

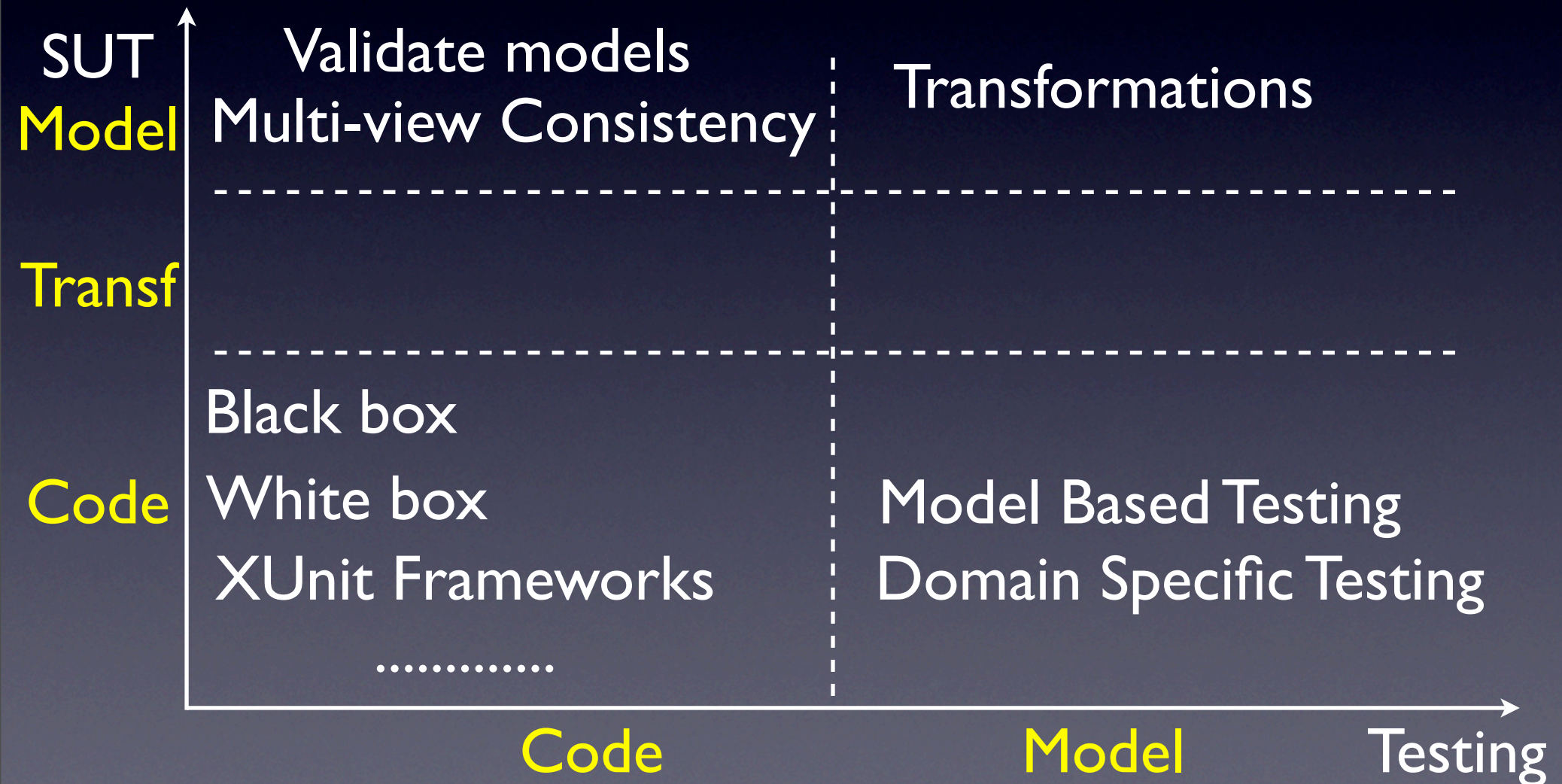
Challenges in testing Model Transformations

Amr Al-Mallah

Outline

- Overview/Motivation
- Systematic Software Testing
- Systematic Software Testing of Model Transformations.
- Focus : Model differencing

Overview



Software Testing

- Why are we testing ? (Test Objective)
- What are we testing ? (SUT)
- How are we testing ? (Test case selection)
- Testing oracles
- Testing process
- Test automation

Testing Activities

1. Generate Input test cases
2. Test Selection
3. Test Execution
4. Test oracle - verdicts
5. Results visualization for debugging and reports.

Why are we testing ?

- Functional testing
- Non-functional testing:
 - Performance
 - Reliability

SUT

- Model Transformations artifacts:
 - Textual Specifications
 - Input/Output Meta Models
 - Implementation:
 - Code, Rule-Based etc.

