MŰEGYETEM 1782

Budapest University of Technology and Economics
Department of Measurement and Information Systems
Fault Tolerant Systems Research Group

Critical Architectures Laboratory
Spring Semester 2016/2017

Model-Driven SAT-Based Resource Allocation

Syllabus
v1.2

Author: Oszkár Semeráth
(semerath@mit.bme.hu)

March 18, 2017

Contents

1 Introduction

1.1 Overview: Reasoning on the Engineering Model

The advancement of formal methods made it possible to effectively aid the formal analysis of the modelling artifacts extensively used in model-driven engineering. In this laboratory we aim to give an insight into the technological and theoretical background through the analysis of data models by logic solvers.

Figure 1 shows the general approach of the analysis. Data models are typically defined in a *Domain*, which defines the metamodel and the design rules. Usually, the analysis is based on initial *Partial Models*, which define the design environment. The goal of the analysis is to provide valid models satisfying the design requirements. This is done by mapping the modelling problem to a logic problem, which is then solved by advanced logic reasoners. The result of the reasoning is a logic model, which can, in turn, be interpreted as a model of the target domain. The basic procedure consists of the following three steps:

1. The artifacts of the modelling tool are mapped to a logic problem.
2. The logic problem is solved by a reasoner.
3. The result of the analysis is traced back to a model of the analyzed domain, which satisfies the requirements.

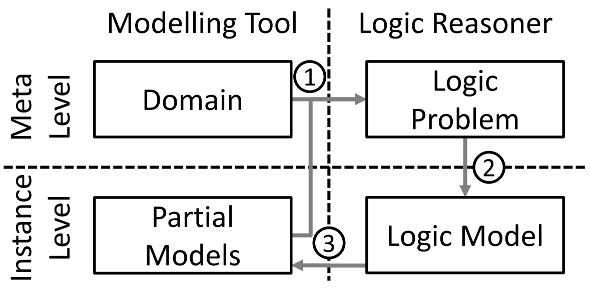

Figure 1: Logic analysis of domain-specific models

The approach will be illustrated on a resource allocation problem in the *Cyber-physical system (CPS)* domain.

The goal of this exercise are the following:

- To provide an introduction to automated logic analysis.
- To gain some experience with Boolean Satisfiability problems and the Alloy solver.
- As a continuation of the previous laboratory, we will overview how engineering problems can be mapped to logic problems in order to automatically provide solutions.
- In particular, to demonstrate how resource allocation problems of the CPS domain can be automatized.

The output of the previous laboratory is an Alloy source file, which defines the main concepts and relations of a CPS. In this laboratory, this code will be complemented with logic constraints, and the logic problem will be processed by the Alloy Analyzer.

1.2 Resource Allocation Problem of CPS

Cyber-physical systems (CPS) are on one hand close to *embedded systems* as they are also built from sensors, controllers and actuators, where the sensors gather heterogeneous information from the environment, the controllers observe the gathered information and order the actuators to modify the environment according to the observed information. On the other hand, CPS systems are aiming to harvest the benefits of elastic cloud based resources to provide even more complex automation services.

The mapping of the different tasks to the specific hardware is a challenging task, because (1) the software and hardware architecture can be complex, (2) several design rules must be adhered to, and (3) the developer sought to optimise different properties of the system, like hardware cost-effectiveness or reliability. In this laboratory the software allocation problem is mapped to a Boolean satisfiability problem, which is automatically solved by the Alloy Analyzer.

2 SAT-Solving by the Alloy Analyzer

2.1 Boolean Satisfiability Problem

The Boolean Satisfiability Problem is a formally specified scheme of decision problems.

> "A propositional logic formula, also called Boolean expression, is built from variables, operators AND (conjunction, also denoted by $\wedge$), OR (disjunction, $\vee$), NOT (negation, neg), and parentheses. A formula is said to be satisfiable if it can be made TRUE by assigning appropriate logical values (i.e. TRUE, FALSE) to its variables. The Boolean satisfiability problem (SAT) is, given a formula, to check whether it is satisfiable. This decision problem is of central importance in various areas of computer science, including theoretical computer science, complexity theory, algorithmics, and artificial intelligence." [4]

The following example shows an example SAT-problem, where two tasks (t_1 and t_2) have to be allocated to different computers (c_1 and c_2). The right hand side shows a possible solution to the problem.

