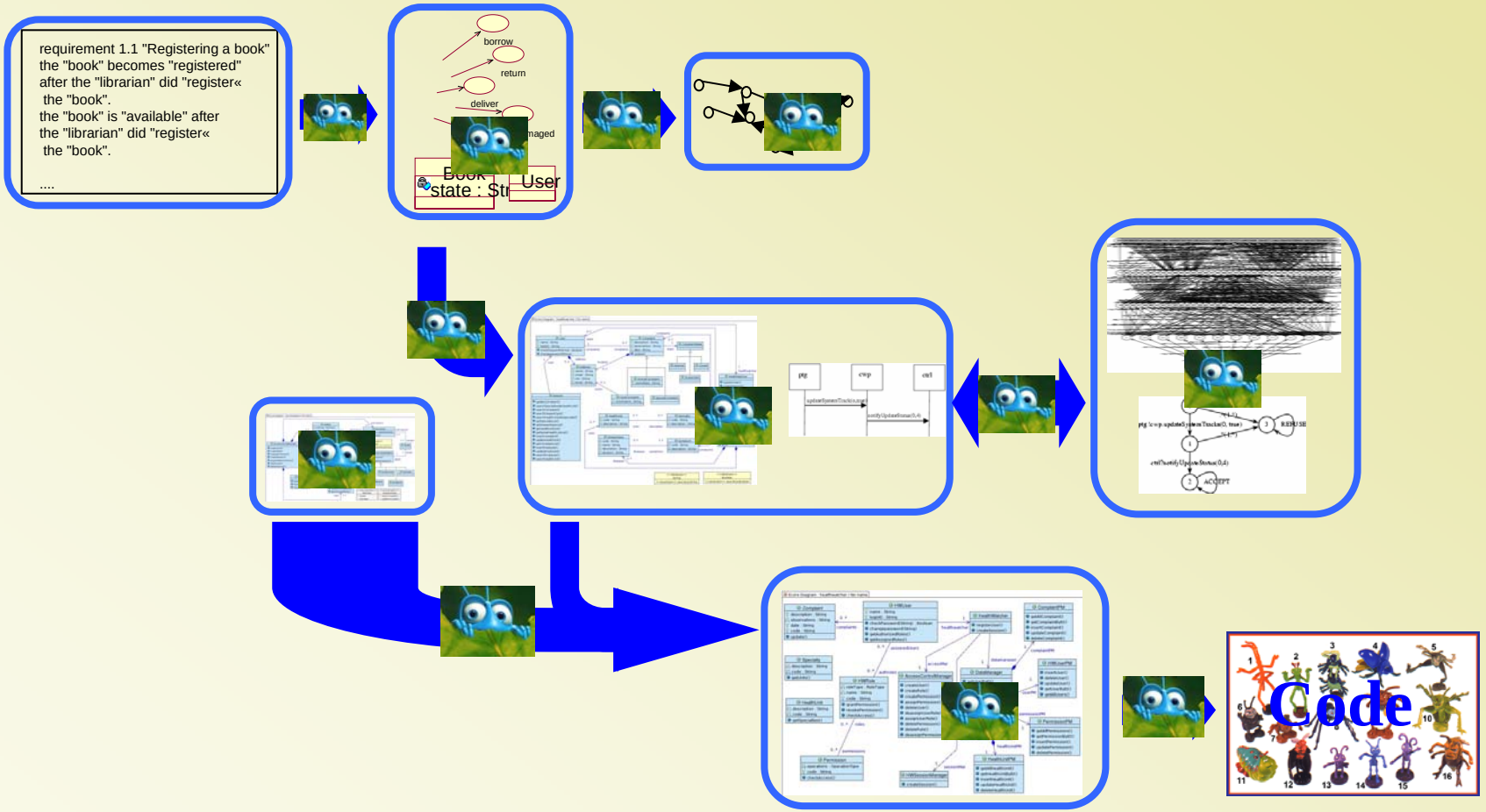
(currently visiting CSU)

Outline

- What about model transformation testing?
- Triskell's contributions
  - Coverage criteria
  - Model synthesis
- Related work
- Challenges