Test case selection

- Black box
- White box
- Hybrid

Test Case Selection

Black box

- Applicable to all languages
- Input meta model coverage
- Fleury et al:
 - EMOF based MM.
 - Coverage Criteria for Class Attributes, and Association End Multiplicities.
- Transformation specifications:
 - Effective Meta Model.

Test Case Selection

White box

- Language and tool specific
- Kuster et al :
 - Business Process models implemented in java code.
 - Conceptual Rules coverage => meta model templates as valid test cases.
 - Constraint coverage

Test Case Selection Hybrid

- Effective Meta Model:
 - Input Meta model (back box)
 - Examine the implementation to enhance meta model (white box) .
- Sen et al 2008 : Combining knowledge into Alloy constraints, and generate possible inputs.

Testing Oracles

- A function which produces a “pass/fail” verdict on the output of each test case.
- For each input, a corresponding output needs to be manually built.
 - Complex, error prone, procedure
- Some scenarios require further analysis than syntactic equivalence to produce a verdict.

Testing Oracles

- Model Comparison
- Contracts : post conditions on
- Patterns :
 - Model Fragments
 - Apply to specific inputs

Test Design Automation

- Generation tools based on the Black box approaches
- Mutation tools:
 - Sen et al 2006 : Himesis Mutation operators.

Execution Automation

- Construction of test case
- Executing the transformation
- Producing a verdict on the output
- Results visualization and reporting

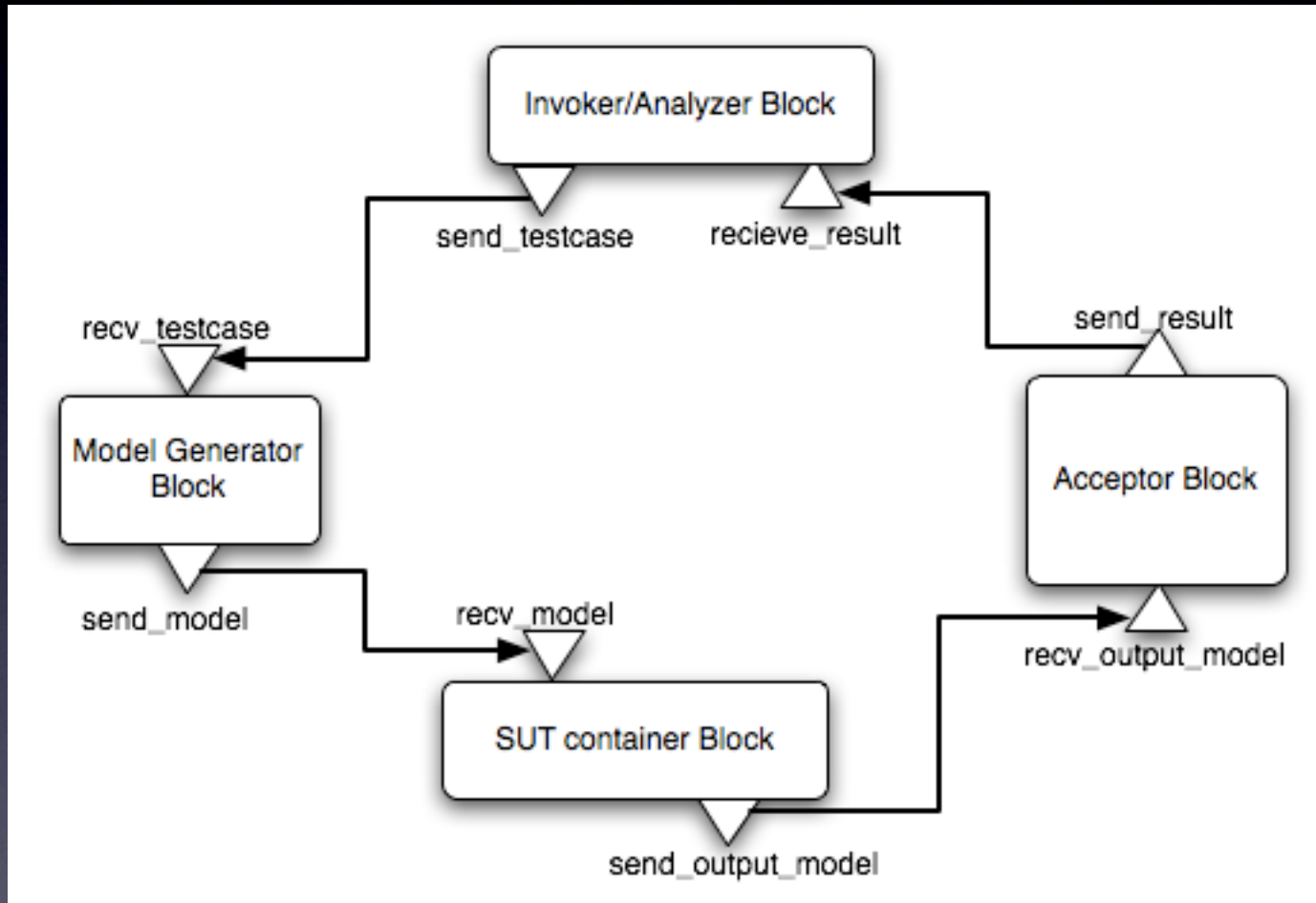
Testing Framework

- By line et al :
 - Endogenous transformations
 - Assumes a unique identifier for comparison.
 - Provides visualizations.
 - Integrated in GME, and C-Saw transformation engine.

