UCL Université catholique de Louvain

École polytechnique de Louvain (EPL)

Modelling and analysing parliamentary law with Alloy

Dissertation presented by
Clémentine Zaninka MUNYABARENZI

for obtaining the Master's degree in
Computer Science

Supervisor(s)
Charles PECHEUR

Reader(s)
Guillaume MAUDOUX, Antoine CAILLIAU

Academic year 2015-2016

Contents

Abstract

Assemblies ranging from small non-profit organizations to state parliamentary assemblies rely on the rigorousness of the rules and protocols suggested by the *Robert's Rules of Order Newly Revised* (RONR) to take important decisions. In the latter case these protocols turn out being a decisive tool at the very heart of the democracy of nations.

The Rules Of Order as described in the RONR [1] appear under the form of a collection of sentences formulated in the natural language. These sentences are aimed at achieving an abstract, equally natural-language-formulated description of goals. The focus of this work is to produce a formal *specification* out of the study of the RONR using the MIT-implemented *Alloy* modelling tool [4] and its modelling language based on relational algebra and first-order logic. The problem to be solved is thus the construction of a formal object that is best capable of *revealing* on one hand and *verifying* on the other the interesting properties of the Rules of Order with respect to its goals.

Proceeding to a first formal analysis of this renowned collection of rules whose constitution was made to some extent along empirical and historical developments is in this regard a task that makes sense and that can come upon interesting results regarding the extent to which the protocols formally succeed to fulfil the theoretical goals of *a fair and efficient debate* in order to get business done within a decisional assembly.

Acknowledgments

I would like to thank my promoter Charles Pecheur and the person of contact for my thesis Guillaume Maudoux.
I would also like to thank Delphine Umuhire, Francine Mukeshimana, Stéphane Vercouillie and Mike Gueye for the final support.
I thank my colleague students for these instructional years of collaboration.
I thank every person that had an encouraging thought for me.

Chapter 1

Introduction

1.1 Context

When building systems, practical implementation can deviate from the intended design. Formal verification can be defined as the mapping of a real-life system into an analysable mathematical object allowing the use of mathematical reasoning to enforce that design goals are met[2].

We thus verify the validity of various types of processes by capturing their interesting characteristics inside models. Types of processes can be for instance physical systems such as railway systems or software applications such as client-server systems. The characteristics we capture are interesting with respect to the properties of the systems that we would like to observe. The absence of train collisions for example in the case of a railway system or the absence of an un-authenticated access to data in the case of a client-server system. One among the types of processes that are analysed through models are *Business processes*.

According to the *Workflow Management Coalition Terminology & Glossary* [8] , a *Business Process* can be defined as *"A set of one or more linked procedures or activities which collectively realize a business objective or policy goal, normally within the context of an organizational structure defining functional roles and relationships."* .
In Martyn A. Ould's paper *Business Process Management: A Rigorous Approach*, [9] it is defined as *"a coherent set of activities carried out by a collaborating group to achieve a goal, where the chunking of organizational activity into processes must be driven by an understanding of the business the organization is in."*

In this work, we are interested in the business processes that lead to the adoption of decisions within gatherings that fall under the description of *deliberative assemblies* as described by the *Robert's Rules Of Order Newly Revised* (RONR)[1], a reference work in this domain. A deliberative assembly is defined by the latter reference as a *"a body of persons meeting under certain conditions to discuss and determine upon common action."* [1, p. xxix-xxx]. This fulfils the requirements stated by the previous definitions of an "organizational structure" or "collaborating group" in order to "achieve/realize a goal", as it happens, elaborating parilamentary law. Parliamentary law and parliamentary law procedures being all together referred to by the RONR as *parliamentary law* proceeds from the definition of an organisation's law not only as the prescription it enunciates but also as how it was brought to be accepted by all as enunciating this prescription.

To his day, there exists no formal verification system that is a universal solution to all modelling problems. Each target domain has different characteristics and aims at its own goals. Moreover, domains in which rules are progressively derived from experience may have intrinsic characteristics that it could be hard to deduce without a formal analysis, for instance, rare and corner cases that have not yet occurred but whose occurrence could have important implications. According to the

type of problem, the level of complexity and the form of its specification, a formal verification method is to be chosen for each modelling problem.

In this regard, we view the Alloy modelling tool developed by the Massachusetts Institute of Technoligy (MIT) as a promising tool in the field of revealing system properties. Its designing language makes use of a relational paradigm in order to describe the represented universe of discourse and by reducing to boolean satisfiability solving (SAT), it offers the possibility to leverage the power of state of the art solvers while being able to rely on optimizations in order to limit the explosion of the model and discover in priority the interesting interpretations.

1.2 Objectives

The goal of this work is firstly to identify a significant and consistent subset of the Robert's Rules Of Order to be modelled into a formal verification system. The question as to what aspects of the domain are more interesting to capture will be addressed by first identifying what is formalizable from what is not and then choosing according to the goals of the system or other heuristics the concepts that will effectively be modelled. Secondly, we shall formulate an verify properties such as the absence of deadlocks, progress and completion of tasks, the absence of conflicting situations, etc. from the analysis of which we will be able to draw some informative conclusions. *What does analysis reveal? What can we learn about the domain? What are the limits of the analysis?* Are the questions to be answered in the second phase of the work.
As an environment for our study we designate the MIT Alloy verification framework which combines an attractive set of formal verification tools whose strengths we will leverage in the completion of this task.

Considering part of the task consisted in getting familiar with the Rules Of Order, our approach to achieve these goals has been in the beginning to kill two birds with one stone by, using the flexibility allowed by the Alloy specification language in order to translate the rules from natural language to elementary specifications thus letting the base structure of the domain emerge. Afterwards, the two above mentioned phases - modelling and analysing - had to be alternated in an as iterative as possible way in order to steer the modelling in the right direction notably regarding the properties that became interesting to study in more details on one hand and on the other hand the resource constraints.

Our study seeks to contribute to the works in the domain of modelling and analysing by :

- suggesting a *formal view of the parliamentary law elaboration process* as described by the Robert's Rules Of Order Newly Revised which is an institution in the latter field;
- building a significantly complex *Alloy model* based on the analysis of a real world process;
- conducting a *study of the correctness and cohesion* of the Rules Of Order as described by the Robert's Rules Of Order Newly Revised.

1.3 Roadmap

The remainder of this document is structured as follows : the next part gives the reader some background material about the Alloy tool and the content of the Roberts Rules Of Order Newly Revised. The problem to be solved is then stated in more precise terms. We shall layout the scope of our work and the factors involved in the definition of this scope and then describe the modelling phase by a walk through of the models that have been designed. The analysis phase will afterwards be described and discussed with respect to properties, their verification and the outcomes of these verifications. Finally we shall conclude and give some incentives about future work.

Chapter 2

Background Material and Problem Identification

2.1 Formal Foundations : Relational algebra related to first-order logic and object-oriented design

As stated in the previous section, using Alloy as a modelling tool implies that we are looking at the modelled domain using a relational paradigm, that is, through the lens of mathematical relations. Objects having links between each other are described using sets, tuples, facts, predicates and types. Definitions of these concepts, and notions of the relationships existing between them is therefore handy in order to approach the Alloy relational paradigm.

Relations : An *n-ary relation* is a subset of the Cartesian product of $n \in \mathbb{N}$ sets of objects. Relations can thus be represented under the form of sets of tuples with a degree equal to n.
For example, considering the sets of objects $A = \{a_0, a_1, a_2\}$ and $B = \{b_0, b_1, b_2\}$, the set of tuples $R = \{(a_0, b_1), (a_0, b_2), (a_2, b_2)\}$ is one possible binary relation between the sets A and B.
A Relational algebra is defined over relations by the use of **relational operators**. A relational operator is an operator that takes one or more relations as operands and produces a relation as a result [14, section4.1]. Among the relational operators that constitute a relational algebra are **JOIN**, **restriction**, (σ), **projection** (π) and **MINUS** ($-$). [14] can be consulted for further explanations of these notions.

Predicates in first-order-logic : A predicate can be defined as a the meaning denoted by a sentence in declarative form. That sentence can contain place holders that are meant do designate something in particular. These place holders are called predicate parameters and a predicate that contains n parameters is said to be an *n-adic* predicate [14, section3.2]. When a predicate has no place holders or when its place holders are actually designating something in particular, the predicate is called a *proposition* : a sentence that is either true or false about the domain. Table 2.3 illustrates the previous concepts.

Table 2.1: Different types of sentences

	Sentence	Predicate	Proposition
"Is the sky blue ?"	yes	no	no
"The sky is [*color*]."	yes	yes	no
"The sky is blue."	yes	yes	yes
"Objects have colors."	yes	yes	yes

In the branch of logics called first-order logic, predicates are combined using, **logical op-**

erators - AND ($\vee$), **OR** ($\wedge$) and **NOT**($\neg$) - and **quantifiers - for all**($\forall$) **exists** ($\exists$) - that quantify over single objects of the domain. [14] can be consulted for further explanations of these notions.

Relations and predicates : There is a very narrow relationship between predicates and relations : in fact, a relations represents the extension of some n-adic predicate P by having a body consisting of every n-tuple that makes P be true [14, section4.2]. For example, if $IsAnAnimal(x)$, x belonging to the set $A = \{cat, rock, ant\}$ is a predicate, then the relation $A' = \{cat, ant\}$ represents the predicate $IsAnAnimal(x)$. This has as an important consequence that there exists an equivalence between relational operators and logical operators. For example, section 4.4 of [14] shows how the **JOIN** operator is in fact the relational counterpart of the **AND** logical operator.

Object-oriented design : in the object oriented paradigm, an object is an instance of a class, the latter meaning the description of an entity of the domain that can posses **attributes** and methods. A class attribute can be viewed as the links that are allowed to exist between the objects belonging to this class and the other objects, in a chained list manner. A class method contains the executions steps that can modify the links between the objects. The classes of a domain are **types**, that is, they classify the objects they describe in different sets between which hierarchical dependencies may exist. **Subtypes** thus **inherit** from **supertypes** meaning that the former possess all the attributes of the latter.

According to the previous definition, classes are sets of objects, in other words, unary relations in the same manner as object attributes defined inside classes constitute n-ary relations upon the objects that exist in the domain.

The mathematical background underpinning the Alloy modelling tool being briefly presented, we can proceed to a presentation of the latter tool's features.

2.2 The Alloy modeller and analyser

Alloy views a modelled domain as a constraint – a logical declarative sentence about the domain – over a Univrse of elementary objects called **atoms**. Atoms are involved in tuples and tuples are involved in sets, thus forming relations. An Alloy **model** is therefore a constraint built by the Alloy analyser from the conjunction of the elementary constraints expressed in the *Alloy modelling language* about links between atoms. The former built constraint is then the input of an embedded *SAT solver* that will discover **instances** or **counter-examples**:

- An *instance* is an assignment of the atoms of the Universe to the sets and tuples that are used to represent relations in a way that satisfies the constraint.
- A *counterexample* is a version of the Universe that does not satisfy the conjunction of the main constraint and of an auxiliary constraint that one would like to be true all the time about the model.

The Alloy modeller and analyser are thus the combination of the following technologies : the Alloy **modelling language**, the Alloy **analyser**, embedded **boolean satisfiability (SAT) solvers** and the Alloy **visualization tool**.

The Alloy modelling language integrates first order logic, relational algebra and object-oriented concepts into a language that allows one to express elementary constraints about a modelled domain. At the lowest level of abstraction an Alloy modelled domain is represented by

a finite set of *atoms*, the *Universe*, the former being grouped into smaller sets representing unary relations and involved in sets of tuples representing more-than-unary relations.We present the features of the Alloy modelling language using examples from [3] and from the examples bundled inside the Alloy software. The main features of the Alloy modelling language are:

- *Signatures* : Signatures can have attributes and are a way of representing a relation. A signature is a unary relation, meaning a set of atoms. A signature containing an attribute also represents a more-than-unary relation. A signature can extend another signature in the object-oriented sense of this term, thus inheriting all the attributes of the former. Figure 2.1 shows an example of singatures with (Book) and without (Name, Addr) attributes.

Listing 2.1: Alloy signatures

```
sig Name, Addr { }

sig Book {
   addr: Name -> lone Addr
}
```

- *Facts* : facts are the conjunction of logical sentences about relations that are expected to be evaluated as true and that are automatically part of the model.

Listing 2.2: An Alloy fact

```
fact eating { eats = Fox->Chicken + Chicken->Grain }
```

- *Predicates* : predicates are logical sentences that are not automatically part of the model and that can be evaluated as true or false. They can be used inside other logical sentences and can have arguments. A *command* can be used to *run* a predicate, that is, to look for an instance that satisfies the conjunction of the statements contained inside the predicate.

Listing 2.3: An Alloy predicate and a run command

```
pred ownGrandpa [m: Man] { m in grandpas[m] }

run ownGrandpa for 4 Person expect 1
```

- *Assertions* : assertions are logical sentences that are not part of the model and that can be evaluated as true or false. A *command* can be used to *check* an assertion, that is, to verify that all the model instances satisfy the latter assertion. If a counterexample is found among the model instances, the check is said to have failed.

Listing 2.4: An Alloy assertion and a check command

```
assert NoQuantumObjects {
   no s : State | some x : Object | x in s.near and x in s.far
}
check NoQuantumObjects for 8 State expect 0
```

- *Functions* : functions are constraints upon the different sets of the universe that use relational and set operators and aim to restrict the universe to a particular subset of atoms or tuples. They can be used inside other logical sentences in which they represent the set of atoms or tuples they restrict the universe to and can also have arguments.

Listing 2.5: An Alloy function

```
fun lookup [b: Book, n: Name] : set Addr {
   n.(b.addr)
}
```

The model in itself is made of signatures, facts, predicates and functions, while assertions and predicates are used for analysis purposes.

An important concept about Alloy modelling is the notion of **verification scope**. The *run* and *check* command will look for instances and counter-examples only for a given number of signatures specified by the user. Listings 2.3 and 2.4 contain examples of such scope definitions. The keyword *expect* is used for testing purposes and specifies the predicted outcome of the command execution.

Manipulating Alloy atoms results in describing relations at tuple level. In order to facilitate the making of specifications about tuples, Alloy allows the use of **cardinality operators**. The operators are **lone** (at most one), **one** (exactly one), **some** (at least one) and **set**(any number). They can be used to describe the tuples of a relation anywhere a relation is represented. For example in the statement of listing 2.2 from section 6.2 of [3], the lone operator used to describe the tuples inside the relation from Process atoms to Time atoms constrains the *elected* relation inside all its tuples to link at most one Process atom to a given Time atom. The translation to natural language is that only one process can be elected at a time.

```
elected in Process lone -> Time
```

Following this example, the relation {(Process0, Time0),(Process1, Time1),(Process1, Time0)} would be a invalid value for the elected relation. More examples of the use of cardinality operators to describe relations are contained in listings 2.1, 2.5.