INRIA

Model Transformation Testing: Motivation



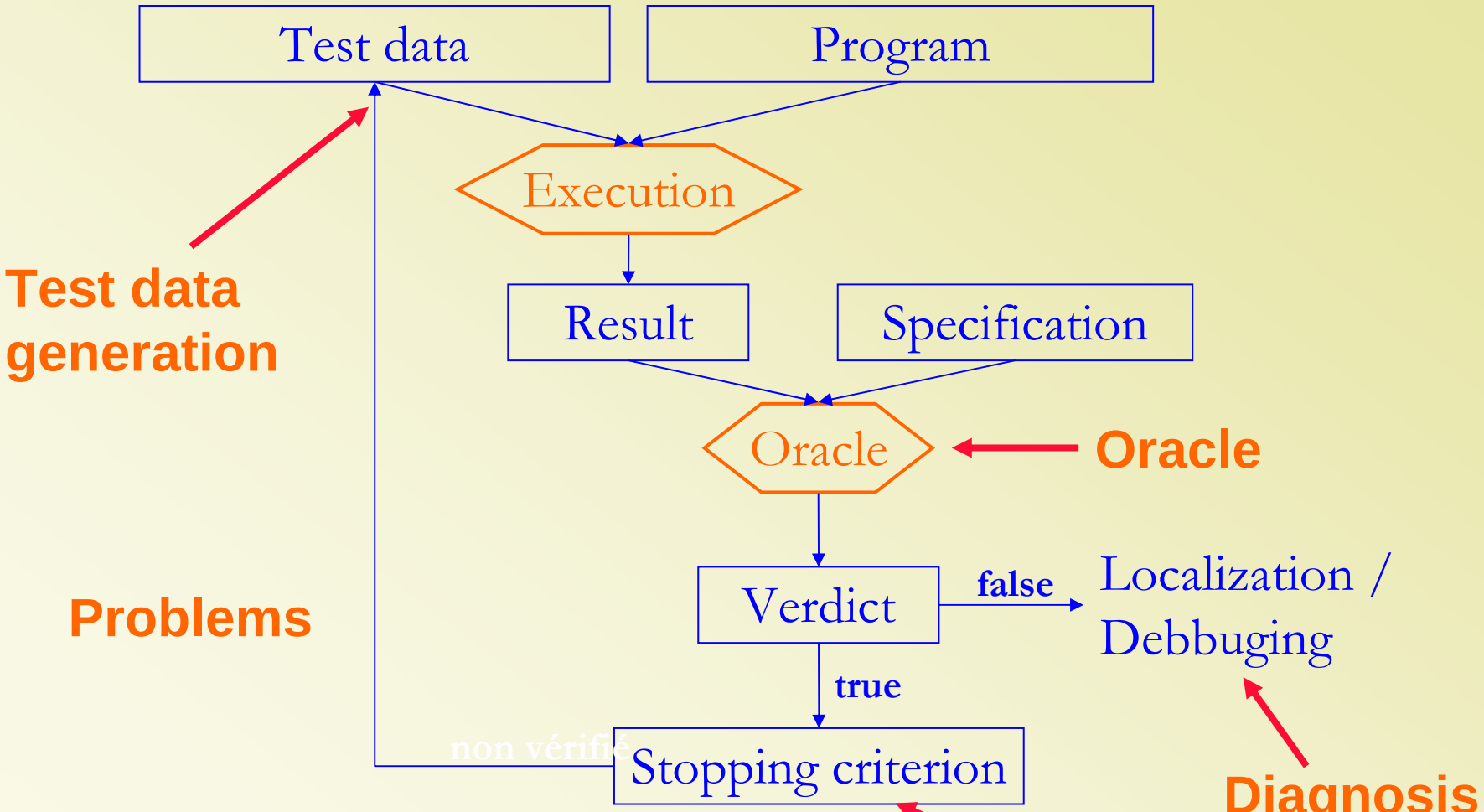
INRIA

Model Transformation Testing: Motivation

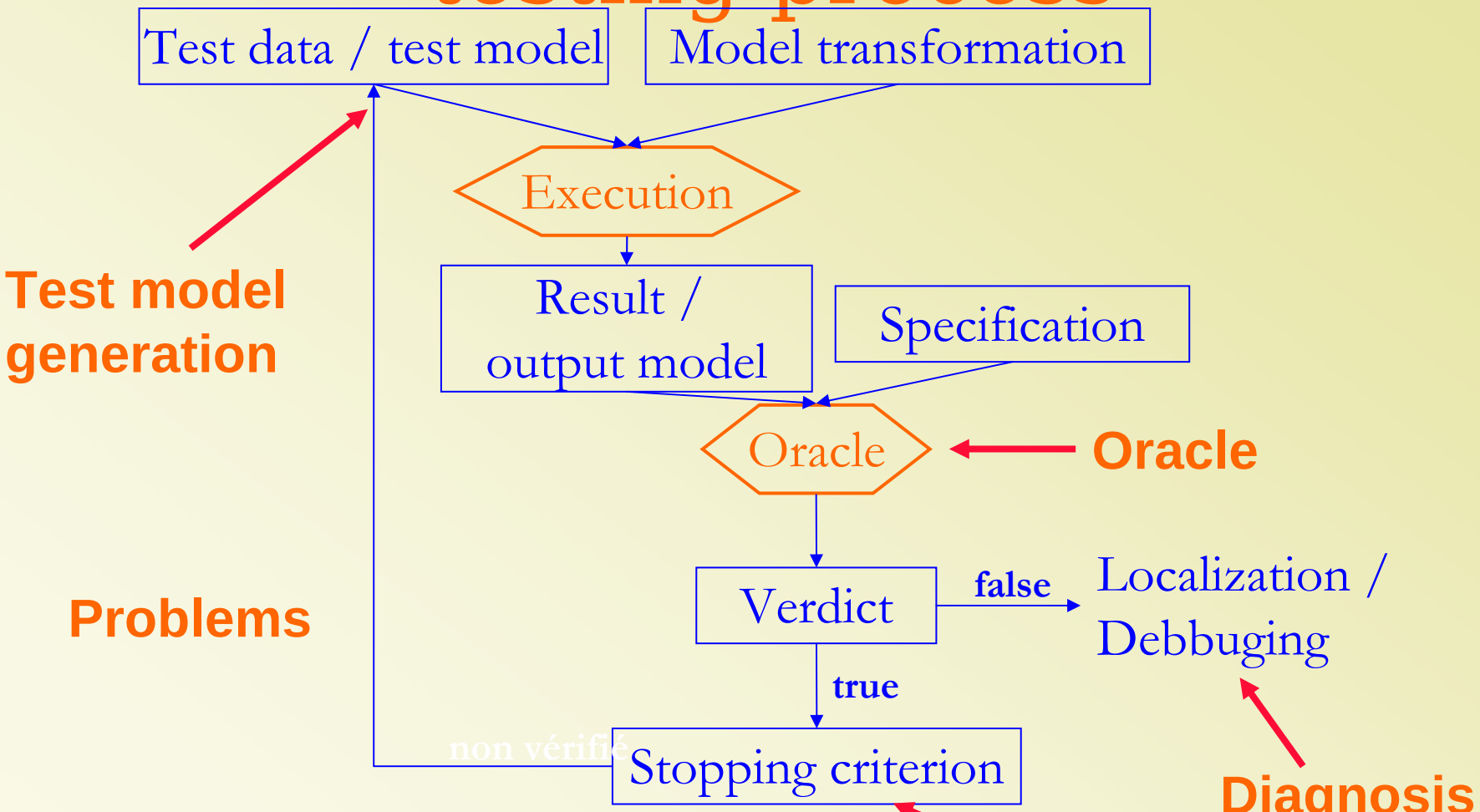
- A transformation is meant to be reused
  - But also has to be adapted from one project to another
- A transformation is meant to hide the complexity
  - we woud like to trust the transformation as we trust a compiler

INRIA

Dynamic testing process



Dynamic transformtation testing process



Dynamic transformtation testing process

- Specific issues
- Complex data
  - Models are manipulated as sets of objects
- Complex constraints
- Lack of specific tools

