Tarski: A Platform for Automated Analysis of Traceability using Constraint Solver

Ferhat Erata, *Information Technology Group, Wageningen University, The Netherlands*

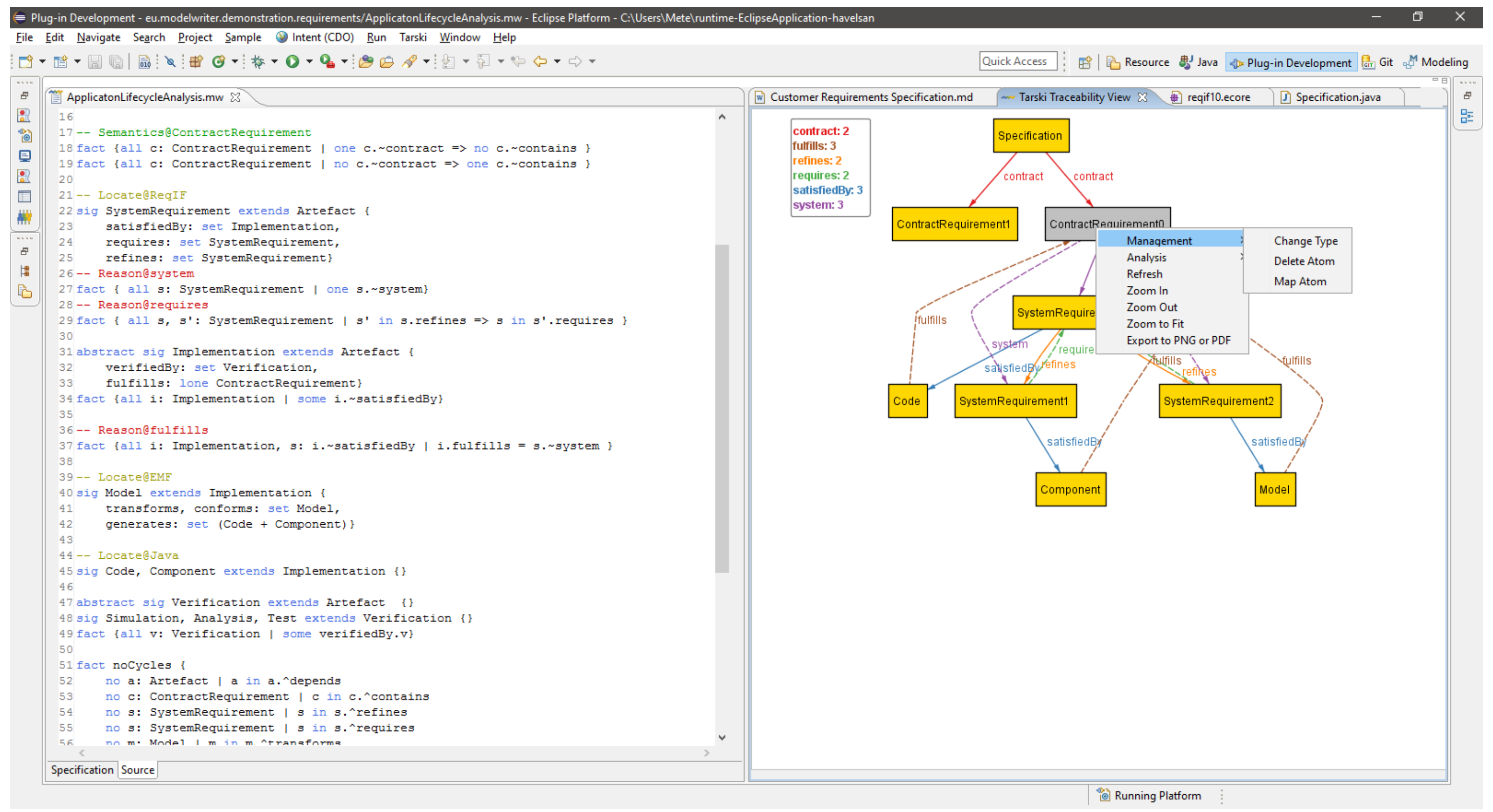
Introduction

In this project, we introduce a new approach with its supporting platform which enables the user to interactively configure and analyze trace relations between different work products (artefacts) of the development process. The user formalizes the semantics of traceability in first-order relational logic to perform automated analysis such as consistency checking, reasoning on trace relations and trace element discovery. The usefulness of the approach is demonstrated in the context of application life-cycle platform in software industry and being tested in aviation industry.