The Alloy modelling language also provides for the use of the first-order-logic operators and relational operators as shown in figure 2.1

First-order logic	Alloy
negation	not, !
conjunction	and &&
disjunction	or, ||
implication	=>
Bi-implication	<=>

(a) Logical operators

Relational algebra	Alloy
arrow(product)	$\rightarrow$
dot (join)	.
box (join)	[]
transpose	$\sim$
transitive closure	$\hat{}$
reflexive-transitive closure	*
domain restriction	$<:$
range restriction	$:>$
override	++

(b) Relational operators

Figure 2.1: operators correspondence in the Alloy modelling language

The Alloy analyser parses and translates the elementary constraints expressed in the Alloy modelling language into a valid input to be passed to a SAT solver. It is in this regard more of a compiler that has the otherwise essential role of performing important optimizations , one of which is for example *symetry breaking*.

Embedded SAT solvers are leveraged by Alloy to discover one after the other some instances of the model. All the constraints that constitute the latter are transformed for this purpose into

a unique boolean formula with boolean variables. The boolean satisfiability solver is thus blind to the model an only views it as a large formula containing variables it tries to assign a value to in order to make it evaluate to true. An assignment that makes the formula true is transformed by the Alloy analyser into an Alloy instance in which each relation is no longer a variable but has a precise value.[3, p. 153] gives a succinct explanation of the basic principles behind theses transformations. The fact that the SAT solver does not directly look at the specification of a model but at a logical propositional formula derived from it, is what shifts the instance finding criteria from business-based factors to purely logic-based ones. This helps the early discovery of the most singular cases, hence significantly improving process optimization. Multiple solvers are available inside the Alloy option menu and differ for instance in their speed or in the type of environment they are best adapted to.

The Alloy visualization tool is a powerful feature of Alloy that not only allows the modeller to have a visual representation of the domain objects, but moreover to use **projection** over the different relations in order to have access to different view points of the same instance.
A projection consists in splitting an n-ary relation into groups of less-than-n-ary realtions according to some chosen positions in the tuples - that correspond to sets of atoms - that compose it. It is then easier to analyse the relation looking at each yielded sub-group of relations. Considering the sets $A = \{a_0, a_1, a_2\}$ and $B = \{b_0, b_1, b_2\}$ and one binary relation $R = \{(a_0, b_1), (a_0, b_2), (a_2, b_2)\}$ the projection over tuple position 1 (or set A) would yield the unary relations $\{(b_1)\}$ and $\{(b_2)\}$. The Alloy visualizer also allows one to easily attach visual characteristics to objects and relations such as colours and shapes enabling the creation of lively simulations. Textual and tree-structured representations are available as well.

2.3 The Robert's Rules Of Order Newly Revised

2.3.1 Elements of history

The Robert's Rules Of Order are named after the General Henry Martyn Robert (1837-1923). In the years 1863 to 1875, his concern was to produce a collection of rules that would harmonize the definition of *parliamentary law* in the assemblies that he had attended in various states of the United States of America where different parliamentary practices were applied, therefore creating a hindering multiplicity of rules and procedures. Since 1912, the original document has undergone multiple revisions all steered by the feedback sent by users experiencing undocumented, contradictory or ambiguous situations [1, p. xl-l]. This tradition is today perpetuated through the RONR Official Interpretations[5].
The basis of the revision process has therefore in some way always been about (1) writing the rules, (2) producing enough instances of their application to discover problematic situations, and(3) logging these situations in order to (4) feedback a new analysis and edition of the rules. In the year 1990,[1, p. xlvii] mentions that "new technology" applied to the latest edition of the book made for the first time the revision become a much more practicable process.
Our attempt is to continue taking further steps in the use of suited technology for the analysis of the Rules Of Order notably using the Alloy tool that integrates in a complete and centralized manner the steps of the previously described revision process while taking advantage of the latest advances in the field of instance finding.

2.3.2 Overview

The activity of a deliberative assembly consists in the settlement of a series of matters addressed as *questions*. The settlement of the questions put before the assembly generally undergoes a voting process in which each member can influence the opinion of others during a debate and give weight to his own opinion by expressing a vote.

The RONR therefore describes a unified body of procedures, however, theses procedures are valid for different types of deliberative assemblies. This corresponds to the context that was at the origin of the creation of the rules : the unification of American parliamentary proceedings. The differences between the types of assemblies are mostly about the number of participants, their degree of expertise about the business at hold, the regularity of the gatherings, the degree to which formalisms constraint the proceedings, and the degree of autonomy of the assembly relatively to another assembly.

Mass Meetings can have any number of participants with low expertise required of them. **Local assemblies** of an organised society should hold regular meetings and participants have a bit more expertise. **Conventions** are led under more formal constraints and gather delegates with high expertise of the business domain. **Legislative Bodies** are similar to conventions but the former assemblies also exist and function generally in pairs of assemblies with a high degree of formalisms within the proceedings. **Boards** at last have very few members and are generally under the authority of an other assembly.

Deliberative assemblies hold **meetings** that are physical gatherings and **sessions** that can be defined as logical units of proceedings with respect to the business to be dealt with. Both generally overlap.

The formal way in which business is dealt with by an assembly is through the making and processing of **motions**, though under some circumstances business can also be settled without any motion being made. A motion is defined as *"a formal proposal by a member, in a meeting that the assembly take certain action"* [1, p. 27]. The motions generally belong to classes according to the way in which they affect the settlement of business. The RONR identifies 86 different motions that can be **Main**, **Subsidiary**, **Incidental**, **Privileged** or **Bringing a Question again Before the assembly**. Subsidiary, incidental and privileged motions are **Secondary Motions** that are made in order to help deal with the main motions. The last class contains main motions that undergo a special process in order for the assembly to be able to reconsider them.

Inside the subsidiary and privileged motions additional relations are defined among the motions that constrain the order in which the latter are allowed to appear. A **precedence ranking** is therefore defined.

Despite the fact that motions are classified and ranked there still exists a high diversity within the classes of motions. Some motions affect the proceeding of the current session or meeting, some others affect only the content of other motions, others also affect how this motion will be processed or which other motion will further be allowed.

When an event occurs during the settlement of business that is not allowed by the rules or by the abstract goals of the deliberative assembly, this event is said to be *out of order*. This brings us to another characteristic of the the Rules Of Order: in order to allow motions to fulfil their function while avoiding being out of order, the rules are assorted with a number of exceptions of various types. For example some motion may not have classes, some may not be allowed according to the assembly type or some may be able to depart from the precedence ranking.

The former and latter facts brings to the forefront the fact that a motion's effect on the parliamentary process is not self-contained. It is dependant on the presence or absence of other motions and the relations between them.

2.3.3 The Robert's Rules of Order as a business process

Coming back to the notion of business process, the definitions that we have given of them point out that processes have different characteristics. One of these implies the way in which its sub- processes interact. For instance, The overall process may be completely sequential or made of concurrent sub-processes, sometimes to the extent of making them closer to be defined as process architectures [7]. In the case of the Rules Of Order, the ambition is to achieve perfect sequentiality and perfect concurrency. The perfect sequentiality ideal is aimed at through the

collection of rules that constitute the Rules Of Order that define what sequence of action is legal during a gathering. Sequentiality is the characteristic that is thus the most formally enforced. The perfect concurrency ideal is aimed at through the quality of the moderation of the gathering by the persons who endorse this role. The concurrent situations are notably the way in which members *access the floor* in order to express their opinion or introduce a question to the assembly and the *formation of an opinion* by each individual member expressed later on by a vote.

Another characteristic is the achievement of a goal or a set of goals or business objectives. The goals of the RONR is to capture and express the collective will of the assembly. To achieve this, the Rules Of Order give rights to and protect the rights of **the majority**, **the strong minorities**, **the individual members**, **the absentees** and **the overall group**.

The two previous characteristics appear in the description of the **fundamental principles of parliamentary law**. The principle therefore stipulate that:

- **Only one question can be considered at a time** [1, p. 59]
- **One person, one vote** [1, p. 407] :
- **A two-third vote is needed to suppress debate** [1, p. 216] :
- **Vote is limited to members present** [1, p. 423] :

To these principles one can join the fundamental rights of which individual members cannot be deprived :

- **the right to attend meetings**
- **the right to make motions**
- **the right to speak in debate**
- **the right to vote**

The consideration of principles inherent to law in its general sense, like for example protection against instability are also part of the concerns of the Rules Of Order. The application of these principles will be modulated by the various factors such as the types and importance of the questions, the roles of the members, the elapsed events inside the process etc.

2.4 Problem identification

Given a collection of statements about parliamentary law proceedings, we are faced to the problem of its representation inside a paradigm based on pure relational algebra using the Alloy modelling language and the representation of abstract goals into formal properties. These properties should bear high chances of allowing us to infer new, implicit or contradicting facts. This general problem statement can be divided into sub-problems to be addressed and solved.

A first sub-problem concerns the combined facts that the domain is large and complex: the RONR describes the gatherings of deliberative assemblies in order to take care of business. These assemblies can be of different types. For example mass meetings gather a high number of members in a highly informal manner whereas boards gather a small number of people with special attributions and for very specific purposes. The questions can be treated undertaking different types of actions according to the importance of the questions, the degree to which they could oppose the assembly members and other factors. Formal motions demanding conditions to be met are made to deal with businesses but under some circumstances business can also be settled without any motion being made and no voting required. Voting processes, when required,

are of different types, use different notions of majority. Discontinuity of gatherings may divide the ongoing processes into meetings in which the conditions could be each time different.

In brief, the conjunction of many factors cause deliberative assemblies to have many characteristics. The latter have to be sorted according to some criteria when identifying a precise self-contained subset of concepts to model. If we moreover consider that the entities and actions mentioned by the previous descriptions have intricate intercourses between each other because definitions of concepts and actions are not self-contained, sorting there characteristics implies the non-trival task of wielding abstraction interestingly enough to allow us to select dependencies and represent them under the form of relations between self contained entities. To do so, one must define a reasonable and incremental modelling scope while being careful not to simplify it excessively but also bearing in mind the limits of the modelling tool.

A second sub-problem concerns the fact that once a scope is defined, we must study the form in which are formulated the rules in order to efficiently and correctly map their meaning into Alloy facts, predicates and assertions. This involves resolving how the chosen natural-language sentences translate accurately to first-order logic and defining the modalities of this translation such as for example how many constraints should map to a single sentence. Because of the diversity of the rules, these are questions that have to be dealt with on a per case basis.

Then, provided a model, a third sub-problem is introduced when we need to make it say something of interest about the domain. Regarding analysis, we should understand which kind of properties we are able to verify upon our model. This will depend on the modelling language capabilities. We should then ask ourselves the question of what property is one that we would like to verify in the context of checking consistency of the RONR. *What should always be a general rule about any instance? What could be seen as contradictory to the assembly's goal?* This will be possible by formalizing the RONR fundamental principles into logical assertions to be checked against the model. But we should also give ourselves the means to observe behaviours or interactions that we had not foreseen.

As an answer to the modelling and analysis questions, we suggest three different sub-models encapsulating different types of concerns. For an analogy using the Unified Modelling Language, the nature of these models combines those of a state diagrams, an entity-relation diagram and a class diagram. The choice of the concerns addressed by each model have been made as much as possible dependant on the characteristics of the domain, but are also dependant on the limits of the Alloy modelling tool and reasonable time and space computing capabilities. To theses sub-models we join a series of testing predicates and assertions that address implementation concerns on one hand, and verification concerns on the other.

Chapter 3

Modelling phase

3.1 Definition of a modelling scope

This section discusses why the definition of a scope matters in context of our problem and the factors that have to be taken in consideration in order to define a scope. It then presents the assumptions made about the domain in order to define this scope.

3.1.1 Model size

Alloy is a model finder. This means that it actively searches a space of values in order to find values that satisfy the model's constraints. In Alloy the variables to be assigned values are relations. A discussion in chapter 5 of the Alloy reference book [3, p. 154] explains the process by which relations are transformed into variables. The size of an Alloy model is therefore dependant on one hand of the number of atoms in each existing signature (which will influence the size of the relational variables) defined by the user-specified scopes, and on the other hand of the number of relational variables defined by the quantity of concepts being modelled. There is a third factor that can influence the size of the model which is the meaning of the relational constraints, but this factor is somehow random since it is entirely dependant on the intrinsic characteristics of the modelled domain. However, given this definition of the a model's size, a model explosion only happens with respect to a defined resources limit in terms of computation memory or computation time. Typically, one would like to limit as much as possible the time spent looking for an instance.

3.1.2 Simplifying hypothesis

Two concerns led the definition of the modelling scope : keeping a reasonable number of constraints contained inside the model and representing instances that were realistic enough in order to be able to draw interesting enough conclusions from the analysis. We call the simplifying hypothesis the assumptions that we make over the domain and that restrict its complexity.

Defining simplifying hypothesis at high level only is good but not sufficient. We therefore have augmented our flexibility in the carving of the models' scope by pushing a part of these hypothesis down to the elementary constraint level. In the model files contained in the appendix the latter are represented by comments marked with a double star. In this document we present the most important high level modelling assumptions.

Motions as first class citizens *"Business is brought before an assembly by the motion of a member"* (*§3 Means by which business is brought before the assembly*): even though other more informal ways are allowed to be used and since the RONR presents the motion of members as the proper and main way to introduce business before the assembly, the models presented are

focused on motions : the models represent a world in which only motions can bring business before the assembly (and remove it) and focus on the way these motions influence each other.

The modelled concepts We have operated the choice of modelling proceedings as they would happen inside Legislative Bodies. Alloy instances are whole or partial sessions of this type of assemblies that we assumed would be complex enough to conduct our study and would relieve ourselves of a part of the exceptions that come along with the rules. After a study of the different possibilities, the retained concepts to be modelled inside a session were the following :

- Session states : all three models represent the states of a session. A session is here defined as a gathering or a series of gatherings in order to deal with the same set of businesses. A session can have a specifically described initial state and final state.
- Businesses : the purpose of a session is to settle all businesses belonging to it by the means of one or more motions. We call the sequence of businesses to be dealt with the order of business.
- Motions : as official statements of a decision adopted by the assembly. Motions have types and can influence each other.
- The actions regarding motions : all the legal actions from one state to another state are related to the creation modification or disposal of one or several motions.