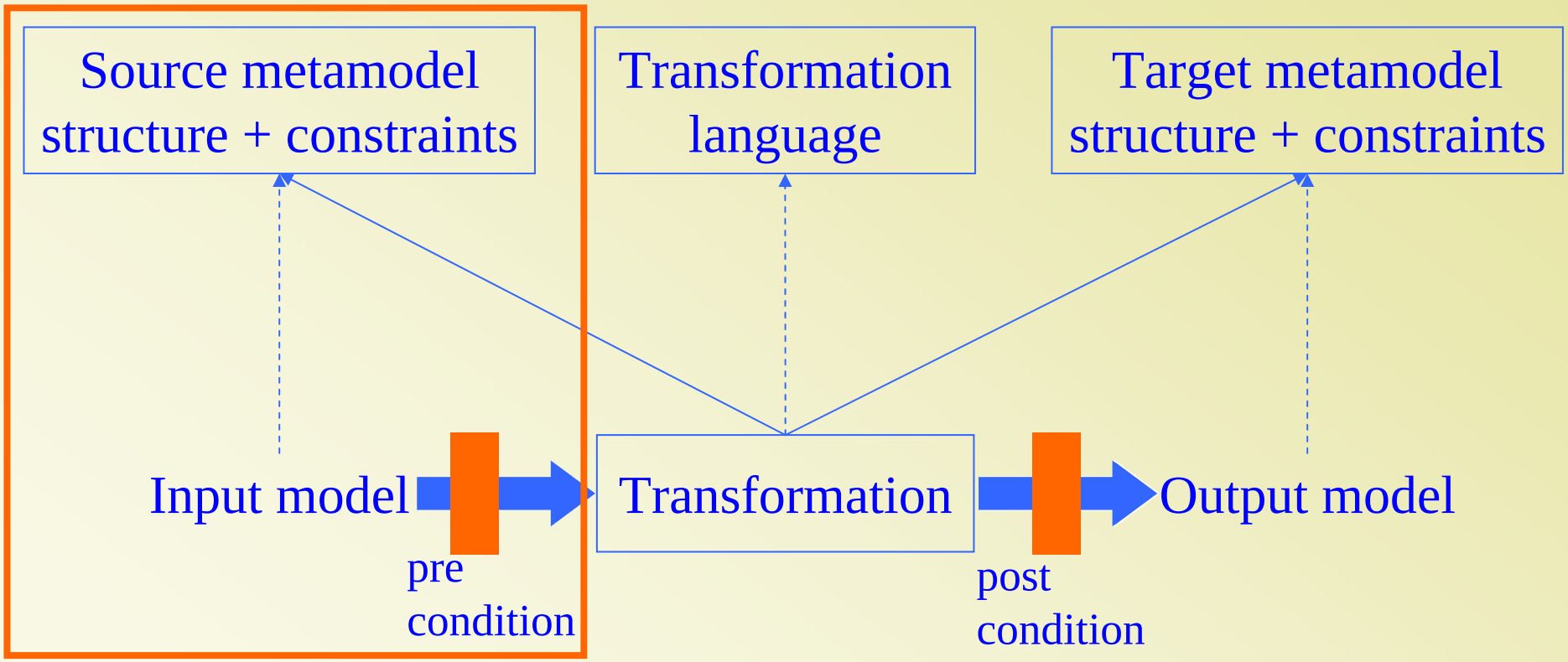
INRIA

Model Transformation Testing

- Currently in Triskell
  - Coverage criteria
  - Automatic synthesis of test models (in coll. With Mc Gill)
  - Specific fault models

INRIA

Model transformation



INRIA

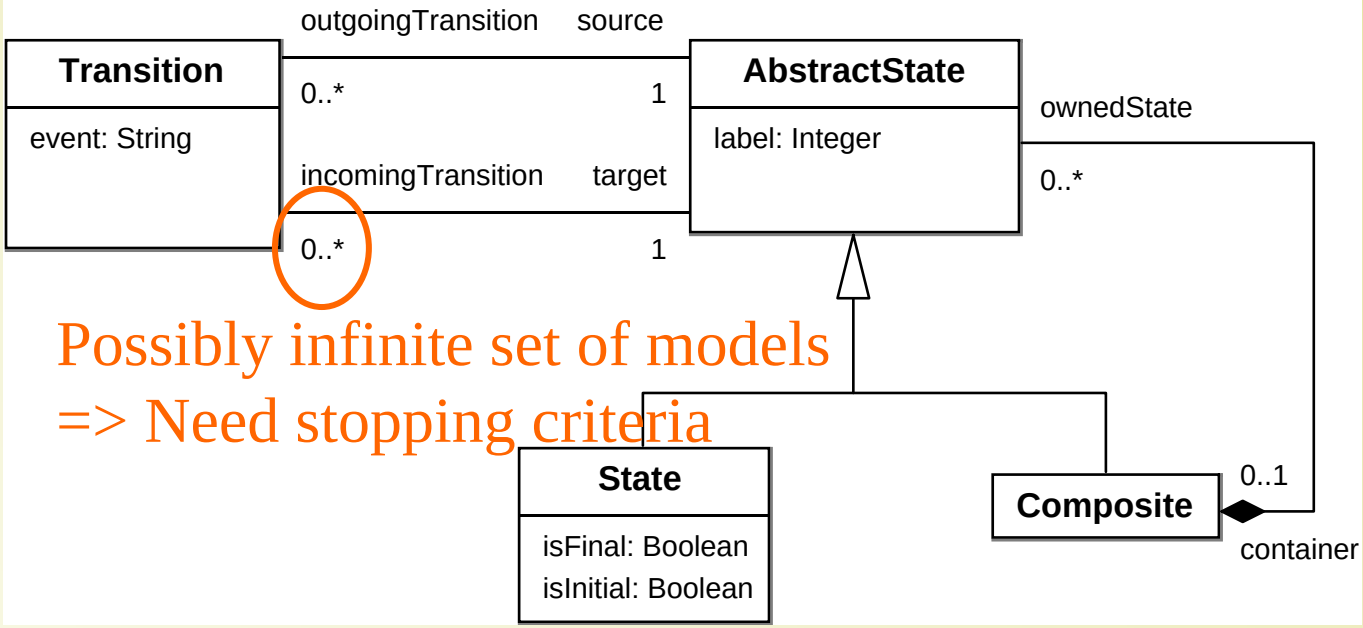
Test data generation: criteria

- Several model transformation languages
  - Different features
  - Different paradigms
  - Different domains
- We did not want to choose
- We define black-box criteria
  - Independent of the model transformation language

Test data generation: criteria

- Define test criteria based on the input metamodel
  - Intuition: a set of models is adequate for testing if every class of the input metamodel is instantiated at least once and if the properties have relevant values
- A model for testing is called a test model