Approach

Formal Specification. The user is able to upload a formal specification to the Tarski platform to formalize the semantics of traceability in first-order relational logic [3] to perform consistency checking and reasoning on the trace instance. To address the scalability of the approach, we adopt the **lightweight formal method** [2] which makes analysis economically feasible.

Model Management. Tarski extracts a type hierarchy from the specification to enable the user to assign to trace elements. Furthermore, Tarski platform provides functions to modify the model.

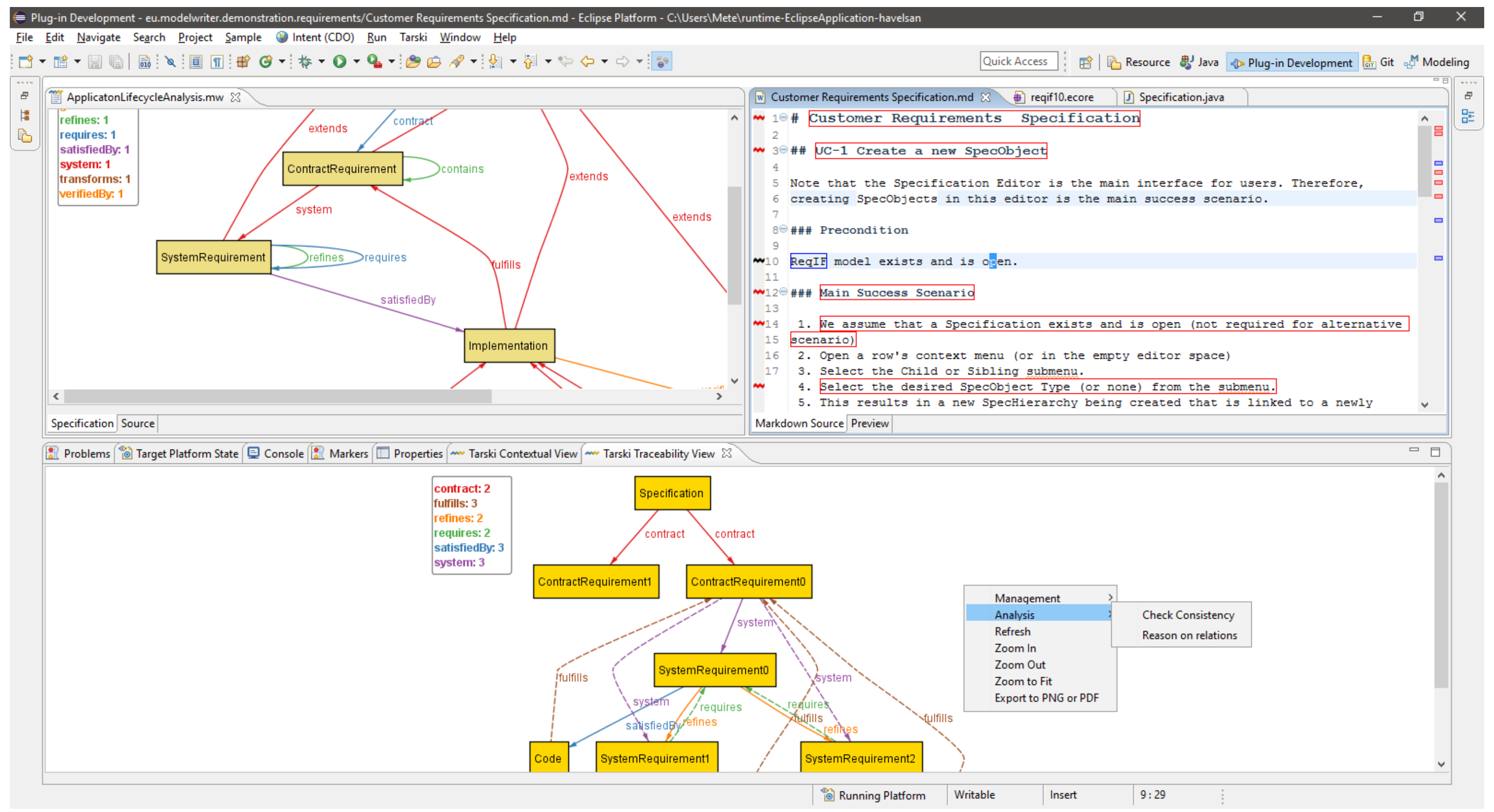

Plug-in Development - eu.modelwriter.demonstration.requirements/ApplicatonLifecycleAnalysis.mw - Eclipse Platform - C:\Users\Mete\runtime-EclipseApplication-havelsan