Motions have been classified by the RONR into families. Some motions directly deal with businesses and others, secondary motions, help with the settlement of the former or with the good proceeding of the meeting. Figure 3.1 shows the hierarchical relationships between the classes of modelled motions generated using the visualizer. The graph generated by the visualizer is a **metamodel** of the domain. A metamodel showing the whole view of concepts put together is available in the annexe.

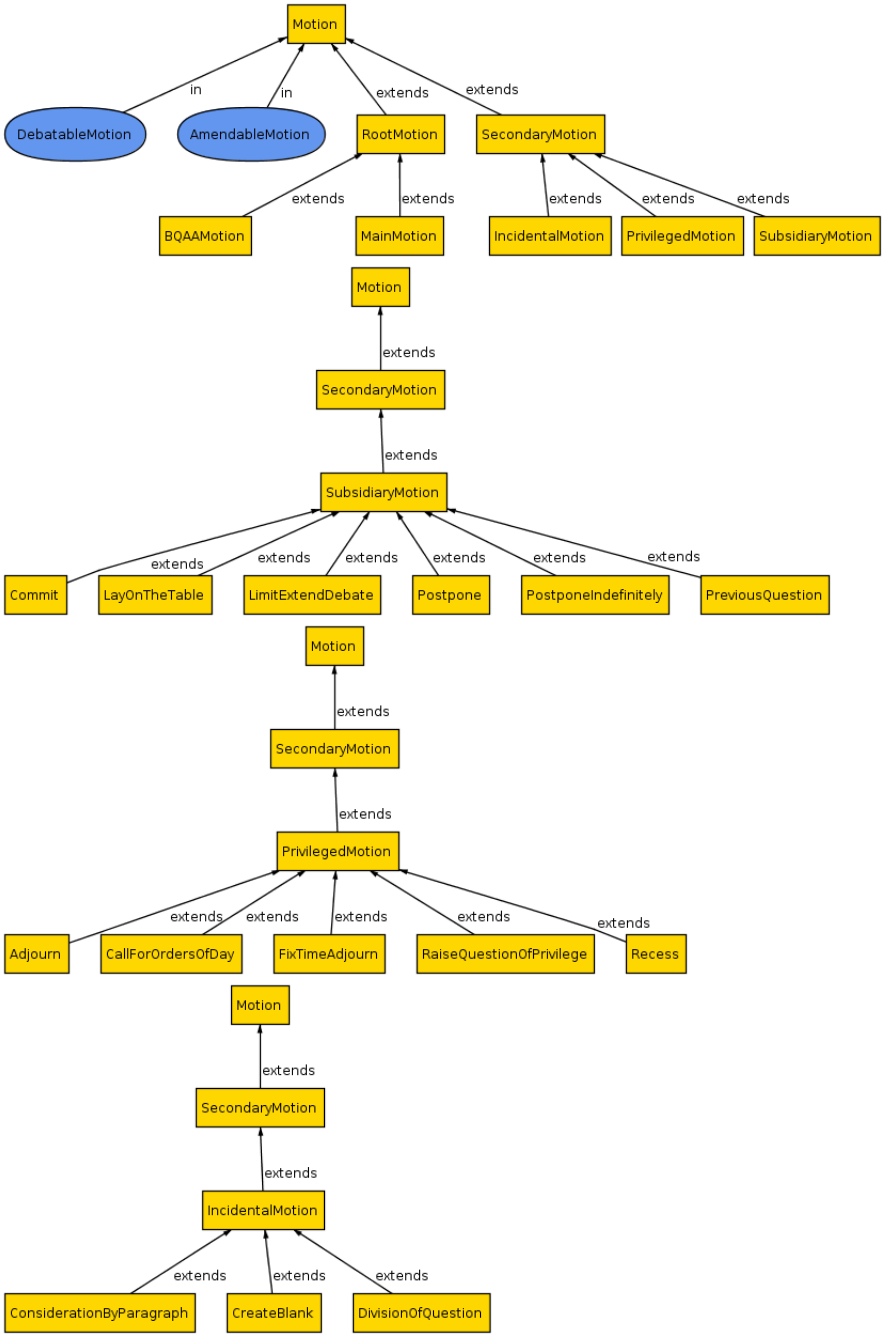


Figure 3.1: Partial metamodels of the motions

The subsidiary and privileged motions are fully modelled. This is because of the fact that the dependencies between them constitute a coherent bloc that we have considered useful to conserve inside one model. The content of this coherent block of dependencies will be translated into relations at modelling phase.

The concepts that are not modelled

- Members : motions have one or more members of the assembly as authors and members express their vote about motions. The models do not represent the rules according to the authoring of motions by members and assume that the existence of a motion implies that these rules have been respected.
- The categories within the order of business : the full description of the order of businesses

within a session is in fact an order of categories in which all businesses belonging to this category should be addressed. As a simplification we have not represented these categories and have given a fixed order to businesses.

- The coherency of the content of motions : the modelling does not take into account what the motions concretely formulates. For example the existence of a motion will not be forbidden because its content is in its verbal meaning in contradiction with the content of another motion. Only the characteristics of motion types are considered. Such motions would be, in a meeting that is correctly directed, ruled as being *Out of order* and rejected by the director of the meeting.
- verbal forms : legal verbal forms required from the members to initiate an action are assumed to be guaranteed.

3.1.3 Workflow

After simplification, the view upon the activities of a deliberative assembly that we have retained is represented in figure 3.2.

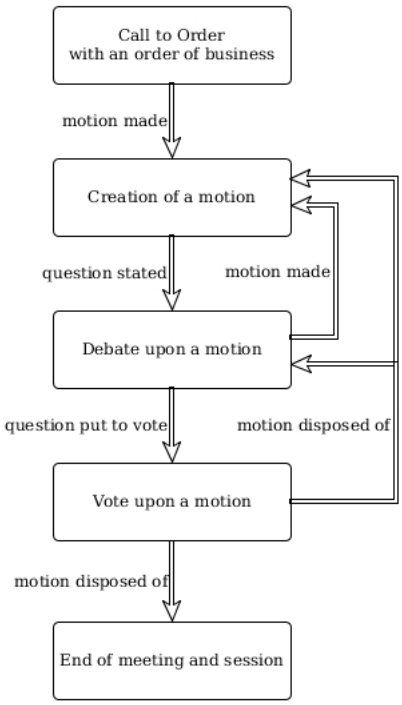


Figure 3.2: High level workflow

The workflow considered represents one session happening within a single meeting and contains five high level activities, each of which is used to encapsulate the complexity of sub-activities :

Call to order contains proceedings related to the initialization of the meeting and the session such as the verification of the quorums, the approval of minutes and the presentation and approval of the order of business.
Creation of a motion contains all the proceedings that allow a motion to be officially taken into consideration by the assembly. These include the access to the floor, the seconding of the motion and its possible reformulations.
Debate upon the motion contains the proceedings that allow members to express their opinion and make their minds about the current motion and includes the access to the

floor and the making of new motions. When a new motion is made while another one is being processed, it is a secondary motion.
Vote upon the motion contains all the proceedings related to the vote, ranging from the approval of the formulation of the decision to be taken, to the verification procedures.
End of meeting and session contains the proceedings related to termination.

After a modelling scope has been laid out, we can now describe how the elements it contains are structured inside Alloy models.

3.2 Modelling design

In this part, we walk the reader through the two sub-models of the RONR that we have realised and that span the previously defined scope. The first part looks at the sub-models in terms of what they have in common presenting their general structure. Then we go into the details of the sub-model 1 and sub-model 2. Sub-model 1 is a model that focusses on navigation and sub-model 2 takes an interest in a particular family of motions.

3.2.1 Structure of the designed models

In this part we explain the global design of our model. We lay out the captured aspects and explain how we have mapped RONR concepts into relations.

Mapping domain concepts to Alloy relations

The constraints listed inside the RONR link the concepts in different ways. In order to yield a correct representation of the rules we needed to proceed to this mapping while strictly following the editorial structure of the RONR. Two groups of dependencies were therefore identified.

The first group contains the following links between the concepts :

- **The classes of motions** : these are dependencies that we have regarded as high-level because they are the base classification of motions. We have used unary relations, that is, sets to represent these rules.
- **The low level rules** : they contain three types of rules : the **application rules**, the **precedence rules** and the **precedence ranking**. An application rule is defined between two motions when one targets the other in order to modify its processing is some way. A precedence rule is one that expresses the chronological appearing of the current motions with respect to each other at a given moment of a session. A motion that appeared after another takes precedence over the latter. The precedence ranking is an ordering system that describes in which order motions have the right to take precedence over each other. It gives a rank number to each motion and motions of lower rank can generally not take precedence over motions of higher rank.
- **The exceptions**: the expression of a derogation to a rule creates a dependency between all the concepts involved.

We have looked at those links in terms of plain dependencies and have translated these dependencies to relations. listing 3.1 shows an example of such a translation.

Listing 3.1: Translation of dependencies to Alloy relations

```
/***
* A Secondary Motion
* §5 : Taking of precedence by one motion over another
*/
```

```
abstract sig SecondaryMotion extends Motion{
   takesOver : lone Motion,
   isAppliedTo : set Motion
}
...
fact {
   //natural ranking priorities over other privileged motions
   no (CallForOrdersOfDay.^(takesOver) & PrivilegedMotion)
}
...
fact{...
   //SubsidiaryMotions but Amend do not take precedence over PriviledgedMotions
   //§6: Privileged motions; §16: Standard descriptive characteristics 1. ;§12:
       Standard descriptive characteristics 1. b)
   no ((SubsidiaryMotion-(Amend+PreviousQuestion)).^(takesOver) & PrivilegedMotion)
}
```

In listing 3.1 SecondaryMotion is a signature, that is , a set of Alloy atoms and a unary relation over these atoms. This signature is a subset of the signature Motion, hence the extends keyword. The attributes of the SecondaryMotion signature, takesOver and isAppliedTo, denote the existence of a binary relation between atoms of the SecondaryMotion set and those of the Motion set. The keywords lone and set express the cardinality of these relations : one SecondaryMotion atom can takeOver at most one Motion atom and can be applied to any number of Motion atoms. A rule of precedence ranking is expressed inside a fact bloc and expresses that the motion CallForOrdersOfDay has the lowest ranking priority among the privileged motions. The last fact block expresses a rule of precedence ranking that has an exception to it regarding the motions Amend and PreviousQuestion.

The second group of dependencies identified contains the links that are created between the domain's concepts due to an action. These are actions that are executed along the workflow presented in the previous section and that are centred on the life cycle of individual motions. The goal is to express the advancement of a session in terms of the lifecycles of its motions. The actions are :

- **The making of a motion** : a motion comes to existence inside the session.
- **The adoption of a motion** : a motion is accepted by a favourable vote of the assembly.
- **The loss of a motion** : a motion is rejected by the a unfavourable vote of the assembly. Note that this action can be implicitly executed when the previous one is not.
- **The disposal of a motion** : a motion is taken out of the current focus of the assembly.
- **The processing of businesses** : which is done in a certain order.

Each of these actions have initial conditions to be met in order to be allowed to happen and must have their post-conditions enforced by the description of the state of the ongoing session after they have occurred. We have represented these dependencies through couples of pre and post-condition constraints encapsulated inside action predicates as shown in the example in listing 3.2.

Listing 3.2: Translation of dependencies to Alloy predicates

```
/***
* A motion is moved by a member who has
* obtained the floor for it to become a pending question.
*/
pred move[sBefore, sAfter:SessionState, motion : Motion]{
   move_preconditions[sBefore, motion]
```

```
    move_postconditions[sBefore, sAfter, motion]
}
```

One can draw the common characteristics of the dependencies inside these two groups with respect to the way they interact with the session states. In this respect the dependencies of the first group can be regarded as "static" and are represented by n-ary relations that are not tied to session states whereas the dependencies of the second group can be considered "dynamic" and involve session states in ternary relations. We however draw the attention of the reader on the fact that the "static" and "dynamic" qualifiers that we apply here to the dependencies between the domain concepts are with respect to the manner in which the RONR lays them out textually and not to their moment of apparition during the session. The moment of apparition of the relations during a session is the moment specified inside the simplified workflow : motions really come to existence only at the moment they are made.

These groups also define the granularity of the elementary constraints of our model: a single elementary constraint is equivalent to one dependency. This may seem a fastidious task but is in fact a careful way of proceeding in order to avoid premature interpretation of the meaning of the rules that could lead to misinterpretations or translation mistakes from natural-language to first-order logic.

The paragraphs here above explain how domain concepts are translated into Alloy constraints inside the sub-models, but apart from the pure domain concepts there are utility concepts that have been added to the sub-models in order to add flexibility and allow the later analysis of the sub-models. Inside an Alloy instance, these utility concepts shall also be represented by atoms, relations and option predicates.

From dependencies to abstract state machines

Following the identification of static and dynamic dependencies, the building blocks from the previous section have been put together in order to simulate state machines executing upon a dependency graph including these building blocks. The dependency graph that we have designed is composed of three functionalities. First, it is a static dependency generator which generates instances that satisfy the static dependencies previously identified inside the domain independently of any session. Secondly, it represents the identified dynamic dependencies by generating inside each Alloy instance one sequence of session states through which the generated static configuration can or cannot be reached in a step by step manner. The second functionality is thus one of a state machine representing a single session. Finally, it is also a compound state machine representing the concurrent execution of multiple sub-machines. Each sub-machine representing the life of an atom (other than a session state atom). We now explain these tree functionalities in more concrete terms.

Static dependancy generation Firstly, starting from the description of the static dependencies, we use Alloy to generate inside each instance it finds a set of interrelated atoms corresponding to domain concepts, the configuration of these connections being one that could potentially appear during the proceeding of a session. Figure 3.3 shows an example inside the visualizer of such generated atoms.