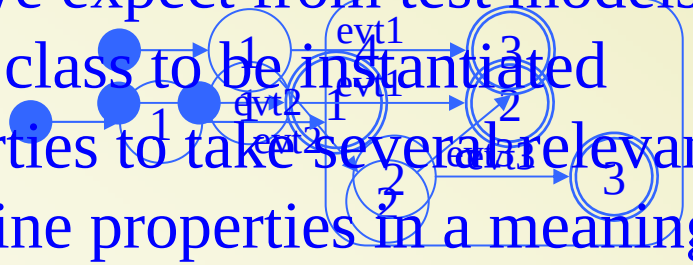
Test data generation: Example



Possibly infinite set of models
=> Need stopping criteria

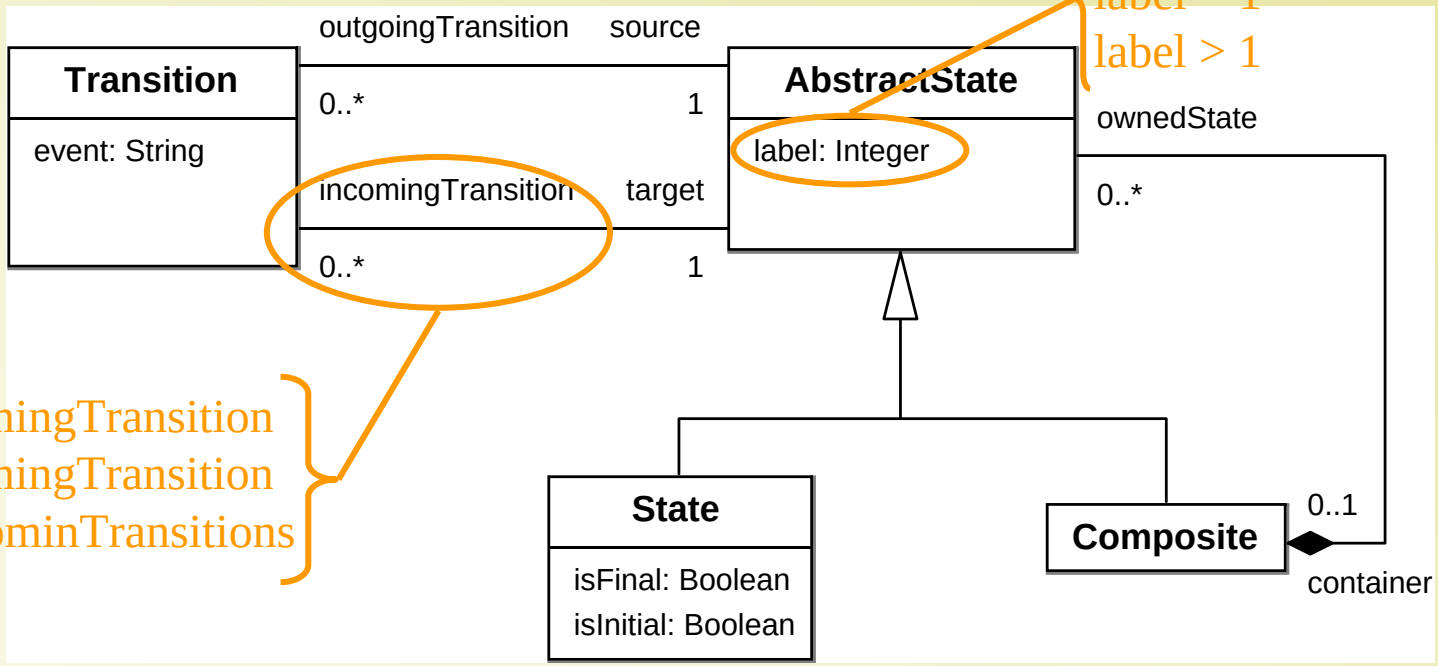
What we expect from test models

- Every class to be instantiated
- Properties to take several relevant values
- Combine properties in a meaningful way



INRIA

Relevant values for properties



Adapt category partition testing to define ranges of relevant values for properties of the metamodel

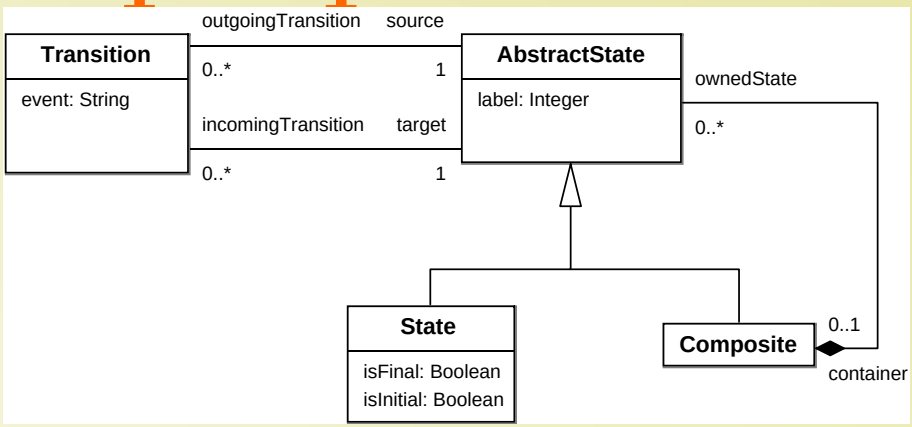
INRIA

Relevant values for properties

- Define partitions for each property in the input metamodel
- A partition defines a set of ranges on a domain
  - choose one value in each range for the property
- Example
  - partition for AbstractState::label={[0],[1],[2..MaxInt]}
  - A set of test models will need to have, at least three states with three different values for label

INRIA

Relevant values for properties



Transition::**event**	**{''}, {'evt1'}, {'.+'}**
Transition::**#source**	**{1}**
Transition::#**target**	**{1}**
AbstractState::**label**	**{0}, {1}, {2..MaxInt}**
AbstractState::#**container**	**{0}, {1}**
AbstractState::#**incomingTransition**	**{0}, {1}, {2..MaxInt}**
AbstractState::#**outgoingTransition**	**{0}, {1}, {2..MaxInt}**
State:**isInitial**	**{true}, {false}**
State::**isFinal**	**{true}, {false}**
Composite::**#ownedState**	**{0}, {1}, {2..MaxInt}**