TUnit

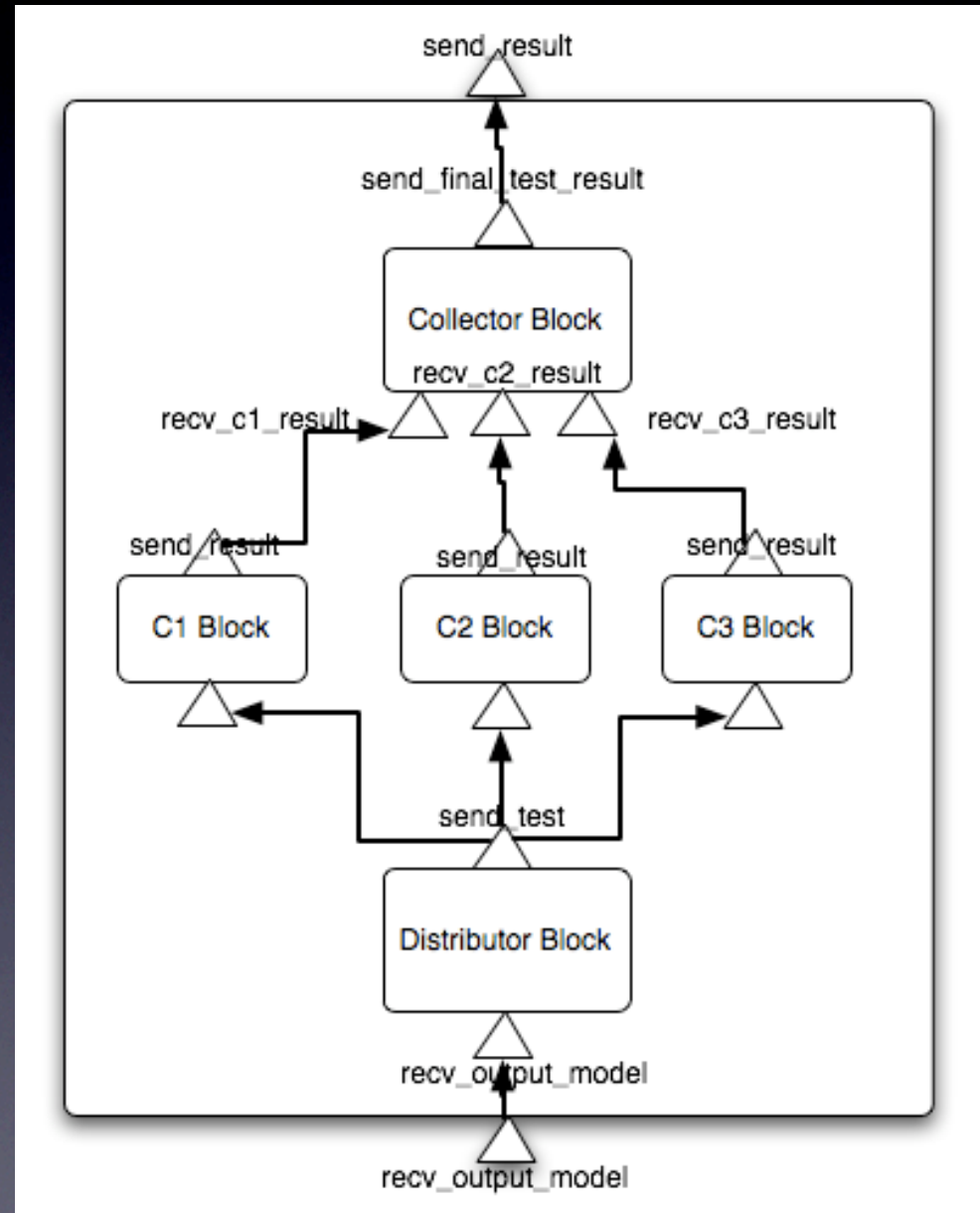
- “Model everything” = modeled .
- Supports model fragments and patterns.
- Supports time (based on DEVS)
- Runs independently.
- Extends PyUnit.