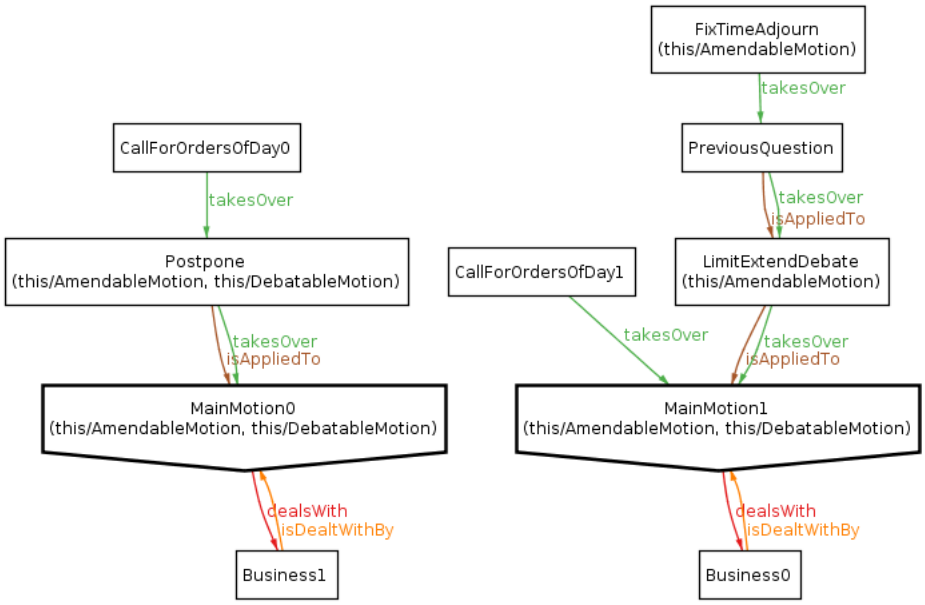


Figure 3.3: A representation of static dependencies inside an Alloy instance

The figure shows ten atoms. Each atom is named after the set it belongs to with a number appended to it as a differentiator. An arrow between two atoms represents a tuple. In this example, as all the relations are binary, the tuples contain two positions. Each arrow is labelled with the name of the relation it belongs to.

Session state machine The second functionality allows us to observe and reason upon the execution traces of an abstract state machine that has the dynamic dependencies between session states and motions as its transitions. The second functionality uses the first one : the dynamic states execute on top of the configuration given by the static dependency generation. Figure 3.4 shows an example of all the atoms and relations generated by the Alloy analyser inside a single instance with the two functionalities being overlapped.

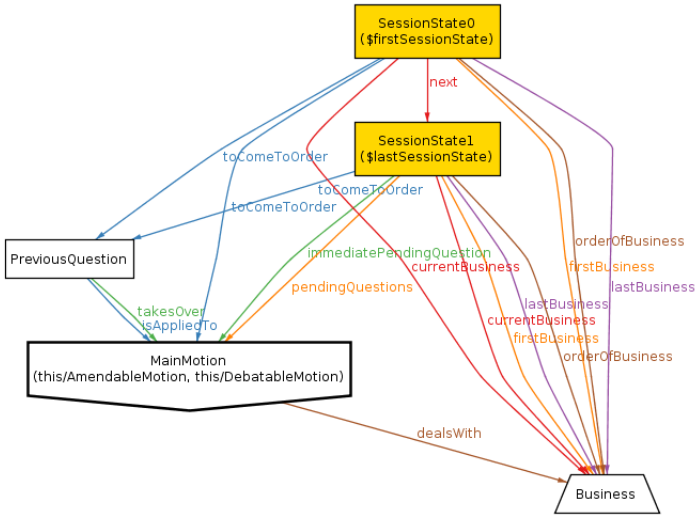


Figure 3.4: A representation of static and dynamic dependencies inside an Alloy instance

The correspondence between the state machine concepts and the Alloy concepts are the following :

- Some **state** of a session is represented by all the atoms that are reached when projecting

the universe upon the atom that represents this SessionState along with the tuples the former atoms are involved in. For example in figure 3.5 obtained by projecting the instance of figure 3.4 on the SessionState signature, SessionState1 is represented by the reachability of the atoms PreviousQuestion, MainMotion and Business through the relations toCome-ToOrder, firstBusiness, currentBusiness, lastBusiness and OrderOfBusiness projected on SessionState1. The relations binding SessionState atoms to other atoms are thus used to cluster the reachable atoms inside state sets. One of such relations will cluster atoms in different ways for each SessionState.

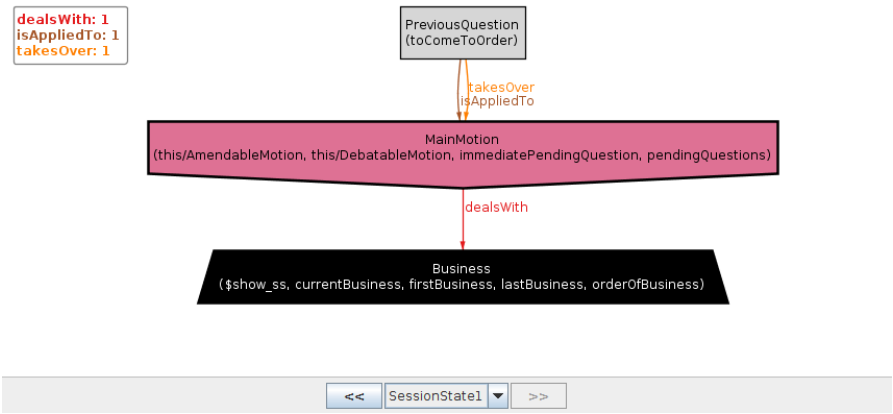


Figure 3.5: Projection of the instance in figure 3.4 upon the SessionState signature

- The **invariant** of a given state is therefore the composition of its state sets. For example, from one SessionState to another if no reachable atom has entered a state set nor exited a state set, this means that the two session states are the same. Because each motion contained inside a set is unique, Alloy makes it possible to capture complex state descriptions in a relatively simple way.
  The main state sets comprised in each session state are : the motions yet to be made (*toComeToOrder*), the motions waiting to be dealt with (*pendingQuestions*), the motion that is the focus of the assembly (*immediatePendingQuestion*), the motions undergoing the effect of some other motion (*under[*]Order*), the navigation pointers between businesses (*nextBusiness and specialOrder*) and the first, current, and, last businesses(*firstBusiness, currentBusiness, lastBusiness*). The navigation pointers sets contain elements that are couples of businesses representing the knowledge of the state about the order in which businesses should be processed.
  As explained in [12], a sate invariant can be represented either by a stable situation or by a state activity. However, within the representation of a state inside the visualizer, no default distinction is made in order to distinguish the invariants of the state that are of either of these types. The software engineer uses the options available inside the Alloy visualizer in order to represent the different invariants that hold in a session state with the responsibility of defining a clear meaning of these representations for the user's interest. In figure 3.5 for example, the main motion being part of the red set means that debate is actually proceeding about the former motion in that session state and the business being part of the black set means that the business is the one currently being settled. The motion PreviousQuestion being part of the gray set rather represents the static situation of this motion not being yet called to existence.
- A **transition** between two session states is represented by an action predicate that evaluates to true. The predicate is itself composed of two groups of constraints representing the conditions to be met in the pre-state and the conditions that are true in the post-state. In our design the name of the action that is executed in order to transition from one state to another is not visible inside the visualizer. However, the knowledge of the meaning of the

customisable representations of the state concepts allows the software engineer to notice the changes that have occurred looking at the generated sequence of states. Figure 3.6 shows two consecutive states from the instance in figure 3.4 between which a move action has occurred.

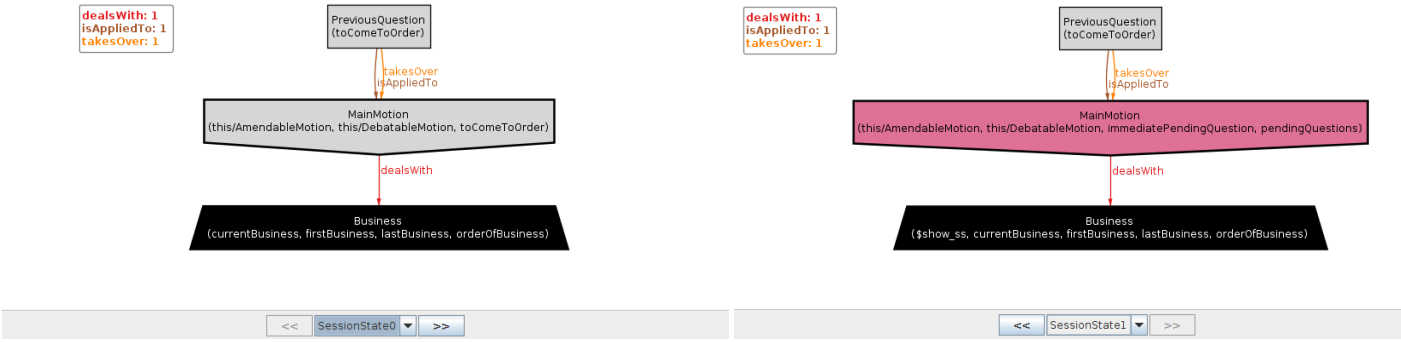


Figure 3.6: States before and after a transition

Concurrent execution of state machines The last functionality allows us to observe the sequence of states in which each atom representing a domain concept goes through, all of these sequences being constrained by the transitions that happen inside the execution trace of the session state machine. This makes the final design of the depeandancy graph be equivalent to N state machines executing concurrently inside the same environment represented by a session and sharing the same pool of transitions. The representations of the execution traces can be viewed by taking different perspectives inside the Alloy visualizer using projection upon each signature of the the Universe. Figure 3.7 shows the result of projection upon the Motion signature of the instance shown in figure 3.4.

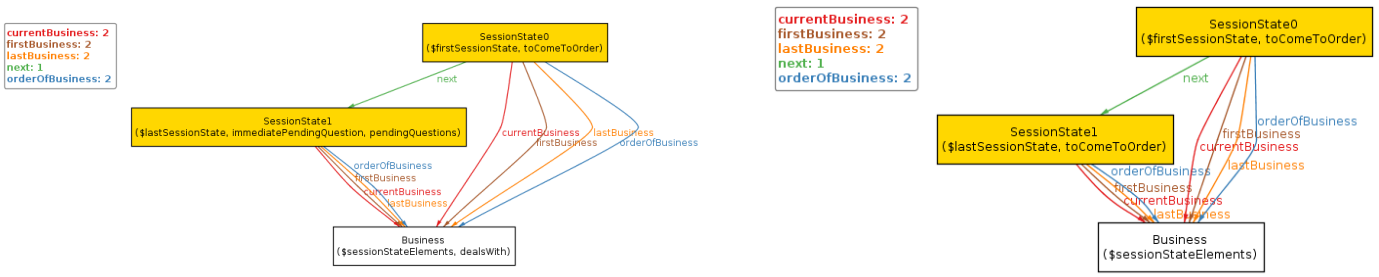


Figure 3.7: Sequences of states of a MainMotion atom and a PreviousQuestion atom.

We can observe that when the session is in SessionState0, the MainMotion atom is in state {toComeToOrder}. It then goes in state {immediatePendingQuestion, pendingQuestion} when the session goes to SessionState1 through a move transition whose pre-conditions it is compliant to. The PreviousQuestion atom doesn't change its state when the session goes from SessionState0 to SessionState1 since it is not compliant with the move transition pre-conditions. More generally,

- A **state** of an atom is represented by its belonging to the state sets.
- A state **invariant** is therefore given by the set of states sets an atom belongs to in a given session state.
- State **transitions** are the same as the session state machine transitions and demand the same conditions to be met.

In our model a certain number of choices are left by default to be arbitrarily made by the analyser. This is the implicit way in which concurrency is modelled in Alloy. If a scope is set for

each top level signature and no additional constraints are joined to the model the choices made by Alloy will be upon :

- The number of states
- The number of businesses
- The number of motions
- The nature of the motions made
- The navigation itinerary
- Which motions are adopted

Specifically, the race condition over of which motion gets to be moved first at a given moment of a session is modelled by the arbitrary choice of Alloy over which action whose transition conditions will be true between two states of the execution traces.

After the presentation of the key features designed during the modelling phase an individual presentation shall be done of the two sub-models that this phase yielded.

3.2.2 Sub-model 1 : focus on navigation between businesses

One of the goals of the deliberative assembly is to deal sequentially with all the elements of business that it has inside its order of business. Elements of business become therefore the focus of the assembly one a after the other. The order in which business is dealt with can be different according to the actions that are made by members during the session. the first sub-model thus focuses on navigation between the businesses and the motions that deal with them.

Motions can affect a session in different ways : they can modify the content of another motion, modify the proceeding of the session state activities or influence the session's overall workflow. Navigation is not influenced by the two first effects since they are not part of the modelling scope we have set previously. The last effect is one of the manners in which motions can have an impact on navigation : the influence of the motions on the session's overall workflow defines *what navigation can do*. The other manner in which motions have an impact on navigation is by restricting it with respect to the precedence rules. The latter define *what navigation cannot do*.

The classification of the terminal motions with respect to the way they affect navigation yields the table 3.1.

Table 3.1: Effect of terminal motions on navigation in model 1

Through precedence rank only	**Through precedence rank and the ability to modify the workflow**
Postpone Indefinitely	Commit
Amend	Postpone
Limit or Extend Limits of Debate	Lay on the Table
The Previous Question	Call for the Orders of the Day
Recess	Raise a Question of Privilege
Adjourn	
Fix the Time to which to Adjourn	

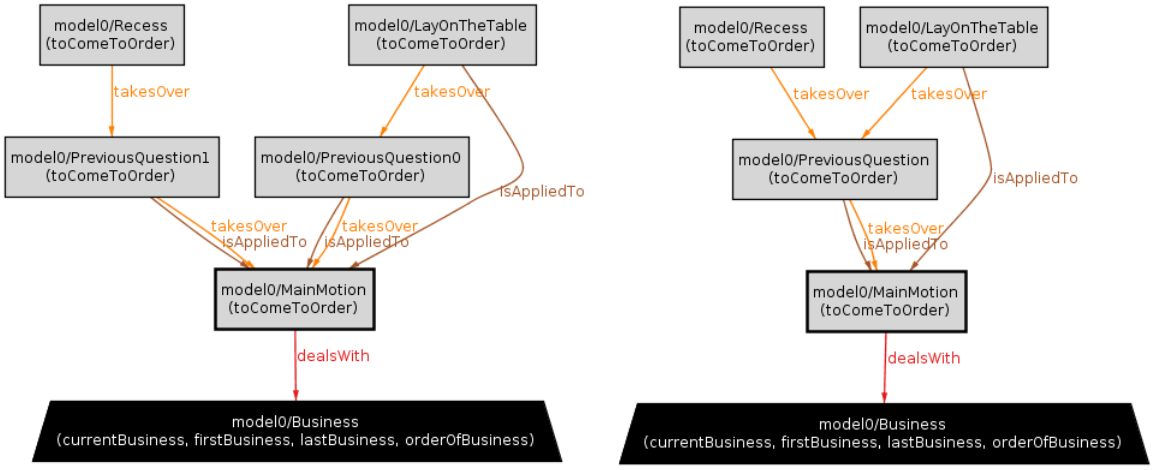
Modelling choices

In order to capture the navigational concerns during a session and to avoid polluting the model with superfluous constraints the following abstractions regarding the domain concepts have been applied:

- *Members and authoring of motions are not represented*: the rules regarding which member has the right to make which motions are considered as respected by default.
- *Motions are not seconded* : some motions in order to officially come to existence need to be supported by a second member different from the motion's author. In this model we consider that the existence of a motion implies that it has been seconded if required.
- *Votes are abstracted in only their final outcome* : the voting process is not represented. Only the fact that a motion has been adopted or lost is visible.
- *Terminal subsidiary motions are instantiated* : the seven subsidiary motions are represented as part of the elements that are navigated between and that influence navigation.
- *Terminal privileged motions are instantiated* : the five privileged motions are represented as part of the elements that are navigated between and that influence navigation.
- *Terminal Incidentals are not instantiated* : the modelling scope does not include defined incidental motions.
- *The effect of adoption of motions on the content of the motions they are applied to is abstracted* : for example we do not represent the effect of an amendment since it is not relevant for navigation.
- *The effect of motions on the status of the motions they are applied to is represented*
- *Partial representation of the constraints upon the motions to Amend* : the motion to Amend appears as having special characteristics that will be addressed by sub-model 2.
- *Every step of the execution of a session modifies the state of the session* : this modelling choice implied the banning of the use of a *doNothing* action that was part of the initial model.