ApplicatonLifecycleAnalysis.mw

```

-- Semantics@ContractRequirement
fact {all c: ContractRequirement | one c.~contract => no c.~contains }
fact {all c: ContractRequirement | no c.~contract => one c.~contains }

-- Locate@ReqIF
sig SystemRequirement extends Artefact {
    satisfiedBy: set Implementation,
    requires: set SystemRequirement,
    refines: set SystemRequirement}
-- Reason@system
fact { all s: SystemRequirement | one s.~system}
-- Reason@requires
fact { all s, s': SystemRequirement | s' in s.refines => s in s'.requires }

abstract sig Implementation extends Artefact {
    verifiedBy: set Verification,
    fulfills: lone ContractRequirement}
fact {all i: Implementation | some i.~satisfiedBy}

-- Reason@fulfills
fact {all i: Implementation, s: i.~satisfiedBy | i.fulfills = s.~system }

-- Locate@EMF
sig Model extends Implementation {
    transforms, conforms: set Model,
    generates: set (Code + Component)}

-- Locate@Java
sig Code, Component extends Implementation {}

abstract sig Verification extends Artefact  {}
sig Simulation, Analysis, Test extends Verification {}
fact {all v: Verification | some verifiedBy.v}

fact noCycles {
    no a: Artefact | a in a.^depends
    no c: ContractRequirement | c in c.^contains
    no s: SystemRequirement | s in s.^refines
    no s: SystemRequirement | s in s.^requires
    no m: Model | m in m.^transforms
```

Consistency Checking. As each interaction of the user with the platform mutates the tuples in the universe, it is not guaranteed that the first order relational model satisfies the constraints. In Tarski platform, the first order model interpreted from the traceability information is automatically encoded in KodKod [4] using the exact bounds to check the possible inconsistencies and the satisfiability of the given model is reported to the user.

Reasoning on Relations. Inferencing is the activity of deducing new relations based on the semantics which is already specified. To realize this reasoning, user guide the reasoner using annotations to define the relation on which the reasoning is targeted.


Plug-in Development - eu.modelwriter.demonstration.requirements/Customer Requirements Specification.md - Eclipse Platform - C:\Users\Mete\runtime-EclipseApplication-havelsan

Customer Requirements Specification.md

```
# Customer Requirements  Specification

## UC-1 Create a new SpecObject

Note that the Specification Editor is the main interface for users. Therefore,
creating SpecObjects in this editor is the main success scenario.

### Precondition

ReqIF model exists and is open.

### Main Success Scenario

 1. We assume that a Specification exists and is open (not required for alternative
scenario)
 2. Open a row's context menu (or in the empty editor space)
 3. Select the Child or Sibling submenu.
 4. Select the desired SpecObject Type (or none) from the submenu.
 5. This results in a new SpecHierarchy being created that is linked to a newly
```

Trace Elements Discovery. In addition to the reasoning on relations, the proposed framework can reason on the provided trace elements and related relations to suggest the missing trace elements based on the semantic rules.