TUnit



TUnit

- Semantics equivalence
- Example :
 - Traffic 2 Petri Nets
 - Compare 2 Petri nets
 - Embed a transformation of PN to Reachability Graph



Model Comparison

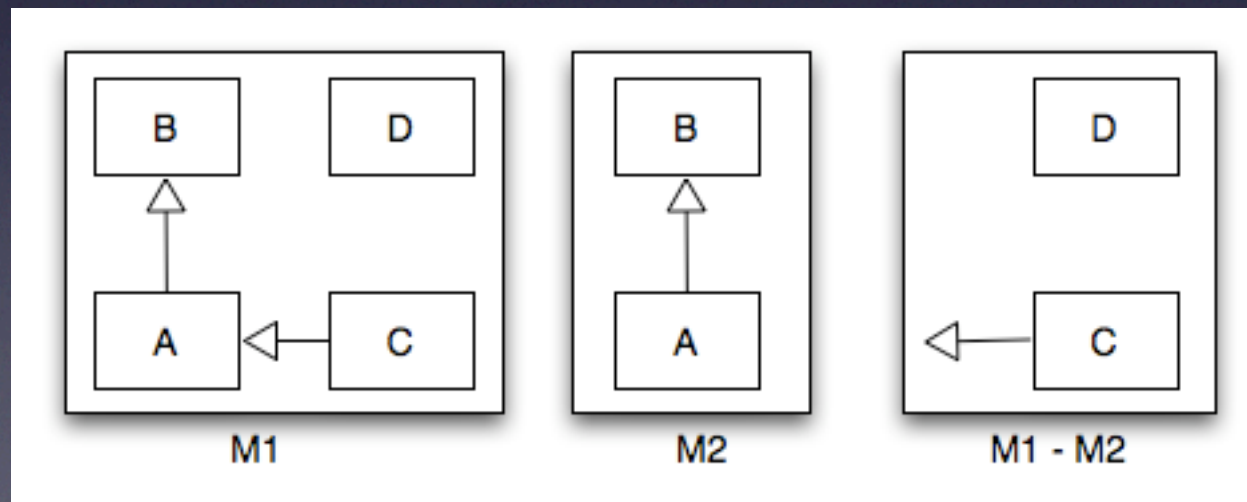
- Importance to MDE:
 - Model Evolution
 - Version Control
 - Transformation Testing

Model Comparison Activities

- Identify criteria for matching model elements. (Unique Identifiers, Matching attributes, same position... etc)
- Calculating and represent the difference. (Algorithm for matching, Edit Scripts)
- Visualize the differences. (Coloring, Difference Models)

Model Comparison

- Comparison is done on two models.
- Both models conform to the same meta model.

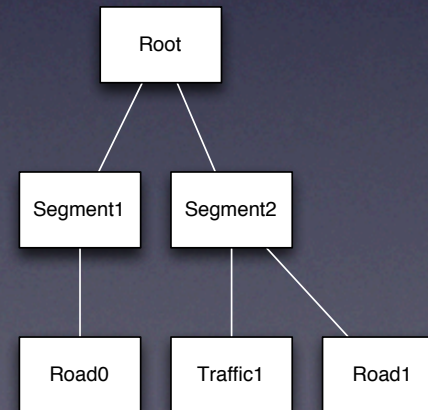
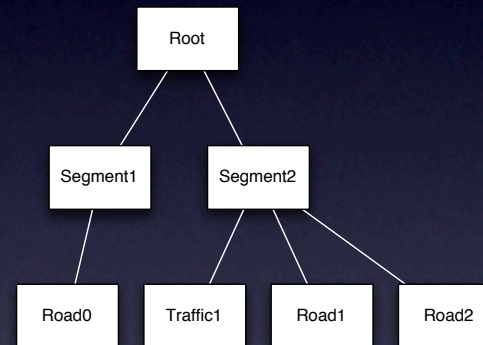


Model Comparison

- Can be applied to:
 - Abstract Syntax (Graph)
 - Concrete Syntax
 - Abstract Syntax of the semantic domain