The following utility concepts have been added to the model :

- *An error flag* : it will be used in the purpose of signalling that an error has occurred and appears under the form of an extra atom in the Alloy visualizer.
- *An error action* : its purpose is to allow navigation errors to occur and be observable in the visualizer.
- *The enable_movedOnce option* : an interesting question led us to write a predicate enabling an abstraction over the model about the number of times that a particular type of terminal motion could be moved on top of the same motion. This abstraction made us ask ourselves the question about what made a motion be considered "the same". In this sub-model we say that a motion is the same one as previously when the motion takes precedence over or deals with the same motions or business as previously and is applied to the same motions and has the same class name. For example in figure 3.8a PreviousQuestion0 and PreviousQuestion1 are both applied to the same MainMotion atom and both take precedence over it. In our models a motion is considered to have tree different types of effect on a session : either it modifies only the content of the motions, or it modifies only the circulation between the motions of the session, or it modifies neither of those two. When the movedOnce abstraction is enabled through the use of the movedOnce predicate, only motions of the first category are allowed to be moved twice on top of the same motion. Since a faithful representation of the debates in not the focus of the model this has as an effect the abstraction of the debating activity involving the main motion so that a motion is only moved once without losing the properties that we are interested in representing. The result of this abstraction upon figure 3.8a is shown in figure 3.8b



(a) Example without the enable_moveOnce option (b) Example with the enable_moveOnce option

Figure 3.8: Solving times for incremental scope values

This has consequences relative to what we are able to model. The advantages are that:

- The size of the model is reduced.
- The outcome of a session due to the modification of the contents of its motions is still represented.

The disadvantages could be that:

- The effect of dependencies between eventual multiple moves of a motion on top of the same motions will not be represented.
- The effect of the order in which the multiple moves are made will not be represented.

The assumptions made are therefore that :

- The order of occurrence of multiple moves of the same motion does not matter.
- There are no dependencies between the precedence branches created from multiple moves of the same motion. Only incidental motions can create dependencies between precedence branches of motions.
- The fact that the debate can modify the opinion of the voters between two moves of the same motion does not matter.

- *The whole_session, session_prefix and session_suffix options* : if these predicates are used, the execution traces of a session state machine can be representing either a whole session with initial and final conditions being met over the first an last session states, a session prefix or a session suffix. In sub-model 1 we consider that a session is terminated when a session cannot navigate to a next business inside its order of business.
- *The reach_all_motoins option* : if this predicate is used, only motions that experience a change in their state will be represented. This means that all the motions represented will actually be involved in the ongoing session.
- *the dynamic_mode_only option* : this option predicate will force the model to only represent static dependencies.
- *The encourage_move option* : when true, this predicate makes the analyser chose the instances in which the move actions are chosen in priority whenever its preconditions are true.

Limitations of the model

This section describes the limitations borne by the sub-model 1 and describes for this sub-model the behaviour of the solving time with respect to the limiting scope values of the *run* command.

The orderings on Businesses and SessionStates. In a first attempt, we had used the *Ordering* utility supplied by the Alloy framework over the Business and SessionState signatures because these where concepts that had an order upon them by definition. But, due to the specification of the *Ordering* module this was constraining the Alloy analyser to produce each time it was looking for an instance exactly the upper bound number of Business atoms and SessionState atoms specified by the scope. The user defined scopes for Businesses and SessionStates where no longer upper bounds but exact bounds. This was an important hindrance regarding the modelling of whole sessions : if the analyser didn't find an instance when running a given predicate, there where now external reasons possible for it instead of reasons only tied to the content of the predicate and to the model itself. It became also possible that the the failure to find an instance was due to the scope for the Business signatures which was maybe too high when there was not enough session states in order to deal with all these businesses inside one session. The situation regarding the SessionState signatures was even worse since the user had to know the exact number of them that were necessary to produce the desired situation described inside the predicate. In summary the use of this utilities created a pernicious dependency between the model in itself and the analysis scope definition and also reduced notably the analysis flexibility in the context of discovering properties whose scopes are a priori unknown. The last paragraph of [15, sect. 3] mentions this problem regarding the specification of the *util/ordering* module. A way to work around this problem is explained in [3, p. 185] and consists in creating dummy abstract top level signatures for the signatures that one would like to order. The *util/ordering* module should then be used on the abstract signature and constraints specified only upon the children of the abstract signatures. This should allow the analyser to decide how many children will be instantiated. But this is only a partial solution since it necessitates to complicate the model because of purely technical concerns related to the tool and not the domain.

Another problem caused by the *util/ordering* module specification regarding analysis concerns is mentioned in the Alloy reference book [3, p. 183] which is that it hinders the flexibility of the analyser when verifying safety properties which will be explained in the analysis part of this document.

Our choice has therefore been to renounce to the use of the *Ordering* utility at the acceptable cost of writing ourselves the needed ordering relations.

One positive side effect is that it could allow the model to represent "parallel universes", that is, instead of representing traces, instances would be in the form of execution trees : different states could lead to the same state or the other way around, which could be useful for further modelling work. However one of the drawbacks was that within the same hardware constraints it took more time to find instances.

The types of specifications in the RONR As said before our model yields a representation of a session in its different states. Most of the specifications that the RONR gives about the domain entities and their properties are under the form of a fact with respect to the other existing elements of the domain. These specifications are relatively easy to express using simple and straight forward first-order-logic sentences. For example a rule saying that *Secondary amendments cannot be amended (§12 Standard descriptive characteristics 6.)* can be translated into the local constraint written as follows :

```
some (isAppliedTo & Amend) => no (@isAppliedTo.this & Amend)
```

Nevertheless we have encountered some specifications that are under the form of an assumption with respect to elements of the domain that do not exist in a given instance but that could exist in another instance having specific characteristics. An example is the description of which motions the motion to Amend can yield to. (§12 Standard descriptive characteristics 1. b) :

It yield to any privileged or subsidiary motion (other than Amend to which the motion that it proposes to amend would yield...)
This rule about precedence makes the possibility of a motion to Amend to yield to another one dependant of the capability of the motion that the motion to Amend takes precedence over to yield precedence to a motion of the same class in the hypothetical situation in which Amend didn't exist. In other words it is describing the motion to Amend as a "transparent" motion regarding precedence. Figure 3.9 illustrates this situation. Since Alloy uses a declarative language meaning that values can only be assigned once to the relations inside its universe, this necessitates that the model be able to produce an instance representing the actual session we are interested in along with a hypothetical session whose characteristics validate the first one. So typically one would have to describe an auxiliary world of atoms whose purpose is only to validate the main world when needed. In formal therms, the difficulty of this formulation is that it necessitates the use of Computational Tree Logic whereas our instances are under the form of traces, reasoned upon using rather Linear Temporal Logic. This is the only constraint within our modelling scope that was actually not expressible using Linear Temporal Logic. We suggest a solution to this based on the definition of a utility relation.

An idea to work around the difficulty introduced by this formulation could be to create an auxiliary relation to help us verify that Amend has the right to yield to some motion in such a situation.

Let m be a MainMotion, a be a motion to Amend applied to m and sm a motion that a yields precedence to and that applies to a as shown on figure 3.9:

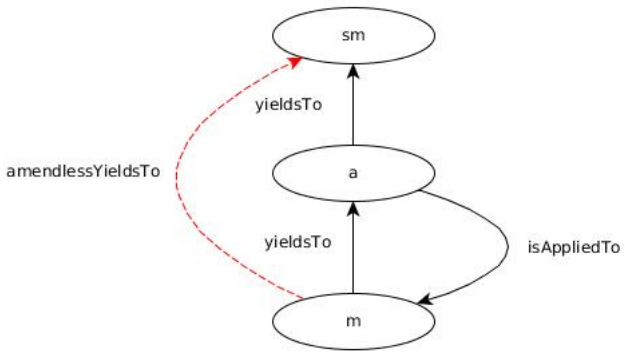


Figure 3.9: A circular definition of constraints

One could think of making use of an *amendlessYieldsTo* relation that would be a relation identical to the *yieldsTo* relation with the only difference that it would skip the Amend motions encountered inside the precedence tree as shown in the figure. But this would be an erroneous way to try to formulate the given constraint since the relation linking m and sm would be built using the relation linking m to a and a to sm. The latter links are supposed to exist under the very constraint that the direct link between m and sm exists. This is a circular definition of constraints that will end up saying nothing relevant about the validity of a link between a and sm.

Another solution could be to try to define a completely new *amendlessYieldsTo* relation imitating the *yieldsTo* relation. The definition of it will be scattered across the model in exactly the same way as the definition of *yieldsTo* and would only have a variation each time it would be dealing with motions to Amend. The problem with this is that this would in terms be equivalent to a complete rewriting of the model with more relational variables for the sake of rendering Amend alone transparent.
A last solution could be to take advantage of the fact that the static dependencies are independent of the state and to extend the definition of the *yieldsTo* relation so that it does not modify its

original desired behaviour. This is the solution that is one of the focuses of the sub-model 2.

Solving times for incremental scope values In order to show the limitations of the model with respect to the scope defined for its signatures we present the solving times for different scopes. First we define a single upper bound scope, then we freeze signature scopes in order to observe the behaviour of the solving time when the scope of the SessionState signature is incremented alone.

Execution environment of the Alloy analyser is the following:

Options : whole_session, dynamic_mode_only, no ErrorFlag
Solver : SAT4J
Machine characteristics : memory 3.7 GiB, Processor Intel® Core™ i3-2367M CPU @ 1.40GHz × 4, os type 64-bit

Command	Time (ms)
Run show for 1	736
Run show for 2	350
Run show for 3	421
Run show for 4	1306
Run show for 5	919
Run show for 6	10278
Run show for 7	6453
Run show for 8	6159
Run show for 9	20135
Run show for 10	39468
Run show for 11	3547
Run show for 12	56483
Run show for 13	376959
Run show for 14	71662
Run show for 15	31580

(a) Incremental global scope values

Command	Time (ms)
Run show for 9 but 1 SessionState	2855
Run show for 9 but 2 SessionState	13714
Run show for 9 but 3 SessionState	2031
Run show for 9 but 4 SessionState	2488
Run show for 9 but 5 SessionState	2979
Run show for 9 but 6 SessionState	4368
Run show for 9 but 7 SessionState	9757
Run show for 9 but 8 SessionState	15343
Run show for 9 but 9 SessionState	17414
Run show for 9 but 10 SessionState	31915
Run show for 9 but 11 SessionState	24581
Run show for 9 but 12 SessionState	17401
Run show for 9 but 13 SessionState	64079
Run show for 9 but 14 SessionState	112772
Run show for 9 but 15 SessionState	13868
Run show for 9 but 16 SessionState	24794
Run show for 9 but 17 SessionState	8769
Run show for 9 but 18 SessionState	39975
Run show for 9 but 19 SessionState	436311
Run show for 9 but 20 SessionState	557511

(b) Incremental SessionStates scope values

Figure 3.10: Solving times for incremental scope values

Figure 3.10 shows the execution times that we have recorded while executing the Alloy Analyser by command line. We have used the *System.currentTimeMillis()* Java function which is the same used inside the Alloy.4.2 source code in order to measure on the Conjunctive Normal Form (CNF) generation time and the solving time itself. In our measures, we only sample the time before and after the execution of the outermost wrapping Alloy analyser class, thus capturing all these measures indistinctly.

In figure 3.10a Even though the increase in execution time is not monotone, one can notice that the scope values correspond to a series of floors: the first floor goes from values 1 to 3 among which the first and the second scope do not yield any instance due to the insufficient number of states in order to form a whole session ; from 4 to 5 the execution time approximately

doubles; another notable increase is visible from 5 to 8; for a value of 9 and 10 the time increases by a factor of two to three before it falls radically at a value of 11; solving time for values of 12, 14 and 15 are of the same order while the solving time for a scope of 13 is higher by a factor of approximately 10.

Figure 3.10b has been established choosing the fixed scope value of 9 for the Businesses and Motions respectively, making only the upper bound for SessionStates vary. The time variations have the same characteristics than the previous result about monotonicity of the increase of values. As previously, analysis for 1 and 2 States do not yield any instances. There are relatively stable floors from value 1 to value 7 and from value 7 to value 12; from values 13 to 18 the variation is irregular until values 19 and 20 for which solving time becomes higher with a magnitude of about 10 than in the previous range.

We think that the existence of these floors are either to be attributed to the memory management done by our computer or to a memory optimization performed periodically by the analyser.

3.2.3 Sub-model 2 : focus on the motion to Amend

The motion to Amend appeared as being a frequent and important motion during parliamentary sessions and that is the subject of interesting exceptions regarding the precedence rules. The sub-model 2 sets about enlarging the modelling scope around the motion to Amend.

Modelling choices

In order to model additional constraints regarding the motion to Amend, the following abstractions upon the domain concepts have been applied using the sub-model 1 as a basis with the following variations :

- *There is only one business* : we are interested on the behaviour of the motion to Amend in the context of the settlement of a single business.
- *Some terminal incidental motions are instantiated* : in the sub-model 2 we decide to put a focus on the motion to Amend. In its definition, this motion has a particular interaction with three incidental motions that we have thus also implemented. The choice has been made to reduce the universe of Incidental motions to these three motions. This could therefore reduce the number of interesting cases in which the former incidentals interact with the motion to Amend on one hand, but on the other hand keeping the possibility of anonymous incidental motions could induce cases that do not exist in reality if no terminal incidental would ever actually be moved in the described abstract way.
- *Full representation of the constraints upon the motion to Amend* : the most interesting of these constraints being the "transparency" rule [1, p. 131] possessed by this motion. The formulation of this rule can be found in §12 Amend, Standard descriptive characteristics 1.b) and says that *"[The motion to amend] yields to any privileged motion or subsidiary motion (other than Amend) to which the motion that it proposes to amend would yield"* Expressing the particularity of this rule was one of the limitations explained in the previous section presenting the sub-model 1. The difficulty will here be worked around using our static versus dynamic dependencies decoupling.