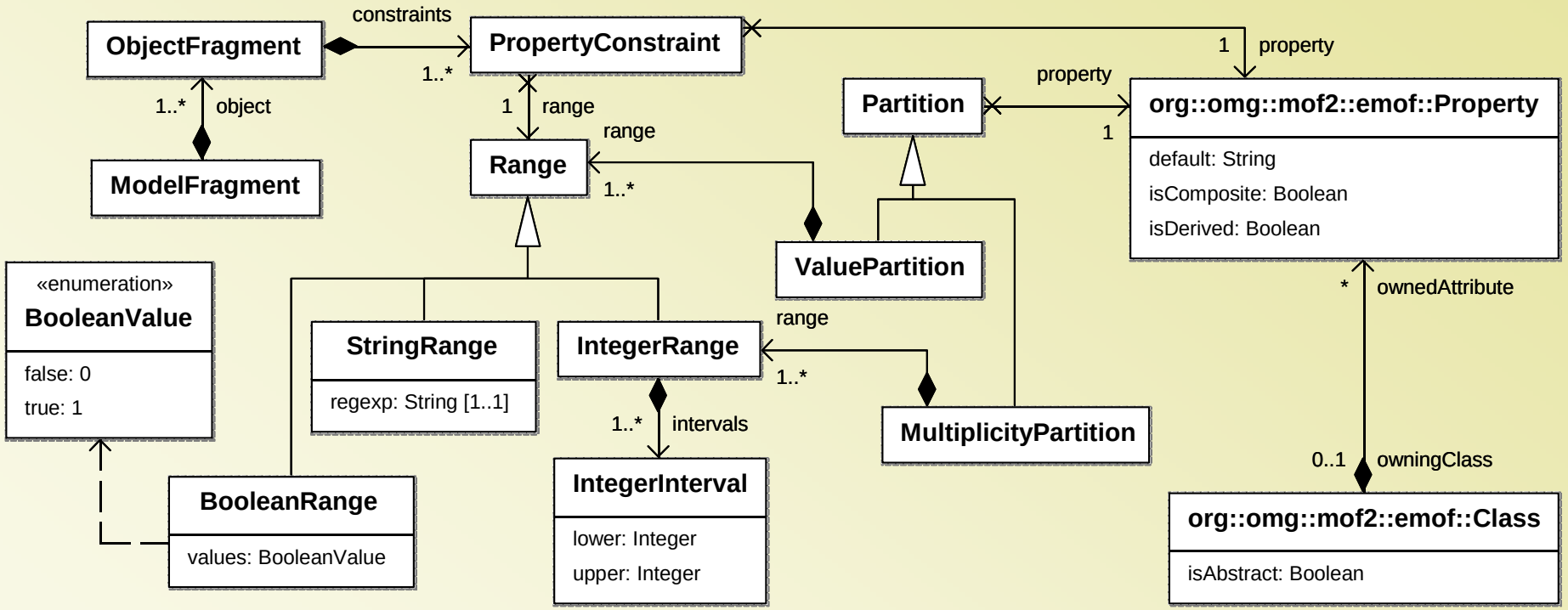
INRIA

Relevant object structures

Transition::**event**	**{''}, {'evt1'}, {'.+'}**
Transition::#**source**	**{1}**
Transition::#**target**	**{1}**
AbstractState::**label**	**{0}, {1}, {2..MaxInt}**
AbstractState::#**container**	**{0}, {1}**
AbstractState::#**incomingTransition**	**{0}, {1}, {2..MaxInt}**
AbstractState::#**outgoingTransition**	**{0}, {1}, {2..MaxInt}**
State:**isInitial**	**{true}, {false}**
State::**isFinal**	**{true}, {false}**
Composite::#**ownedState**	**{0}, {1}, {2..MaxInt}**

We would like to constrain the models to have a State with one outgoing transition **and** more than one incoming transitions

Relevant object structures



Relevant object structures

- Criteria define structures that must be covered by test models
- These criteria combine partitions
- One criterion = set of constraints
  - one criterion declares the set of ranges that should be covered by a set of test models
- Example
  - Range coverage: Each range of each partition for all properties of the meta-model must be used in at least one model.

Test criteria

- Six test criteria (different combinations of ranges)
  - AllRanges
  - AllPartitions
  - + 4 class criteria
    - object fragments constrain each property of the object
- Do not consider constraints on the metamodel
  - Might generate insatisfiable fragments

INRIA

Evaluating a set of models

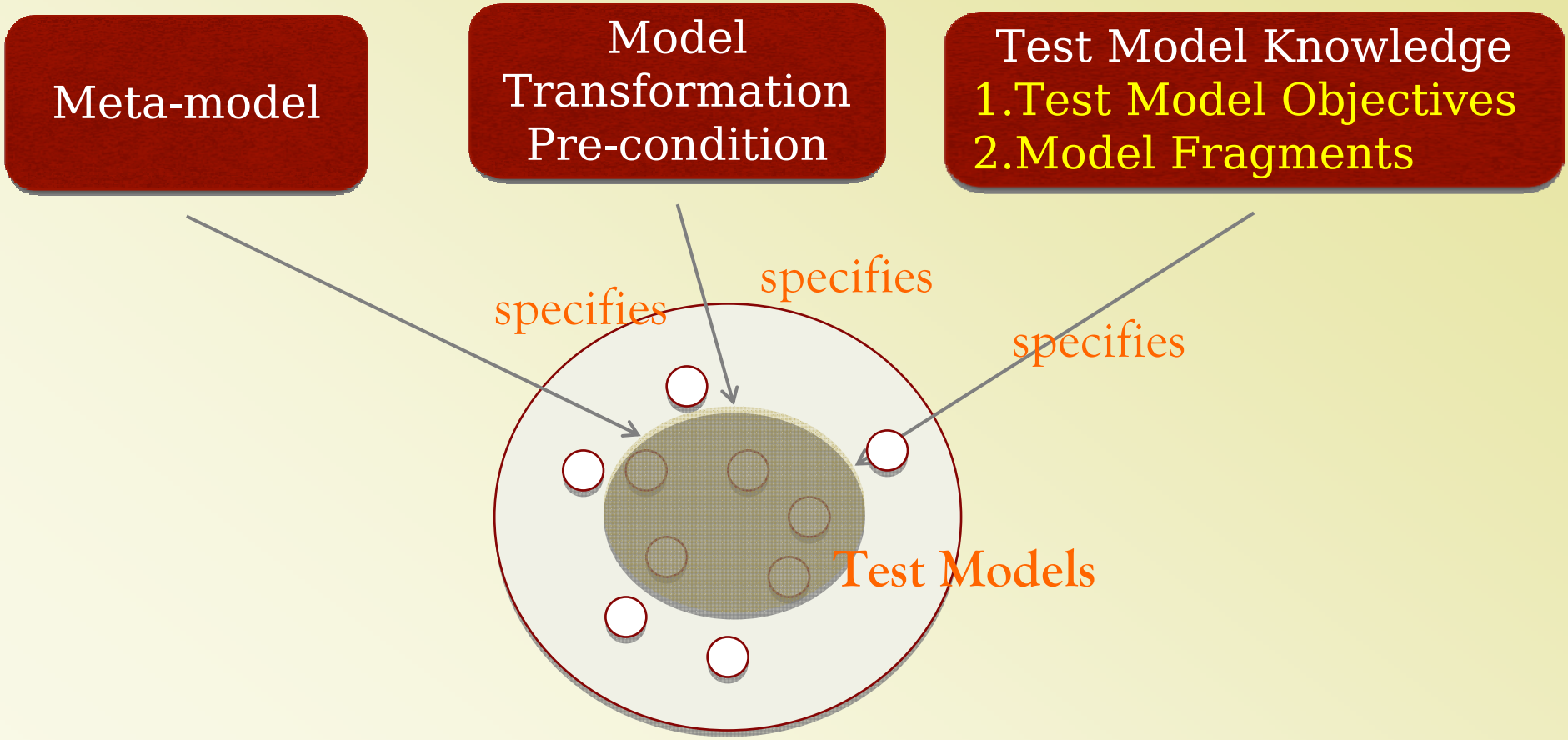
- A prototype tool: MMCC
  - Framework for partitions and fragments definitions
- Computes a set of model fragments according to
  - Input metamodel
  - Test criterion
- Checks the coverage of a set of test models
  - With respect to the set of model fragments