Model Comparison

Concrete Syntax Comparison



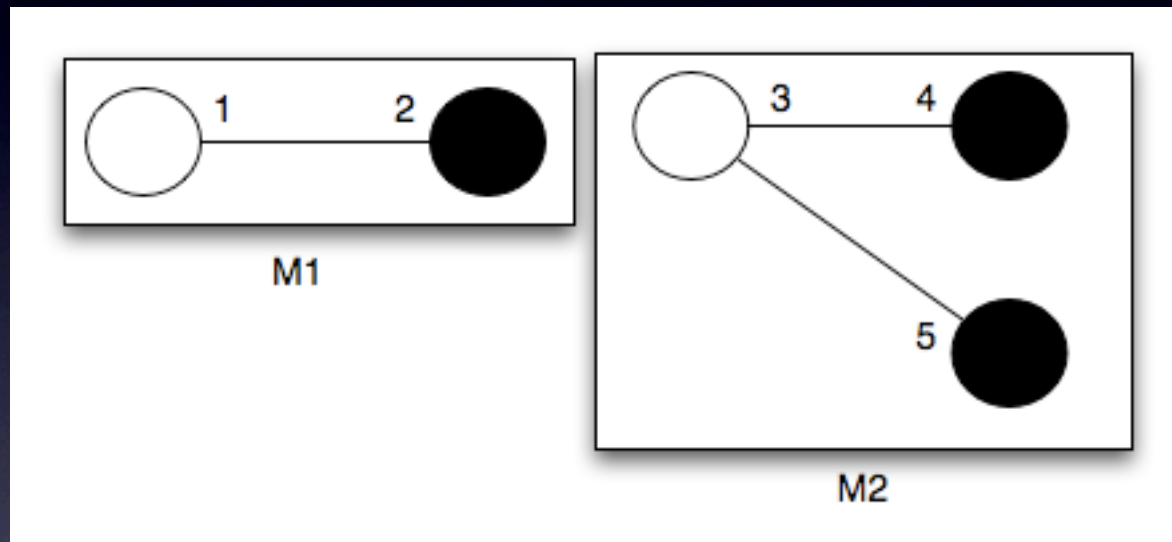
Model Comparison

- Models can not always be represented using trees
 - May have cyclical dependencies.
 - Search algorithm dependent.

Model Comparison

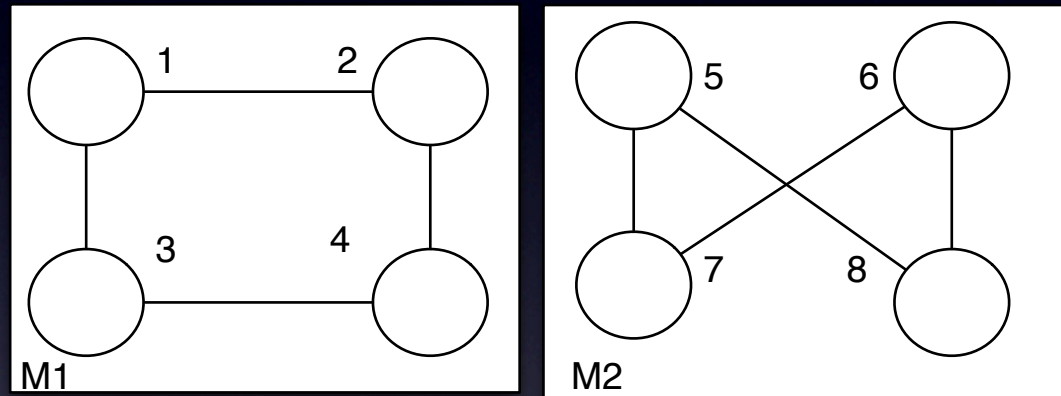
- Models are represented as Graphs.
- Graph matching is NP complete.
- Several workarounds have been proposed.

Model Comparison



Does v2 in M1 Maps to v4 or v4 in M2 ?

Graph Isomorphism Problem



Does M1 Map to M2 ? (are they isomorphic?)

Model Comparison Approaches

- Static Uniques Identifier
 - Each Model Element has A Global Unique Identity GUID
 - GUID is the matching criteria.
 - A simple sort would serve as the search algorithm. (Fast, Simple)
 - Environment, tool specific and dependent.

Model Comparison Approaches

- Dynamic Uniques Identifier (“**Signatures**”)
 - Each Model Element have a function to generate its unique identity.
 - The signature function has to be specified by the user with guaranteed uniqueness.
 - Related to using canonical forms.
 - Language, Structure specific.
 - Examples: **XMLDiff**.

Model Comparison Approaches

- Language Specific
 - Customized/Optimized for a specific language or a formalism.
 - Utilizes domain specific knowledge.
 - New Formalism = New full algorithm
 - Examples: **UMLDiff**, **State charts compare**.

Model Comparison Approaches

- Similarity Based
 - Assumes all models are typed attributed graphs. (or could be transformed to one)
 - Nodes are compared according to their feature similarities. (structural, or user defined)
 - work with any graph based models (meta model independent)
 - Examples: SiDiff, DSMDiff .

SiDiff

- Configurable for any model with graph structure.
- Models have to be first transformed into the internal representation (Directed, typed graphs).
- Users has to provide custom file for specifying nodes features to use in similarity matching, each with an associated weight.