The utility concepts used for the sub-model 1 are also present in sub-model 2. The following utility concepts have been added or undergo a variation :

- *Auxiliary atoms and the disable_auxiliary_atoms option :* This is the solution that has been modelled in order to address the circular constraint definition problem mentioned in the section about sub-model 1. When this predicate holds and a motion to Amend yields to another motion, the context in which the motion to Amend appears is reproduced. A

set of motions and Businesses are identified as the context of the motion to Amend. These motions are equipped with auxiliary relations that link each of them to one auxiliary atom of the same type. We then force these auxiliary atoms to be involved in exactly the same relations than the atoms they are linked to. An example of such auxiliary atoms is shown in figure 3.11.

- *The whole_session and session_suffix options* : the slight variation is that in sub-model 2 we consider that a session is terminated if the current business has been disposed of.

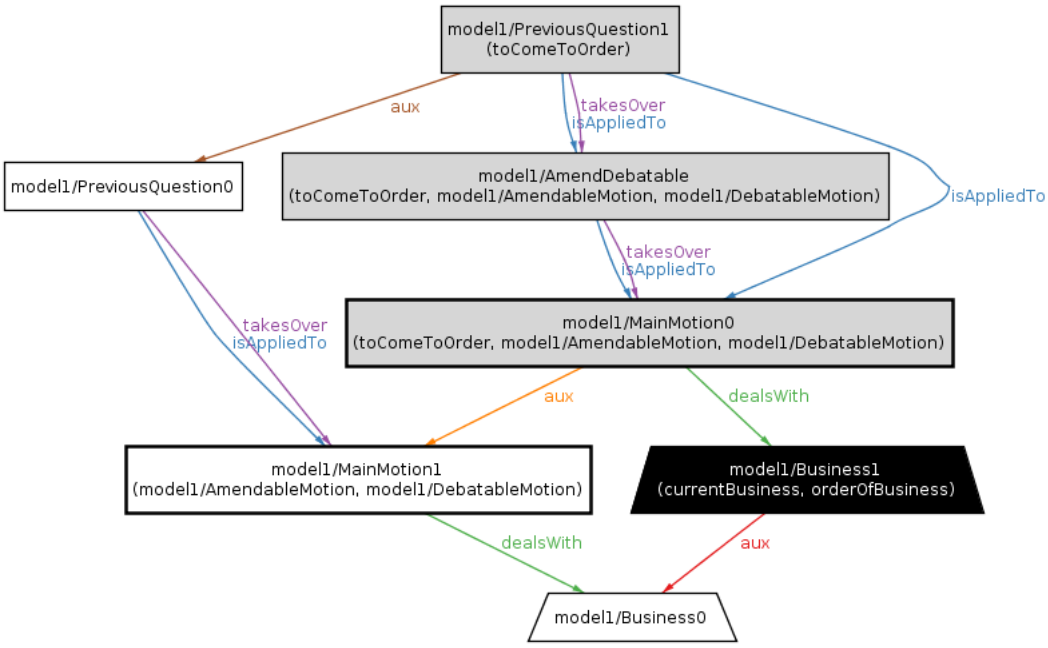


Figure 3.11: Auxiliary atoms used to validate the taking of precedence over a motion to Amend.

This device therefore addresses the two concerns about duplicating the Businesses and Motions that are part of the context of the motion to Amend and testing whether a motion succeeding to Amend could directly succeed to the motion that Amend succeeded to :

The analyser must be able to recognize alone the atoms needing auxiliary atoms due to the presence of the motion to Amend.
The analyser must be able to blindly reproduce the given configuration when for each existing transparent motion to Amend.

- *The Aux singleton:* this singleton atom knows about all the motion to Amend atoms that need to be transparent inside an Alloy instance.

Limitations of the model

This section describes the limitations borne by the sub-model 2 and describes for this sub-model the behaviour of the solving time with respect to the limiting scope values of the *run* command.

Solving times for incremental scope values Execution environment of the Alloy analyser is the following:

Options : whole_session, dynamic_mode_only, no ErrorFlag
Solver : SAT4J
Machine characteristics : memory 3.7 GiB, Processor Intel® Core™ i3-2367M CPU @ 1.40GHz × 4, os type 64-bit

Command	Time(ms)
Run show for 1	613
Run show for 2	402
Run show for 3	827
Run show for 4	1348
Run show for 5	3681
Run show for 6	4364
Run show for 7	7282
Run show for 8	15679
Run show for 9	76210
Run show for 10	13533
Run show for 11	119323
Run show for 12	141572
Run show for 13	416508
Run show for 14	15414
Run show for 15	749560

(a) Incremental global scope values

Command	Time(ms)
Run show for 9 but 1 SessionState	6624
Run show for 9 but 2 SessionState	4701
Run show for 9 but 3 SessionState	2976
Run show for 9 but 4 SessionState	2991
Run show for 9 but 5 SessionState	5223
Run show for 9 but 6 SessionState	16764
Run show for 9 but 7 SessionState	20916
Run show for 9 but 8 SessionState	22887
Run show for 9 but 9 SessionState	72475
Run show for 9 but 10 SessionState	79002
Run show for 9 but 11 SessionState	119328
Run show for 9 but 12 SessionState	139719
Run show for 9 but 13 SessionState	188398
Run show for 9 but 14 SessionState	4761
Run show for 9 but 15 SessionState	594312
Run show for 9 but 16 SessionState	1138181
Run show for 9 but 17 SessionState	622944
Run show for 9 but 18 SessionState	25054545
Run show for 9 but 19 SessionState	2110992
Run show for 9 but 20 SessionState	677563

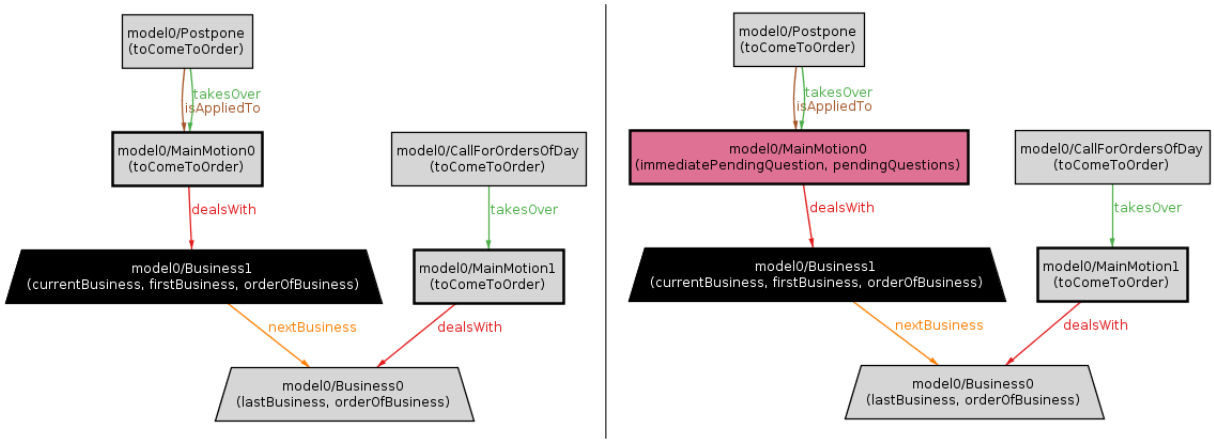
(b) Incremental SessionStates scope values

Figure 3.12: Solving times for incremental scope values for model 2

3.2.4 Example

We shall now go through the proceeding of a simple example session illustrating the descriptions made in the previous sections. The words in bald are the actions that happen so that the session transitions from a state to another.

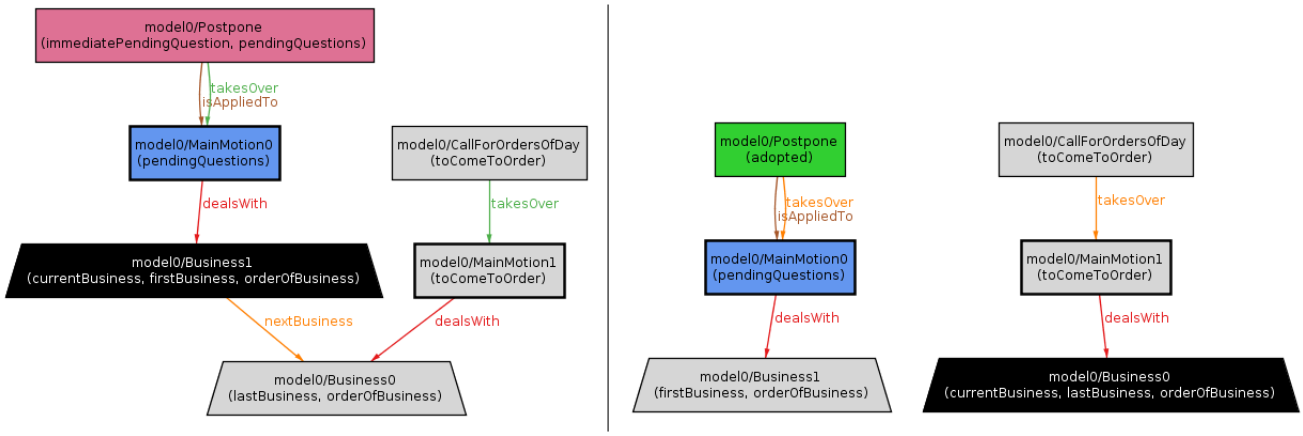
Table 3.6: Example (a)



Example (a) : at the beginning of the session there are two businesses in the order of business. Business1 is the first to be dealt with followed by Business2. The Business marked in black represents the one that is the current focus of the assembly. The motions that will potentially appear during the session are marked in gray and are part of the state set *toComeToOrder* as indicated inside the motion signatures. This first state of the session represents the static

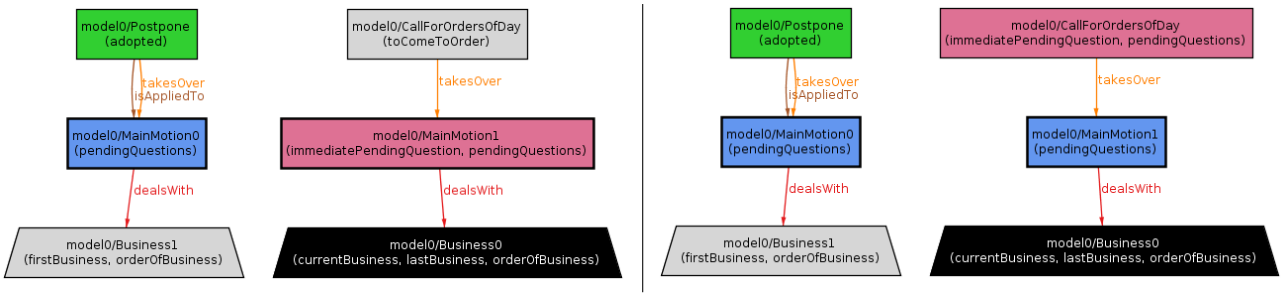
configuration of the session when all the motions will be called to existence. Between the two states of figure (a) MainMotion0 is **moved** and receives a second and becomes the immediate pending question marked in red, that is the motion about which debate is ongoing.

Table 3.7: Example (b)



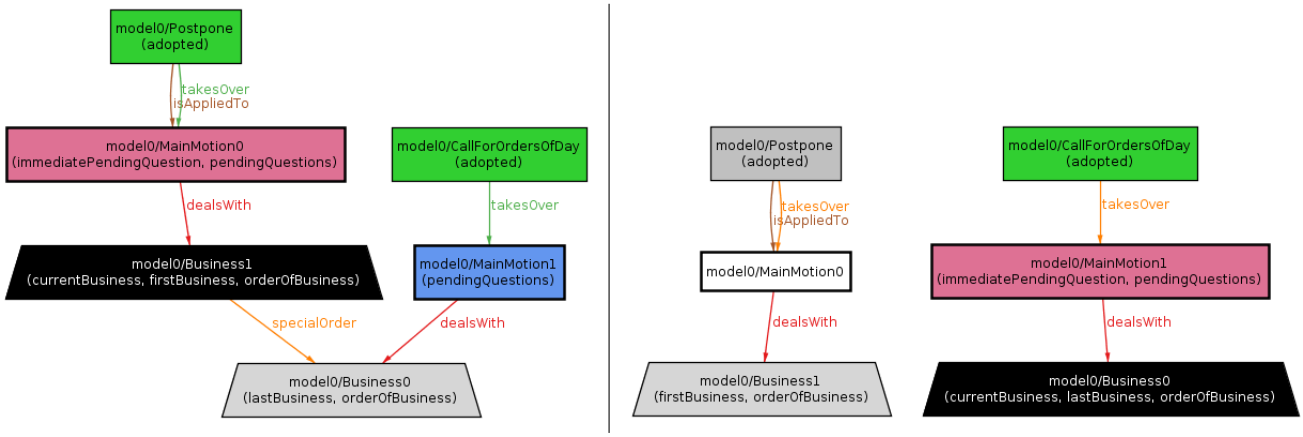
Example (b) : at some point of the debate about MainMotion0, a member estimates that it will be more constructive to debate MainMotion0 at another time. He **moves** to postpone the question until time t and his motion is seconded. The motion to Pstpone is now the focus of the assembly and the MainMotion0 marked in blue is left pending. Then, when debate time comes to an end about the motion to Postpone, the latter is put to a vote and is **adopted** by the assembly. It is marked green. The action to adopt causes the focus of the assembly to be put on the next Business to Business1 which is Business0. The navigation information through the normal order of business is updated and there is no longer a navigation link from Business1 to Business0.

Table 3.8: Example (c)



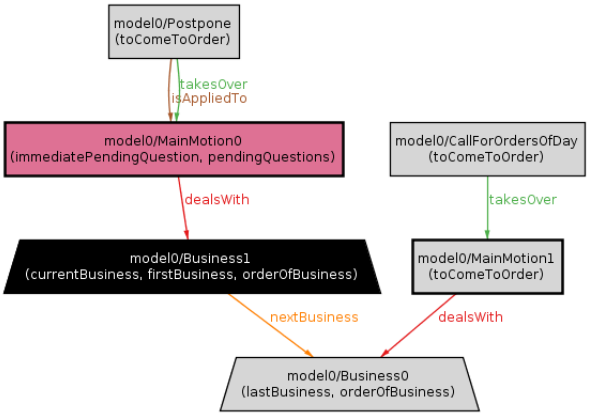
Example (c) : A member **moves** the MainMotion1 and is seconded so that it becomes immediately pending. Then during the debate about MainMotion1 the time t occurs. A member calls for the postponed order of the day. Without a second being required, the latter motion is **moved** and becomes the focus of the assembly as the new immediately pending question.