Problem	**Solution**
Variables:	*Interpretation:*
t_1c_1 = Task 1 $\rightarrow$ Computer 1	t_1c_1 = TRUE
t_1c_2 = Task 1 $\rightarrow$ Computer 2	t_1c_2 = FALSE
t_2c_1 = Task 2 $\rightarrow$ Computer 1	t_2c_1 = FALSE
t_2c_2 = Task 2 $\rightarrow$ Computer 2	t_2c_2 = TRUE
Constraint:	*Satisfied Constraint:*
$\neg((t_1c_1 \wedge t_2c_1) \vee ((t_1c_2 \wedge t_2c_2)))$	$\neg((T \wedge F) \vee (F \wedge T))) = T$

SAT is a decidable fragment of logic problems and there are several advanced SAT-solver applications which are able to effectively solve SAT problems.

2.2 The Alloy Language and Analyzer

Alloy provides a logic language for describing graph-structures and conveniently define constraints with higher-order logic expressions (quantified with $\exists$ and $\forall$) expressions and relational algebra (join operators, $\subseteq$, $\cup$, $\cap$). The Alloy Analyzer tool translates the logic problem to pure SAT, solves it and visualises the results.

Some advantages of the Alloy Analyzer are listed below:

- It has been actively developed for more than 10 years. The application is stable and proven.
- The Analyzer tool is a light-weight, small and multiplatform Java application.
- It has been used in a wide range of applications from finding holes in security mechanisms to designing telephone switching networks.
- It provides a tool that is able to consistently solve a wide and well-defined range of problems.
- It is very well-documented, and thus easy to learn and use.
- It is easily incorporated in automated workflows or integrated into a research prototype.

To learn the use of the Alloy Analyizer, read the following manuals: [3] (Only File System Lesson I-IV are needed). Make yourself familiar with the reference manuals at [2]. You can download and try the Alloy Analyzer at [1].

3 Tasks

In preparation for the class, read the tutorial and familiarize yourself with the referred manuals. Download Alloy Analyzer, and check some examples. During the class, do the tasks listed below.

3.1 Initial Model Generation

Try the Alloy Analyzer with the result of the previous lecture.

1. Start the the Alloy Analyzer, and load the final code generated in the previous laboratory.
2. Try to execute the satisfiability problem. If the code contains any syntax error, correct it. Certainly, the analyzer is able to find a valid instance easily.
3. Generate some examples and examine random graph outputs. In the next steps additional facts will be added to the code in order to generate models representing valid CPS configurations.

3.2 Structural Constraints

Add structural constraints to the Alloy problem in order to represent valid models.

- **Important:** You *only* need to edit the Alloy code for this specific metamodel, not the generic code generator of the previous laboratory!
- **Important:** Generate models after each step with small scopes to check your solution.

- **Important:** Do not generate string attributes, or add **`exactly`** `3 String` to the scope of the run configuration in order to represent names in the logic the problem. (Otherwise the problem will be unsatisfiable, because it tries to create models without any strings, what contradicts the lower multiplicity of an attribute.)
- **Important:** If you have scalability issues, try to remove unnecesary generic supertypes (like `EObject`, `NamedElement`, `CPSElement`) from the problem.

1. It is challenging to represent integers as a (Boolean) SAT problem. In order to tackle this difficulty, extract the attribute slot to individual objects. Those objects will represent the resources of the computers. Figure 2 shows the new metamodel. Implement this change in the Alloy code.