SiDiff

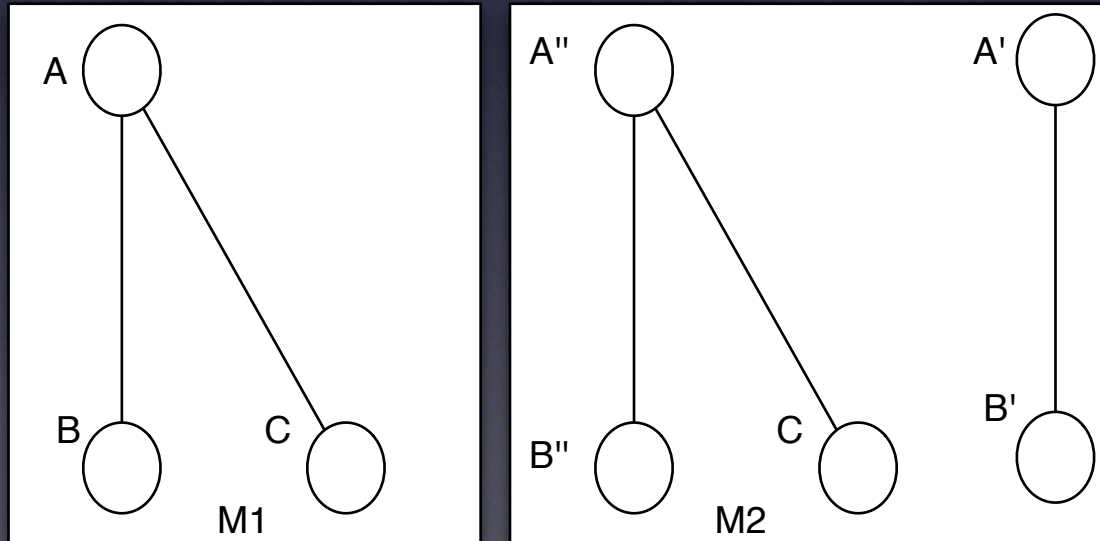
- Three phases
 - Hashing phase where hash value for each element is calculated. (rep as a vector)
 - Indexing phase, creating S3V trees which can efficiently find, for a given element, the most similar elements in the other model.
 - Matching phase where exact matches calculated by looping over each element of one model.

DSMDiff

- Claims to work on any Domain Specific Language which where MM is in GME.
- Uses signature matching, combined with structural similarities.
- Supports hierarchical graphs.
- Does not attempt to find optimal solution.
(Greedy Algorithm)
- Does not support move

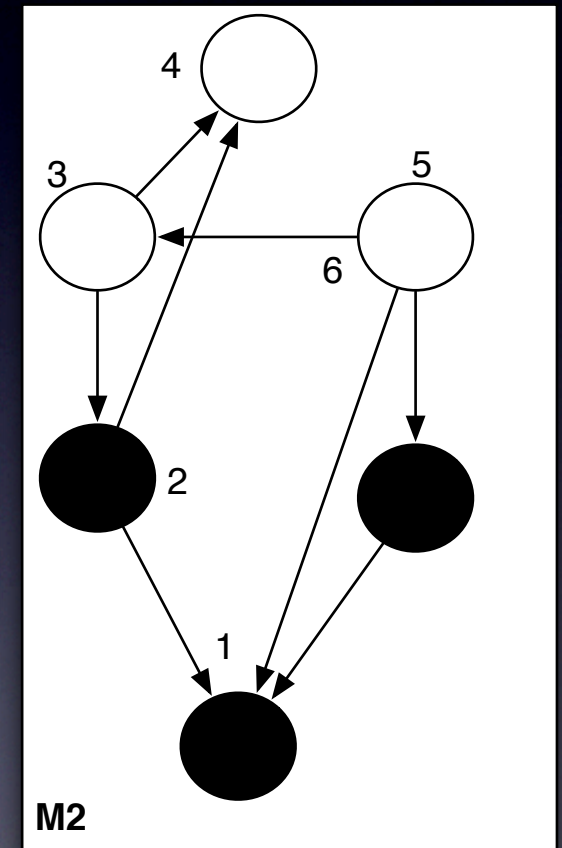
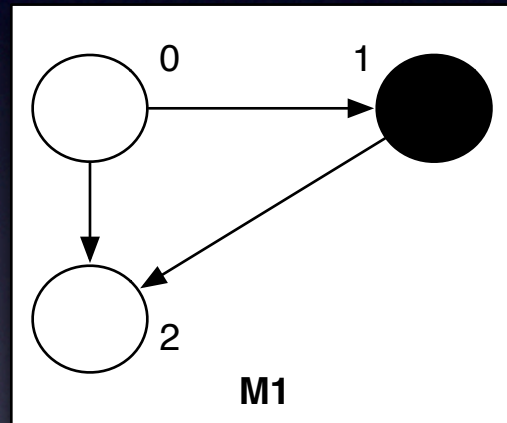
DSMDiff

- Limitations :
 - Can lead to incorrect solutions.
 - Does not support move operations.



Subgraph Isomorphism problem

- Find all possible occurrences of the pattern described in M1 in the host graph M2.



- Is this related to Model Comparison ?