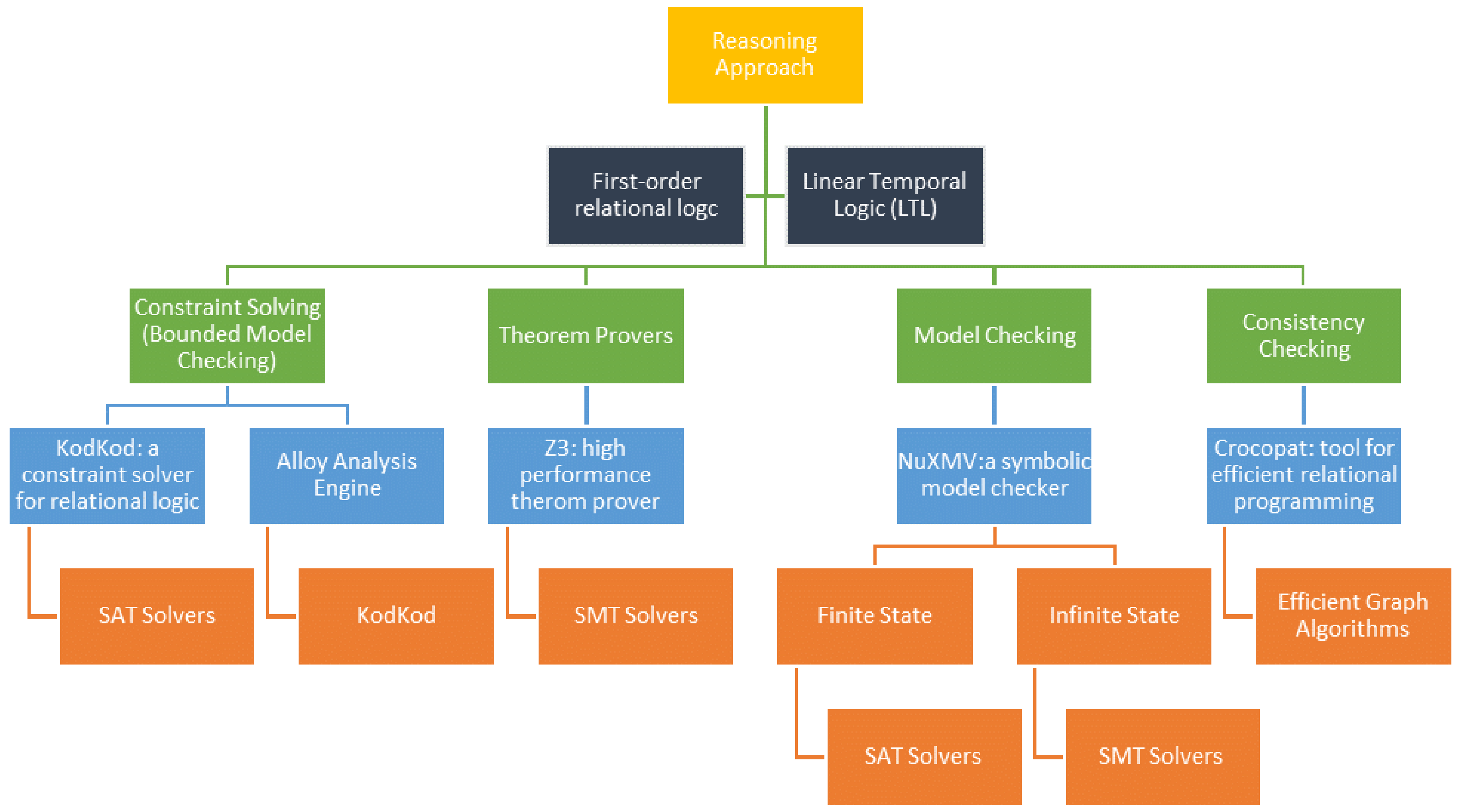
When the user accepts a suggested unary or binary relation on the visualization view of Tarski platform, the relation will be encoded as a new constraint of the model and further reasoning will be done on this new model. This mechanism will provide an evolutionary approach to develop the models.

Validation of the Approach

In this study, we have been validating the approach as a **Cyber-physical System** scenario in the formal analysis of different models and specifications produced by **Airbus**, a leading aircraft manufacturer in the world, along its system installation process. These artefacts constitute a set of design rules to formalize various aspects of the system installations on airplanes. Moreover, we've applied the approach on the **Application Lifecycle Management** platform of **Havelsan**, one of the largest systems and software company in Turkey.

Discussion

We have been working on integrating different decision procedures to incrementally and effectively repair broken trace links and discover new trace elements.


Reasoning Approach
First-order relational logic
Linear Temporal Logic (LTL)
Constraint Solving (Bounded Model Checking)
Theorem Provers
Model Checking
Consistency Checking
KodKod: a constraint solver for relational logic
Alloy Analysis Engine
Z3: high performance therom prover
NuXMV:a symbolic model checker
Crocopat: tool for efficient relational programming
SAT Solvers
KodKod
SMT Solvers
Finite State
Infinite State
Efficient Graph Algorithms
SAT Solvers
SMT Solvers

Acknowledgements

This work is partially supported by the Scientific and Technological Research Council of Turkey, Technology and Innovation Funding Programs Directorate and Minister for the Economy, Industry and Digital Affairs of France, Enterprise General Directorate.

References

[1] D. Beyer, A. Noack, and C. Lewerentz. Efficient relational calculation for software analysis. *IEEE Transactions on Software Engineering*, 31(2):137–149, Feb 2005.

[2] D. Jackson. *Lightweight Formal Methods*, pages 1–1. Springer Berlin Heidelberg, Berlin, Heidelberg, 2001.

[3] D. Jackson. Alloy: A lightweight object modelling notation. *ACM Transaction Software Engineering Methodology*, 11(2):256–290, Apr. 2002.

[4] E. Torlak. *A constraint solver for software engineering: finding models and cores of large relational specifications*. PhD thesis, Massachusetts Institute of Technology, 2008.