Automatic synthesis of test models

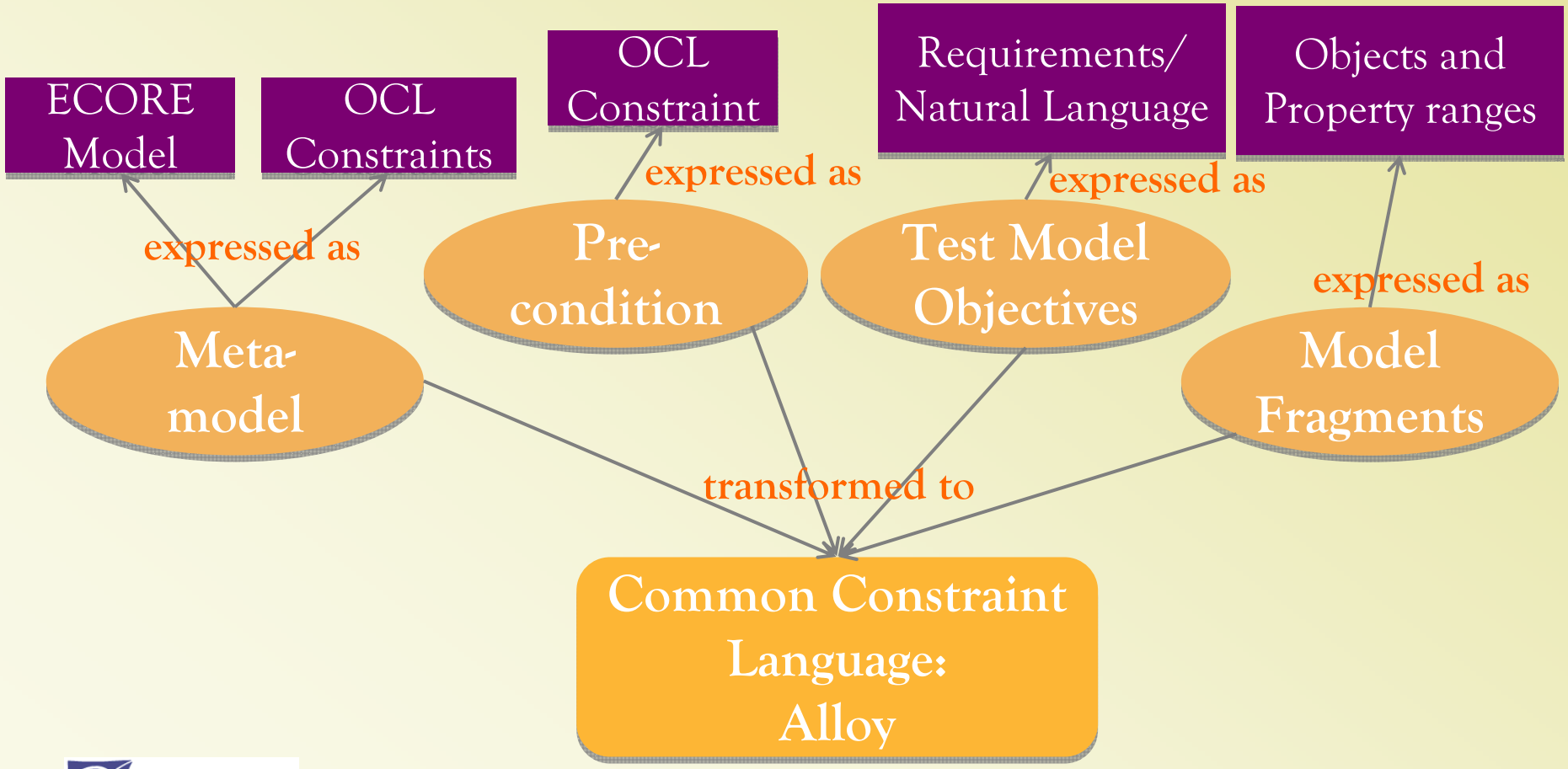
- Automatic synthesis useful to
  - Limit the effort for test generation
  - Evaluate the test criteria
- Challenges:
  - Combine different sources of knowldege
  - Expressed in different formalisms
  - Complex constraints

INRIA

Automatic synthesis of test models



The Solution(1):
Combining Knowledge to Common Constraint Language



INRIA

Model synthesis

The *run* command:

Integer scope

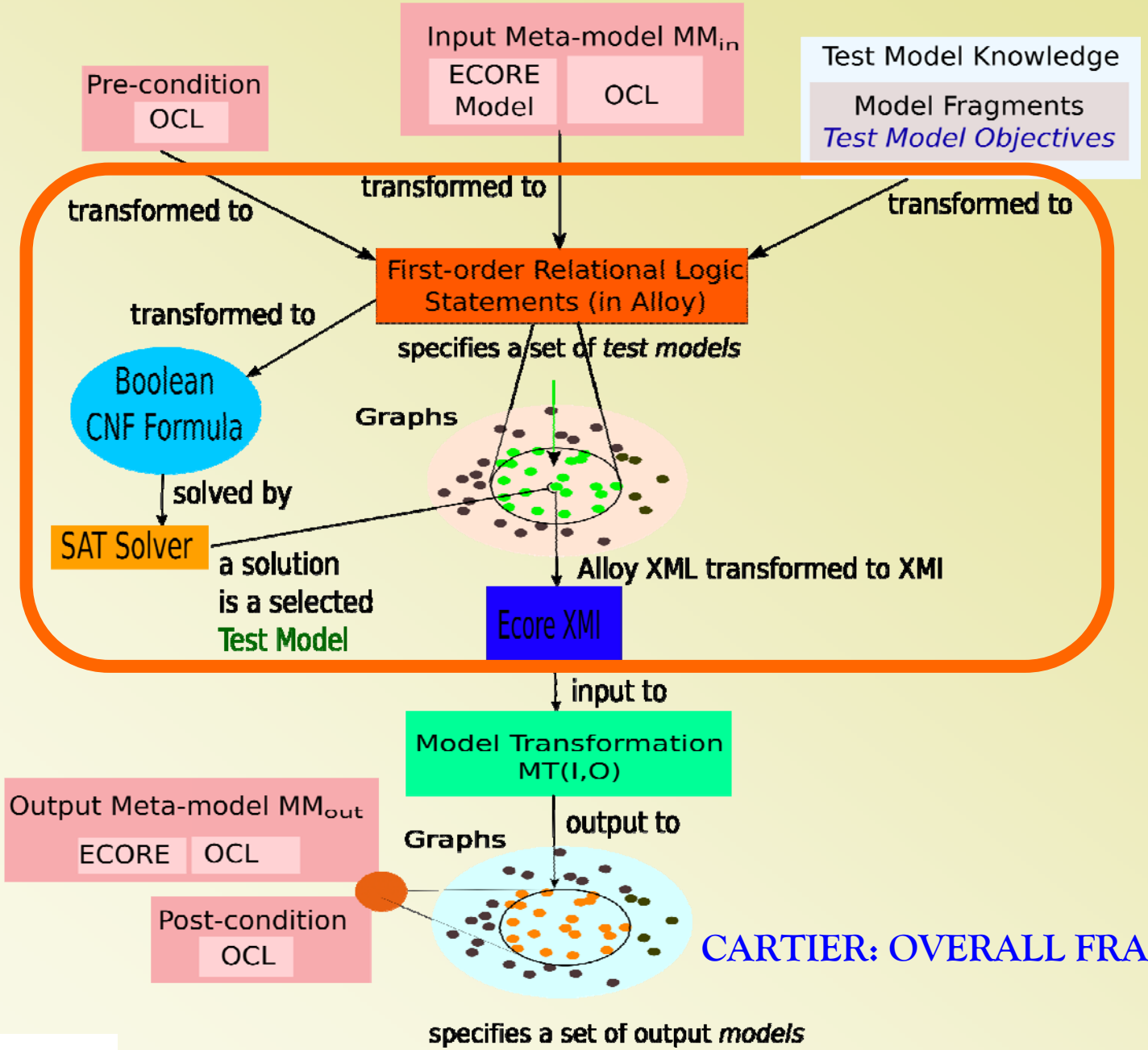
run test_requirement1 for 1 ClassModel,5 int, exactly 5 Class,
exactly 20 Attribute, exactly 4 PrimitiveDataType, exactly 5 Association

Exact number of objects

1. Specify a scope
2. Specify an exact number of objects

Output: Alloy model instance that satisfies meta-model + pre-condition + test_requirement1 and has the specified size

INRIA



CARTIER: OVERALL FRAMEWORK

INRIA

Perspectives on model synthesis

- Strenghten the tool
  - Automate what can be
- Experiments
- Design experiments to test model transformations
- We want to numerically estimate via *mutation analysis* the efficiency of test models

INRIA

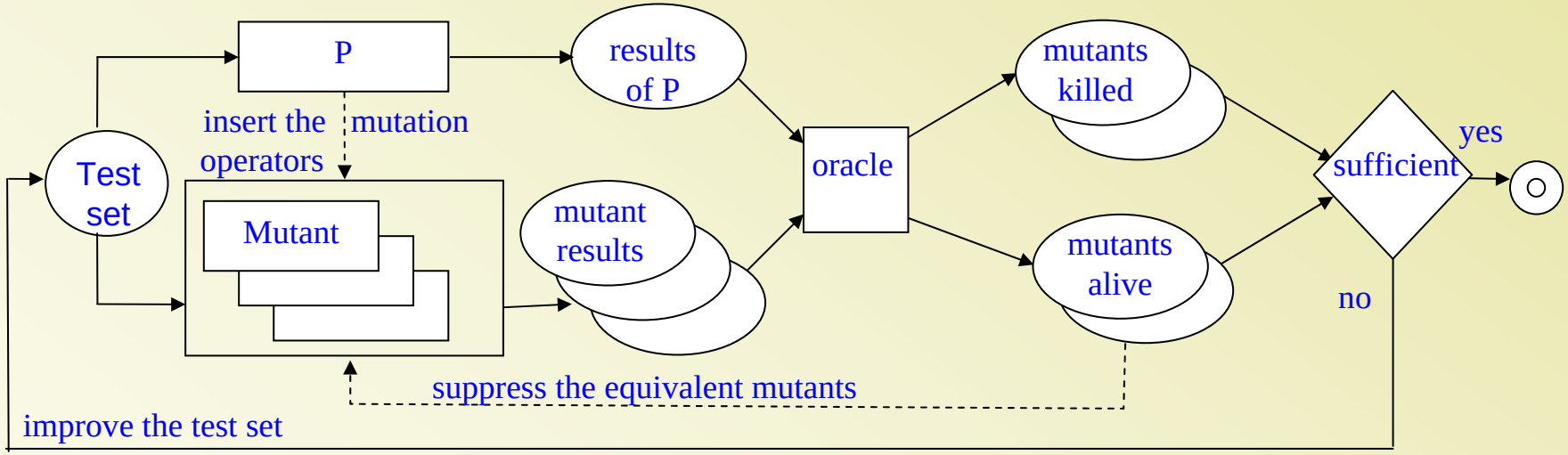
Mutation Analysis

INRIA

Mutation Analysis

- Evaluate the set of models
  - Producing a Mutation Score

Mutant	1	2	3	4	5	6	7	8	proportion
Killed	X	X		X	X			X	5/8



Mutation Analysis

- Analysis based on fault models
- Faults are based on syntax of programming languages
  - Most common errors
  - For procedural languages, OO languages…