Subgraph Isomorphism problem

- NP Complete Problem => Use Heuristics
 - Constraints solving problem.
 - Backtracking.
 - HVF by Marc Provost combines pruning techniques from many sources: VF, VF2 and Ulmann's Algorithm.
- Can we re-use these techniques ?

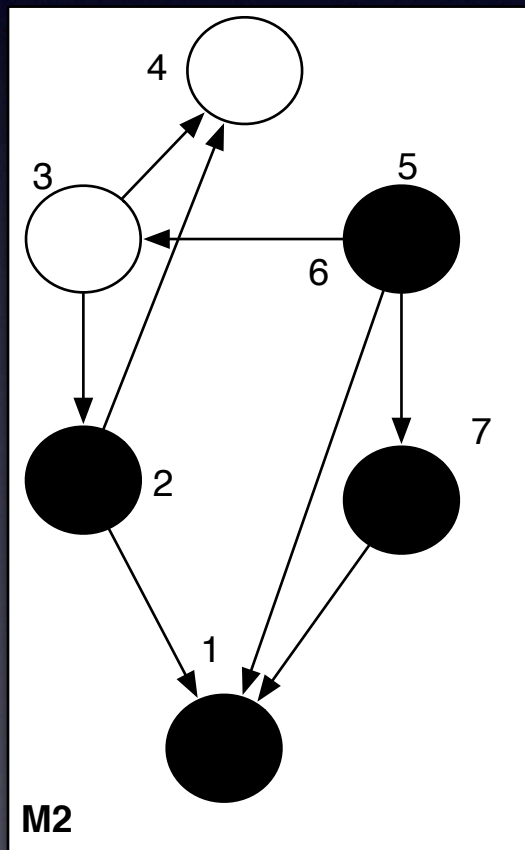
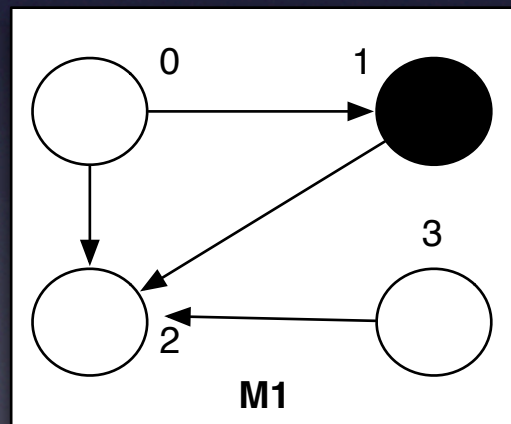
Subgraph Isomorphism problem

- Multiple Occurrences ? Multiple solutions ? one is optimal ?
- Model Comparison:
 - Either of the models can be the subgraph. (Size)
 - Not all of the elements in subgraph should exist necessarily in the host graph (Deleted Nodes).
 - More related to “Maximum Common Subgraph Isomorphism” problem.

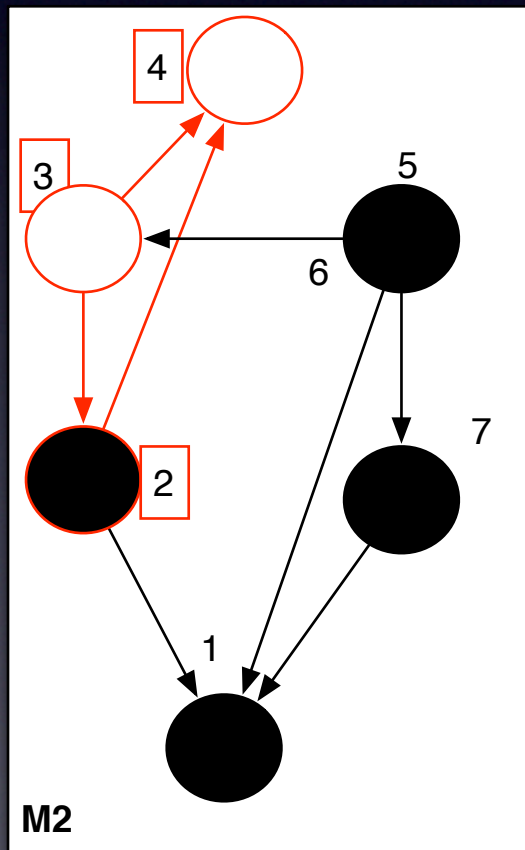
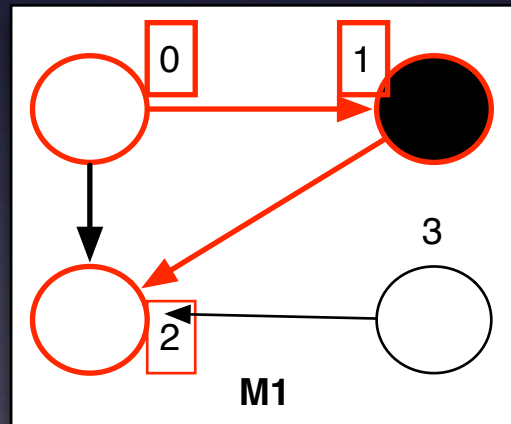
Maximum Common Induced Subgraph Isomorphism

- Induced Subgraph of graph G is : “ A set S of vertices of G , as well as the edges of G with both endpoints in S .”
- Common Induces Subgraph of graphs G_1 and G_2 is : “is a graph $G_{1,2}$ which is isomorphic to induced subgraphs of G_1 and G_2 .”