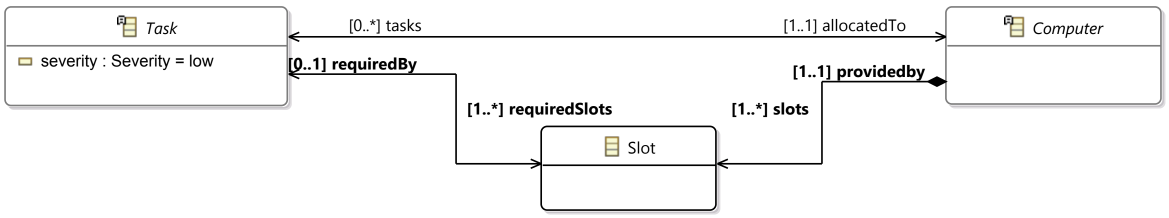

Figure 2: New `Slot` type

2. Visualise the new metamodel with Execute | Show Metamodel. The result should be similar to the one illustrated on Figure 3.

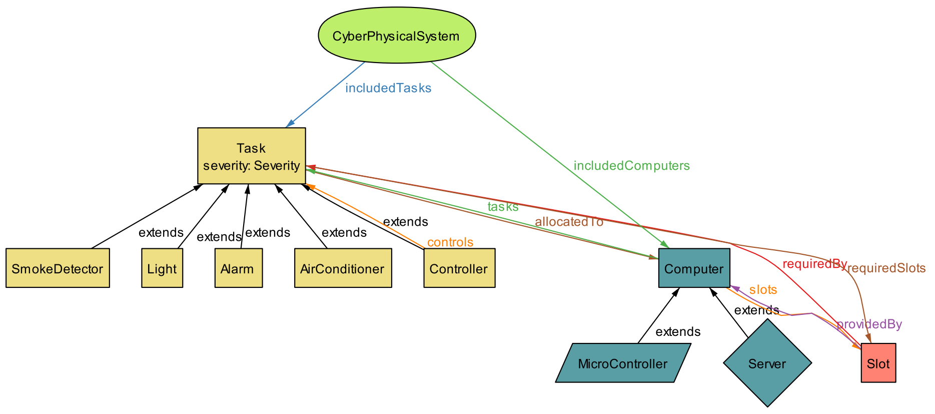

Figure 3: Metamodel of the CPS visualised by Alloy Analyzer

3. Define the inverse directions in the metamodel (navigation to the opposite direction).
4. Define the containment hierarchy in this metamodel: There should be one `CyberPhysicalSystem`, and each object should be connected to it (containment relation).

3.3 Additional Validation Constraint

3.3.1 Navigation Constraints

1. Tasks uses the slots of the Computer to which they are allocated.

2. Controller controls only a non-controller task.
3. Each Task except the controller has to be deployed to MicroControllers.

3.3.2 Cardinality Constraints

1. A microcontroller has 1-3 slots.
2. A server provides up to 10 slots.

3.3.3 Security Level Constraints

Write the following fact which defines that the AirConditioner is a Low Severity Level task:

```
fact{ AirConditioner.severity in Low }
```

1. What would be be the difference if the operator "**in**" would be changed to the operator "=" in the code?

Extend the model with the following constraints:

2. The Light can be Medium or High severity.
3. The Smoke Detectors and Alarms are Critical tasks.
4. The controllers severity is equal with the controlled unit severity.

3.3.4 Advanced Constraints

1. Critical and noncritical tasks can not be mixed in a computer.
2. Controller tasks in one computer should control different tasks. It is unnecesary to deploy multiple controllers with the same target to a computer.