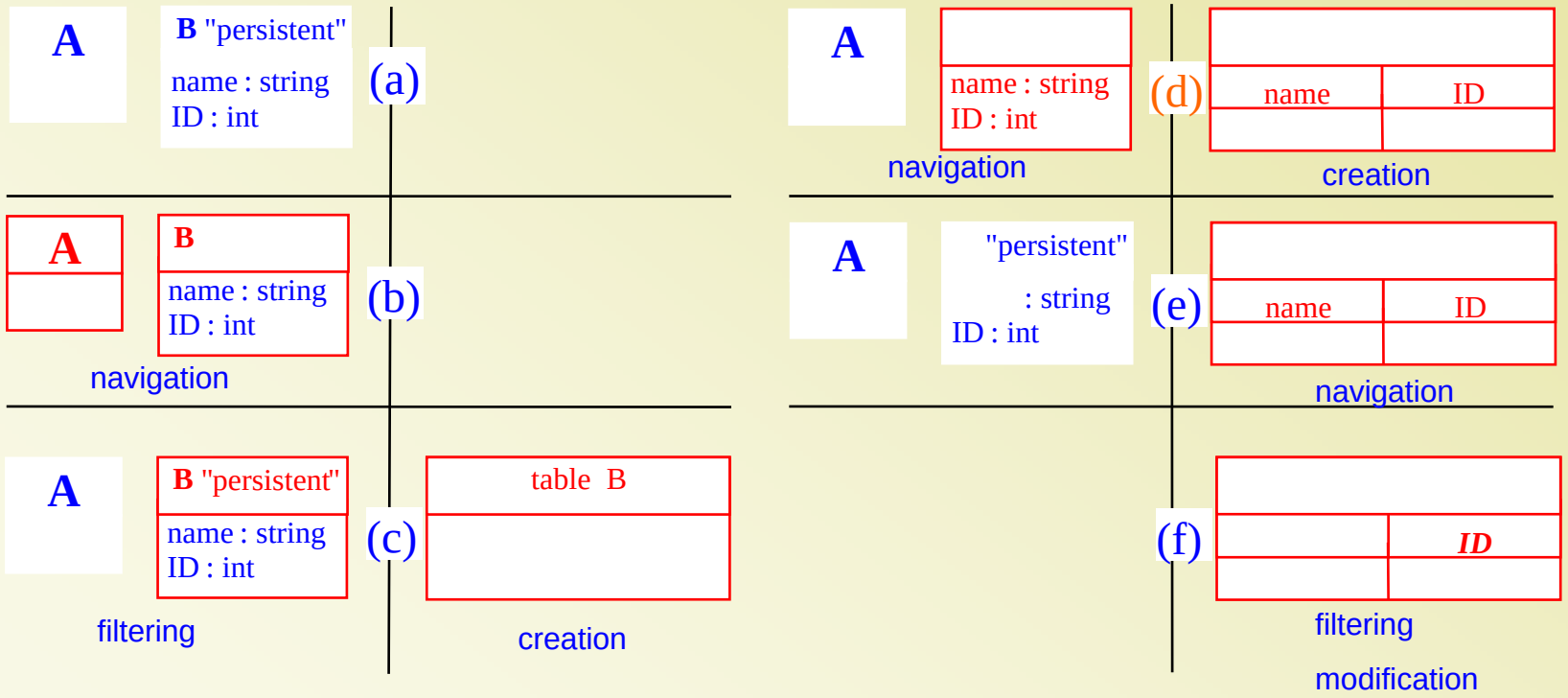
Mutation analysis for model transformation

- What errors occur in a model transformation?
- Implementation language independency
  - Too many different languages
- Lack data on common errors

INRIA

Abstract transformation operations

- Navigation, filtering, creation, modification
  - Example of one transformation



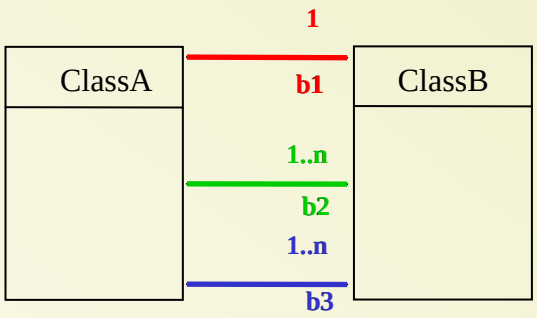
Mutation operators

- Navigation
  - Relation to the same class
  - Relation to another class
  - Relation sequence modification with deletion
  - Relation sequence modification with addition
- Filtering
  - Perturbation in the condition
  - Delete a predicate
  - Add a predicate
- Creation
  - Replace an object by a compatible one
  - Miss association creation
  - Add association creation

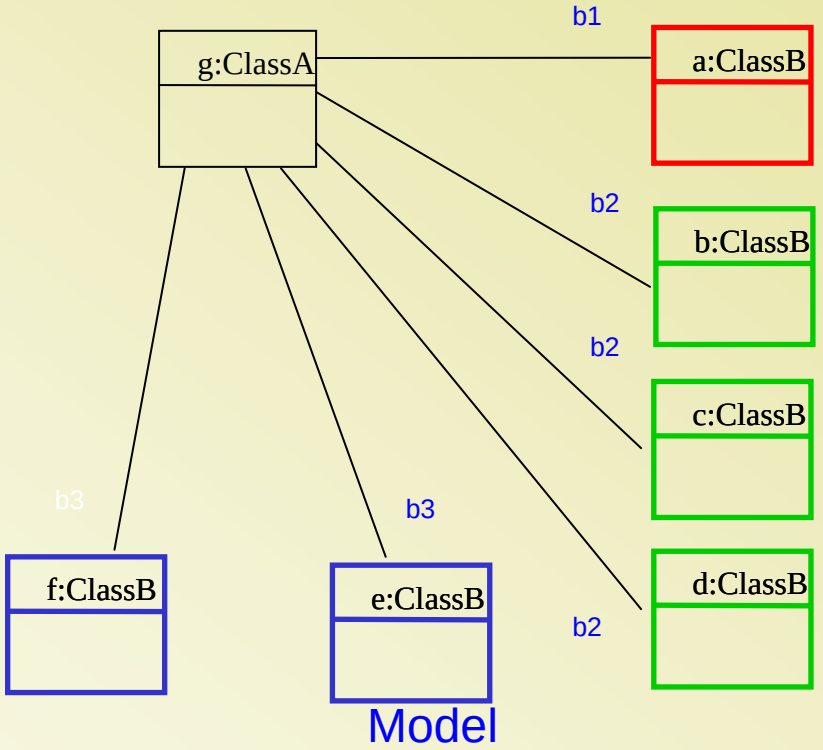
INRIA

One specific operator example

- Navigation
  - Relation to the Same Class Change - RSCC



Metamodel



Model

Mutation Analysis

- The proposed operator have been adapted to the Kermeta language
- Experiments:
  - To compare mutation operators
  - To evaluate the coverage criteria
  - To evaluate different knowledge for test generation

INRIA

Perspectives in Triskell

- Experiment!
  - We have spent a lot of time defining ideas and building the tools
- White-box techniques for specific languages
  - Specific adequacy criteria
  - Fault localization
- Oracle function definition

INRIA