Table 3.9: Example (d)



Example (d) : Without being put to a vote and no member of the assembly requiring that the order of the day be set aside or that debate should continue about the previous question, the motion to call for the orders of the day is **adopted** and the focus of the assembly shifts back to debate about MainMotion0. The session remembers however that this is a special order that is being taken and a navigation link is created Business1 to Business0. Then MainMotion0 is put to a vote and is not adopted by the assembly. It is **disposed of** without being adopted and is marked in white. The loss of the MainMotion0 makes the focus of the assembly come back from Business1 to Business0 following the special order link. MainMotion1 is immediately pending again.

Table 3.10: Example (e)



Example (e) : MainMotion1 is **disposed of** without either being adopted and the session enters a final state that is one in which all the businesses inside the normal order of business have been navigated to.

3.2.5 Comparison with the leader election model[3, p. 171]

Section 6.1 of the Alloy reference book [3] presents some design choices and techniques through the modelling of the Leader Election protocol inside a ring of nodes executing concurrent sub-processes : nodes connected in a ring topology send each other their identifiers in order to allow the process with the highest identifier to be elected as the leader.

The author introduces the notions of state machine traces through this example and presents one of the ways in which they can be designed using the Alloy modelling language. A few points about our design are interesting to be compared against the design features of this model.

The states modelled for the Leader Election Ring state machine is composed of the sub-states of the Processes that run on each node of the ring. All the processes are concurrent and do not share transitions: between two steps, any number of processes can be active and executed in any order. In our modelling of a deliberative assembly session, if we chose to view each motion as a process, the latter are also concurrent but they do so while sharing transitions : their change from one state to another are constrained to each other. This is visible when looking at the formulation of the trace facts in both models in listings 3.3 and **??**. Time is the signatures playing the role of states in the Leader Election Model.

Listing 3.3: Leader Election Model trace fact

```
fact Traces {
    first.init
     all t: Time - last | let t' = t.next |
       all p: Process |
          step [t, t', p] or step [t, t', succ.p] or skip [t, t', p]
}
```

Listing 3.4: deliberative assembly Session Model trace fact

```
fact sessionTrace{
 initialSession[firstSessionState]
 all sBefore : SessionState - lastSessionState |
     let sAfter = sBefore.next |
     (one motion : Motion |
        move[sBefore, sAfter, motion] or
        adopt[sBefore, sAfter, motion] or
        disposeOf[sBefore, sAfter, motion])
      or
      error[sBefore, sAfter, Motion]
}
```

In the first case the trace fact targets all the Process atoms between each transition whereas only one motion at a time is targeted in the second case. Also, the Leader Election model has an operation that makes it possible that no change occurs between two states because this is a behaviour that is part of the protocol and that is interesting to be able to observe. The previous modelling choice is coupled with the use of the *util/ordering* module for the ordering of processes, adopting the solution mentioned in the previous sub-section.

While analysing the Leader Election model an important element stands out as a difference with the kind of process that we have designed which is the existence of a Machine Diameter.

A machine diameter as defined in [3, p. 185] is *"the maximum distance of a state from an initial state, where the distance between two states is the smallest number of execution steps that can take you from one to the other"*. In the example of the Leader Election model, an upper bound of the machine diameter can be calculated : it is the number of states starting from which every generated execution trace contains a loop. Section 6.1.5 explains an upper bound of a machine diameter can be calculated using the Alloy analyser. The major difference with the deliberative assembly sessions is that if we consider the state being composed of the *toComeToOrder, pendingQuestions, adopted, under[*]Order and [*]Business* sets sessions cannot loop : no two states are ever the same since motions are not considered again once disposed of. If we take the *toComeToOrder* set out of the state composition, then a session may theoretically loop (since we are not modelling the debate and motion contents) because the number of atoms is scoped, but this does not reflect something practically true : in a real life session one cannot

decide that the number of motions or businesses will be limited to a certain number. This would be contrary to the fundamental rights of the assembly members. Therefore, any execution trace looping between two states can be extended into a longer execution trace looping between the two same states.

The existence of a diameter has important implications regarding analysis. Since a machine diameter exists in the case of the Leader Election model, it becomes possible to claim that the analysis is complete with respect to the states when verifying safety properties for incremental values of the scope for the Times signature. In the case of deliberative assembly sessions, this cannot be claimed. The analysis conclusions will always be with scope-limited. However, even if one cannot estimate a machine diameter, a somehow interesting situation can be deduced by applying a similar method than the machine diameter calculation to sessions which is producing the longest session that is possible given a fixed number or determined types of motions. The details of this estimations shall be addressed in the analysis phase.

Chapter 4

Analysis phase

4.1 Modelling task correctness

The preliminary phase of the analysis consisted in verifying correctness of the translated constraints along the modelling phase. This preliminary analysis was necessary in order to detect the modelling errors or omissions that were not related to the consistency of the rules.

These tests where very useful in order to challenge our own comprehension of the rules since they compelled us to reformulate the translated constraints and verify their consistency with those contained in the models. We provide along with the Alloy code for the models a set of test files corresponding to the types of rules that where tested.

The latter test cases address the following concerns:

- The precedence rules should be correct
- The application rules should be correct
- Sets representing state should not illegally oscillate
- Actions should have expected consequences
- No domain elements violate their characteristics

We run predicates in order to verify that the model was not over constrained by producing instances of the expected behaviours and we checked using assertions that basic invariants held. The most efficient tests have been those regarding the states sets oscillation because they forced us to formulate the conditions that triggered the modification of the state sets in a different perspective. Listing 4.1 shows an example of a set oscillation test case.

Listing 4.1: deliberative assembly Session Model trace fact

```
pred a_motion_illegally_outside_the_pendingQuestions {

   some motion : Motion, before, after : SessionState |
      exited[before, after, motion, pendingQuestions ]and
      not move[before, after, motion] and
      not adopt[before, after, motion] and
      not disposeOf[before, after, motion] and
      (no pi : PostponeIndefinitely | adopt[before, after, pi ] ) and
      (no c : Commit | adopt[before, after, c]) and
      (no lot : LayOnTheTable | adopt[before, after, lot]) and
      not disposeOf[before, after, motion.questionOfPrivilege]
}
run a_motion_illegally_outside_the_pendingQuestions for 5 but 6 SessionState
expect 0
```

In this example, a predicate block is actually used to assert a property about the set of pending questions in all states and looks for an instance in which a motion atom is taken out of this state in a way that is not one described by the model constraints. The property is verified if the analysis yields no instance. The listing of the legal situations for a atom to leave the pending questions states highlights an effect that is a common to the PostponeIndefintely, Commit and LayOnTheTable motions.

4.2 Analysis properties

4.2.1 Formal categorization of properties

The goal of this work is to verify properties in a formal way.

A formal definition of a property : a process property can be viewed as a set of infinite sequences of states through which the process goes after the execution of an atomic step, action or event. Finite processes can be made infinite by concatenating the terminal state to its finite sequence of states. It can be prove that all properties belong to two groups of properties, the safety properties and the liveness properties, some being a combination of both.

Safety properties : a property is said to be of safety when for every sequence that is not inside the property it is possible to observe a finite execution prefix (up to some state) that makes us sure that the sequence is not inside the property an that it never will be no matter what happens in the next states. A safety property informally says that an undesired event never happens.

Liveness properties : a property is said to be of liveness when for all finite prefix of states of every sequence that is not inside the property, there will never appear any execution suffix that would allow it to be part of the property. A liveness property informally says that a desired event can always happen.

4.2.2 Property formulation and justification

There are two big families of properties that can be verified about a process : *safety* properties an *liveness* properties. Safety properties forbid a particular fact to become true at some step of the process whereas liveness properties demand that some particular event should always be able to happen at any step of the process. Verifying a liveness property means verifying that something desirable can happen infinitely often. For something desirable to be able to happen infinitely often means that the studied system has either an infinite number of states that are all different or that it has a finite number of states that it can loop between. With a model finder such as Alloy that only performs scope-bounded verifications, if one is able to generate this finite number of states then the verification of liveness properties can be done in a complete way. The Leader Election protocol compared against our modelling of the Robert's Rules of Order in the last section of the modelling phase chapter is a good example of such a system over which Alloy can be used to perform analysis of properties that are complete for a finite number of processes. But as explained in the same section, according to the principles underlying parliamentary procedures, on one hand, infinite execution of sessions does not make sense given that their goal is to settle a finite amounts of business. On the other hand, one cannot make before hand assumptions about the number of motions and businesses that will count a session.

About checking for safety properties, that is, discovering the violation of invariants and despite the fact that Alloy only succeeds at verification within a *scope*, if this scope is defined wisely enough the tool becomes then very efficient because once a undesired situation is identified in one state, one can easily infer that the safety property holds for any subsequent execution

trace appended to the trace up to that state. The verification for safety is more in accordance with the paradigm of deliberative assembly sessions in which infinite execution is aberrant and actions must not be *out of order*.

Safety properties have therefore been the focus of the analysis phase.

In accordance with the goals of a deliberative assembly, the properties that we have set out to verify in order to attempt the discovery of inconsistencies inside the rules of order where the following:

- *Property 1* : *No deadlock should occur*, that is, no session can reach a point where it cannot progress within the rules. By definition , a deliberative assembly deals with business. The rules of order should not over-constraint the behaviour of the assembly to the point that this main goal is not reached.
- *Property 2* : *All business should be dealt with completely* at the end of all sessions. This implies that motions settling business should go through a coherent life cycle (no motion should starve).
- *Property 3* : *Legitimate actions should be possible* in order to enforce the rights of the members, that is what they are "free" to do.
- *Property 4* : *There should be no incoherence* between the static and dynamic descriptions of the domain.

In our design of models for the Rules Of Order, the three first properties can have two different origins:

- A trace enacting a session cannot reproduce a static description because the latter are incoherent with the dynamic descriptions.
- The static and dynamic descriptions are coherent but a certain situation still cannot be enacted by a session trace.

The rules of order provide within themselves for mechanisms in order to avoid undesirable situations. Concerning deadlocks and starvation, respectively the incidental motion to *Suspend the Rules* can be applied within specific conditions in order to bypass the rules and a scheduling of floor access has to be done by the person directing the debate in order to alternate the interventions of members of different opinions and protect their right to speak. Concerning the settlement of business in an efficient manner each motion in the secondary motion class is by definition a mechanism to help reach this goal. However in the case of deadlock avoidance these are mechanisms that aim more at allowing the assembly to act in an exceptional way with a clear awareness of the infringement to the Rules Of Order. Our analysis is rather oriented at discovering and isolating in a formal manner contradiction within the rules as formulated by the RONR so that the causes can be analysed.

4.2.3 Translation to Alloy

We have suggested to translate the previous hight level goals into the following Alloy devices:

- *Property 1* The verification for deadlocks is done using the *error [sBefore, sAfter :SessionState, motions:Motion]* action predicate, the ErrorFlag signature, the *whole_session* option predicate and the action *suffix_predicate*. The initial conditions of the error action say that the error action should be chosen when no other action has its preconditions satisfied by the the before session state. When looking for instances in a given scope, the Alloy analyser will try to find a value for the relations variables that make the error action preconditions true. If it succeeds the error action is specified by default to bring the session immediately to a final state. This way, the analyser looking for whole sessions will also produce an

instance for the sessions that chose the error action so that it can be analysed inside the visualizer. Listing 4.2 shows the body of the error action predicate used in the sub-model 1.

Listing 4.2: Error action predicate for model 1

```
pred error [sBefore, sAfter :SessionState, motions : Motion] {
/** initial situation **/
all motion : motions |
   not move_preconditions[sBefore, motion] and
   not adopt_preconditions[sBefore, motion] and
   not disposeOf_preconditions[sBefore, motion]
//the before state should not be final
not finalSession[sBefore]
/** situation after the action **/
//the error state must compliy to the specifacations of a final state
finalSession[sAfter]
//The error state conserves the other attributes that are not specified by
    the final state
sAfter.immediatePendingQuestion = sBefore.immediatePendingQuestion
sAfter.orderOfBusiness = sBefore.orderOfBusiness
sAfter.firstBusiness = sBefore.firstBusiness
sAfter.lastBusiness = sBefore.lastBusiness
sAfter.adopted = sBefore.adopted
sAfter.toComeToOrder = sBefore.toComeToOrder
sAfter.pendingQuestions = sBefore.pendingQuestions
sAfter.underCommitOrPostponeIndefinitelyOrder =
    sBefore.underCommitOrPostponeIndefinitelyOrder
sAfter.underPreviousQuestionOrder = sBefore.underPreviousQuestionOrder
sAfter.laidOnTheTable = sBefore.laidOnTheTable
}
```

Since actions are not represented by visible atoms in our models, to allow an easier analysis of the visual representations of instances inside the visualizer, we have added a fact to the models that causes the existence of an *ErrorFlag* atom when ever the displayed trace has used the error action in order to reach a final state.

- *Property 2* We have looked for unfinished business using the *whole_session* action predicate and the *pendingQuestions* state set that describes a final session state. The *finalSession[lastState : SessionState]* contains the constraints that are minimal for a session to be terminated. In sub-model 1 and sub-model 2 we consider a session is terminated respectively if a session cannot navigate to a next business inside its order of business and if the current business has been disposed of. We run a predicate against the model that specifies constraints are about the composition of the *pendingQuestions* state sets at the end of a session about whether it is left empty or not .

- *Property 3*The verification for the possibility of legitimate actions was less straightforward than the verification of the two previous properties. We have encountered two levels of difficulty. Firstly , Alloy allows one to reason upon individual traces by leverageing the expressiveness of first-order-logic but does not allow to reason richly enough upon sets of traces. We designate by the term "legitimate action" an action that "is free to happen". Therefore, to be able to verify that an action can freely be made, we have verified the possibility of two different situations, implying the execution of two different predicates. One predicate is used to find the instance in which the given action happens and the other is used to find the instance where the given situation does not occur so that if one of the predicates yields an instance then the second predicate should yield an instance as well. Moreover, exact scope values should be specified for the number of session states

so that scope concerns do not interfere in the possibility or not for an action to happen. Secondly, the term "legitimate action" introduces the difficulty of its meaning in practice. For example moving a motion is a fundamental right of members in the absolute. But in practice conditions must be respected for the motion to be moved. So in order to verify that a legitimate action is always possible one must be able to describe the latter legitimate action. But the number of possible situations is important. This implies therefore that the verifier have heuristics according to which he would like to check the freedom of action inside a session.