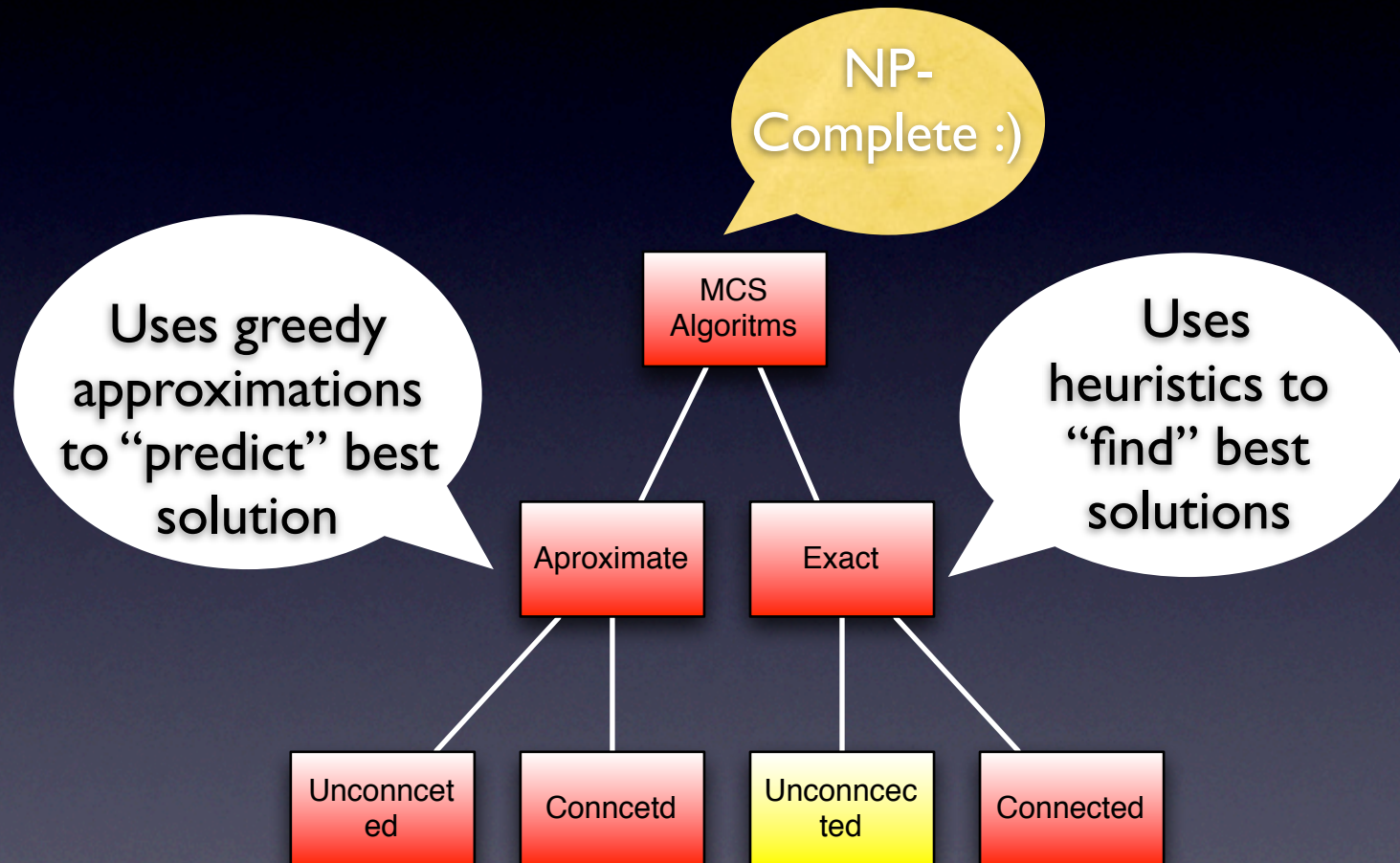
Maximum Common Induced Subgraph Isomorphism



Maximum Common Induced Subgraph Isomorphism



MCIS Algorithms



Next ?

- Look into solutions for comparing chemical compounds.
- Attempt to find the most effective algorithms and heuristics.
- Implement such algorithm into TUnit framework.
- Additionally allow the user to specify domain specific similarity features to enhance the pruning.