In the end, the Alloy Analyzer generates valid instance models that satisfy each requirement. A possible result which contains each type of element is depicted in Figure 4. The following scope is used for this model:

```
run {} for
  exactly 1 CyberPhysicalSystem,
  exactly 2 MicroController,
  exactly 1 Server,
  exactly 7 Slot,
  exactly 1 Controller,
  exactly 1 SmokeDetector,
  exactly 1 Alarm,
  exactly 1 Light,
  exactly 1 AirConditioner
```

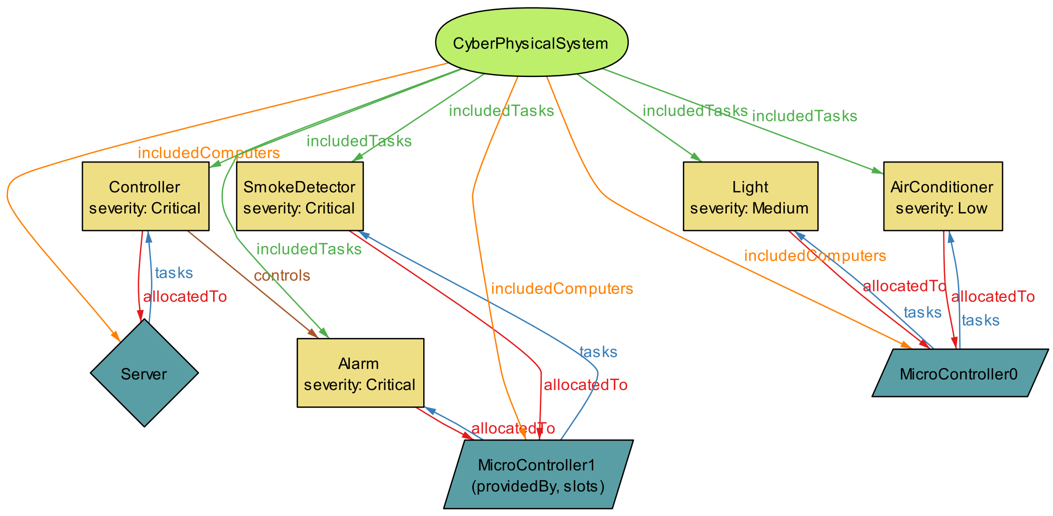

Figure 4: Example of a CPS that contains each type of (Slots are excluded)

3.4 Software Configuration Allocation

Figure 5 shows an example software configuration of two controllers, a Smoke Detector and a Light module.

1. Create a fact that ensures the generated models contain this initial structure. (Pay attention to the)
2. Show an example configuration.
3. Prove that the allocation can not be realized without at least two MicroController. It can be checked for the scope 50 with the following scope:

   ```
   run {} for 50 but exactly 1 MicroController
   ```

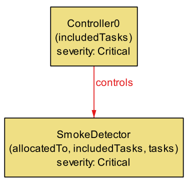

Figure 5: A concrete software configuration

4 List of Questions

Read this document and the referred [3] tutorial, then answer the following questions:

1. Define the signature of a `Folder` type which contains arbitrary number of `subfolders`!
2. Write an example to abstract signatures and inheritance!

3. What kind of multiplicities are available in the Alloy language?
4. Define a scope for generating a model with 10 `Folder`!
5. What is the difference between the "=" and the "**in**" operators?
6. What does it mean if the Alloy Analyzer shows the following message: "*Instance found. Predicate is consistent.*"?
7. What does it mean if the Alloy Analyzer shows the following message: "*No instance found. Predicate may be inconsistent.*"?

---

1. `sig Folder { subfolder: set Folder }`
2. `abstract sig A extends B{ ... }`
3. `lone = 0..1, one = 1..1, some = 1..*, set = 0..*`
4. `run {} for 10 Folder`
5. "`A=B`" means that that the two set is equivalent ($A = B$), "`A in B`" means that $A \subseteq B$
6. The logic problem is satisfiable, and the tool is able to show an example model for the given scope.
7. The SAT-problem is unsatisfiable for the given scope.

References

[1] Alloy analyzer. `http://alloy.mit.edu/alloy/download.html`.

[2] Alloy language: Referrence Manual. `http://alloy.mit.edu/alloy/documentation/book-chapters/alloy-language-reference.pdf`.

[3] Alloy Tutorial: File System Lesson I-IV. `http://alloy.mit.edu/alloy/tutorials/online/frame-FS-1.html`.

[4] SAT-Problem. `http://en.wikipedia.org/wiki/Boolean_satisfiability_problem`.