Thus the overall method to verify property 3 uses 2 predicates in which a *configuration* and an *action* must be described. Listing 4.3 shows a template for the writing of such a verification that succeeds in sub-model 1.

Listing 4.3: Template for Property 3 verification

```
pred configuration [breakpoint : SessionState]{
-- Describe dynamically some session prefix here
some s1, s2, s3, s4 : SessionState, m1: RootMotion, m2: LimitExtendDebate |
   (s1+s2+s3+s4).(pendingQuestions) = (m1+m2) and
   initialSession[s1] and move[s1, s2,m1] and move[s2, s3, m2] and adopt[s3, s4,
       m2]
   and disposeOf[s4,breakpoint , m2]
}

/** should both succeed **/
pred some_motion_is_possible_for_a_given_session_configuration_1 {
   some s1, s2 : SessionState , motion : /*Chose motion here*/ Amend |
      configuration[s1] and move[s1, s2, motion]
}
run some_motion_is_possible_for_a_given_session_configuration_1 for 7 but
    exactly 7 SessionState

pred some_motion_is_possible_for_a_given_session_configuration_2 {
   some s1, s2 : SessionState , motion : /*Chose motion here*/ Amend |
      configuration[s1] and not move[s1, s2, motion]
}
run some_motion_is_possible_for_a_given_session_configuration_2 for 7 but
    exactly 7 SessionState
```

Therefore, we believe that the verification of this property would benefit from the analysis of a domain expert in order to establish whether for instance equivalence classes of session configurations could be defined allowing the verification of this property for groups of sessions presenting the same characteristics rather than for individual sessions.

- *Property 4* The coherence between the descriptions of a session in the RONR that we have categorized as static and dynamic is verified using the *encourage_move*, *whole_session*, and *reach_all_motions* option predicates. An assertion checks that whenever the session state machine in biased to include in its dynamic processing all the motions that can be moved, it also includes all the motions that are statically called to existence. The is necessary to verify this property because otherwise all the cases in which the members simply choose with their free will not to call to existence all of the motions inside the *toComeToOrder* state sets will be found as spurious counter-examples.

4.3 Analysis outcomes

4.3.1 Targeted properties analysis

After running our properties translated to the Alloy language the outcomes where the following:

- *Property 1* : no counterexamples illustrating a violation of the property 1 where found within our simplifying hypothesis and chosen scope values; this meaning that after the analysis of the counterexamples found, none was a case discovering a violation of the Rules Of Order. Figure 4.1 shows one of them.

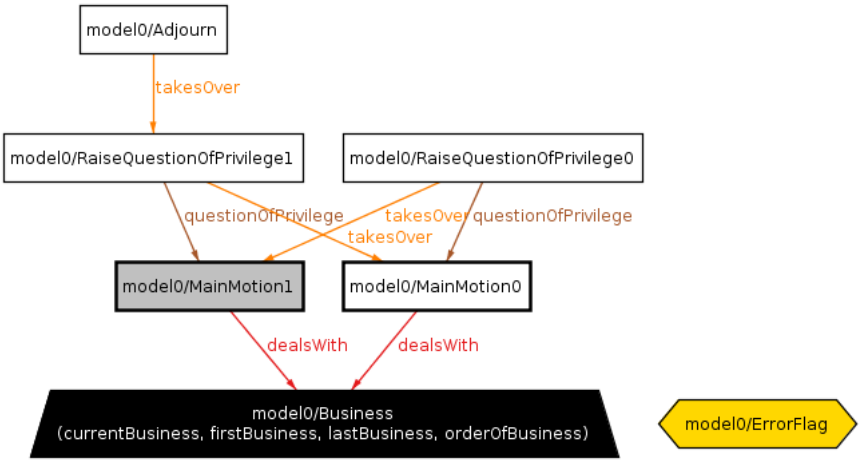


Figure 4.1: A counterexample to *property 1* in sub-model 1

  In this counterexample, an impossible situation is represented in which questions of privilege are raised in a crossed manner. This situation is chronologically impossible and provokes the existence of an ErrorFlag. This situation is not the fact of a contradiction in the Rules Of Order but of a limitation of our model that was missing a utility rule preventing this configuration to happen.
- *Property 2* : the verification that all business should be dealt with completely according to our simplifying hypothesis, modelling scope and formulation of the former property in Alloy did not yield any counterexample that constituted a violation of the property 2.
- *Property 4* : our analysis has not yielded any violation of the targeted property within our simplifying hypothesis and chosen scope values : counterexamples where indeed found, however none was a case illustrating a violation of the Rules Of Order.
  Figure 4.2 shows one of these counterexamples in which the presence of the motion to Call For The Orders Of The Day on top of the last business was causing sessions to reach the end state without moving this motion. All the counterexamples found proved to be illustrating the same configurations with only slight variations. The reason for this is to be found not inside contradictory Rues Of Order but are to be attributed to the fact that calling for the orders of the day at a time where a session had no more businesses to deal with is something that could not be described statically without creating a dependency between static and dynamic representations inside the models.

To summarize the outcomes of our property verification, our analysis has not yielded any violation of the targeted properties within our simplifying hypothesis and chosen scope values. However this phase has highlighted interesting facts about our models, their limits and some characteristics of the modelled domain.

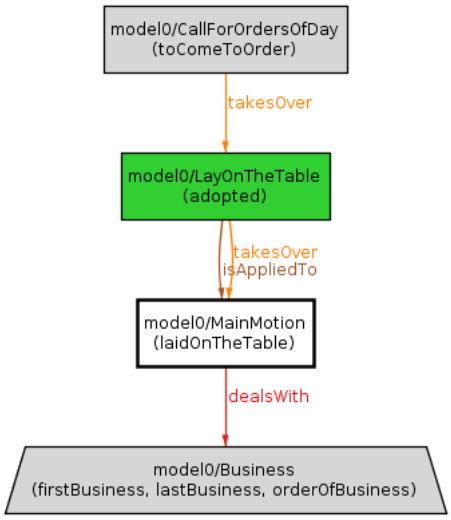


Figure 4.2: A counterexample to *property 4* in sub-model 1

4.3.2 Other analysis observations

The data structures emerging from the models

One of the strengths of the Alloy modelling tool is that it allows the modeller to build relations from elementary links between the domain objects, drawing progressively the nature of this relation at tuple level. The outcome is that Alloy can be used to discover or enforce interesting relational topologies. During our analysis of the RONR two data structures with particular characteristics appeared. We consider this as an interesting observation since these data structures are not named explicitly as such in the RONR but lend their characteristics to the correctness of the Rules Of Order.

- **The precedence Tree** The precedence Tree answers to the manner in which the RONR describes the settling of a main motion. It holds the "static" descriptions of the processing of a main motion that are descriptions in terms of how could look the final precedence relations between motions once all all of them have been moved but giving no precision about the actions that lead the session gets to theses configurations.
  A motion takes precedence over another when it comes after it and becomes the new focus of the assembly. The tree shape is the consequence of the fundamental principle that says that only one question is dealt with at a time : because two motions cannot become immediately pending at the same time and then yield precedence to another, the precedence relation forms an acyclic connected graph.
- **The application graph**
  A motion is applied to another motion in order to modify its processing. The application structure does not have a well defined form as a result of the diversity of the nature of the motions and their purpose.

Rule entailment

The implementation of tests helps the modeller discover some properties about the model. We have realised that new rules statically described can be more easily inferred from the old ones and their pertinence analysed.

For instance, while verifying that our modelling of the precedence ranks among the subsidiary motions was not over constrained we have tried to produce an instance showing all subsidiary motions inside a single precedence branch and have discovered that the 7 subsidiary motions

could not all become pending at the same time. This to us highlighted the ambiguousness of the sentence used to describe the precedence rank for subsidiary motions in *§§ Description of classes and individual motions, Subsidiary Motions : "The subsidiary motions are listed below in reverse order of rank - which is the chronological order in which they would be moved if all of them became pending at one time"*

Longest session with defined motions

If we set the number of businesses and the types and configuration of motions inside a session. We have realised that it became possible to attempt to yield an instance that represented the longest session containing the describes characteristics by applying the following method:

Chose a configuration for the motions and business
Enable the *whole_sesion* option
Show instances with an incremental number of SessionState
Stop the incrementation when no instance can be found

4.3.3 Limitations

Limitations of our models have been derived from the verification of our formulated properties regarding the decoupling of static description and dynamic description.

As illustrated by the impossibility of crossed questions of privilege to be raised or the presence of a motion to Call for the Orders of the Day inside a single business session , this separation can cause aberrant counterexamples to be generated. However, we consider that this does not limit the efficiency of the property verification since these counterexamples can be displayed inside the analyser and therefore can be very easily set aside as irrelevant.

The verification of property 2 was hindered by the fact that reasoning that could be done upon different execution paths of a session was limited. This is due to the fact that the instances generated by the analyser where in the form of execution traces and not under the form of execution trees. This can be resolved by allowing multiple actions to be chosen by the analyser between two states but would also have as a consequence of increasing the size of the model.

Regarding the definition of scope values inside the run and check commands of the targeted properties, we have chose values that we believed where large enough to enable us to discover intersting situations. These choices are the result of the experience about the solving times encountered along the incremental development and analysis of the models. But strictly speaking, since there is no notion of machine diameter, one cannot rely on any objective indicator when defining scope values.

A last limitation resides in the fact that our models try to entail the dynamic descriptions given in the RONR from the static descriptions but not the other way around. The possible scenarios that could be dynamically possible but not statically described are therefore not explored.

Chapter 5

Related work

In this part we compare the work done to similar case studies. Firstly we look at the case study of existing parliamentary assemblies in [10] , that is, organisations conducting the same type of business process as the ones described by the RONR. Secondly we compare our design choices about modelling a subset of the RONR with the ones presented by [11] which describes the use of the Alloy Modeller and Analyser in the context of business rules discovery.

5.1 Modelling Parliamentary Workflows a Case Study in Belgian Parliaments[10]

The document assembles the conclusions about a requirements elicitation and modelling work of parliamentary business processes in the practical case of two Belgian parliaments. The work performed is observed under tree perspectives: the identification and synthesis of the most important points to be viewed when looking at parliamentary proceedings, the research and comparison of modelling tools that comply to those requirements. And the conclusions to be drawn from this comparison.

Over a first phase the comparison of this work to our analysis of the RONR brings out a set of common points as well as a set of differences.

With regard to the common points, the two works are interested in the same type of business process. Notably, the underlying motivation of the RONR and the motivations described by the authors of this paper are the same : understanding what parliamentary business processes have in common in order to enhance portability of practices and adaptability of practitioners.

In section 1, the paper speaks of the modelling parliamentary proceedings as fitting into the process of maturation of the e-democracy in its form named e-legislation. Among the identified goals of this maturation process, the Alloy modelling tool appears well suited to achieve *procedural quality*. Indeed procedural quality is achieved by allowing an interactive and incremental modelling that is suited for re-engineering, leaving the *output quality* and the *participatory quality* to be better dealt with by the operational frameworks.

Section 2 describes a list of discovered requirements on the modelling language. The Alloy Analyser is able to totally or partially contribute to each of the listed levels. *The ability to capture behaviours* is ensured by the possibility that offers Alloy to describe an abstract state machine and visualize execution traces.The *abillity to capture responsibilities* is ensured through the relational paradigm in which agents and other domain concepts can be represented by atoms and their responsibilities by relations. Predicates or assertions can also be used in purpose of capturing responsibilities under the form on an invariant. *The ability to capture goals* is possible in the Alloy language because of the possibility that it offers to formulate and verify LTL properties. *The precision of the language* is ensured by the possibility to rely directly on the precision of relational algebra and firs-order-logic. The later facts makes the *easy to understand* requirement a bit less attainable considering the Alloy language alone. But put together with the visualization

tool, the Alloy framework offers simplicity and flexibility at the graphical modelling level *Tool support*, the Alloy framework taken singularly offers a complete support only at modelling level, the run-time level not being directly supported.

In section 3 the authors do a review of a non-exhaustive list of modelling techniques and comment about which modelling requirements the latter are best suited to capture. Use case diagrams backed with Sequence diagrams, Goal models, finite state machines backed with state diagrams and business process oriented languages are mentioned. **Goal models** and **finite state machines** are the closest resembling techniques to Alloy by allowing to reason about system properties through precise semantics on one side, and the other side by describing in an understandable way its behaviour.

Section 4, synthesises the observations of the previous sections and holds them up against the the elements of the e-legislation maturing process. It also makes clear that the mindset of a modeller shouldn't be about expecting a given tool to constitute a universal solution. The authors notably point out while activity diagrams are best suited to represent parliamentary workflows as they have observed them, the other notation are non the less useful at estimating the implementation margin that exist in a system with respect to abstract goals.

In the conclusion done in section 5, we believe that the Alloy modelling tool has its place among the backup techniques that *"allow the analyst to have a deeper insight of the system"*. Alloy can also be of significant help in the reduction of the effort and cost of writing and executing tests over the modelled system destined to be implemented as an application.

Nuances existing between the two works should however also be highlighted. Firstly, the works of this paper is based on the study of two particular instances of deliberative assemblies when what the RONR describes is a general set of rules not tailored to a particular instance. This does not have a particular consequence other than making the number of concepts to chose from for modelling purposes even larger in the case of particular instances. Secondly, the work was done starting with a primary elicitation and validation phase and the verification for specification consistency and completeness was done manually whereas our work with Alloy spans only verification but driven by an automated model finding tool. Our work thus appears as a more detailed and specific component of the kind of task conducted by the authors of the paper.

This work done over the modelling of parliamentary workflows in Belgian parliaments helps us discover a practical context in which our work can be inserted and sketches out additional motivation behind this task.

5.2 Simulation-Driven Approach for Business Rules Discovery[11]

The authors of this paper present their study of a typical business process happening inside a commercial organisation having to deal with the orders of its different types of clients. They summarize their work in tree phases: first a presentation of the design of an Alloy model of this process is made, then a second part walks us through the discovery of business rule using the Alloy Analyser before a general method is deduced from the previous steps.

The design choices laid out about this analysis work vary in various ways with respect to the ones made during our study of the RONR, and this mostly because of differences in the scope of the studies of the problem in both cases. Indeed, the former study spans across both elicitation and verification whereas in the study of the rules of order, verification is the main focus. Their final general method therefore simply describes the next steps to be taken in the face of inconsistencies discovered through the analysis of the Rules Of Order as described in the RONR.

Regarding the modelling phase, both works adopt a state-based approach in order to describe behaviour. However, an important difference is the number of states that each model allows. In the case of this paper, the exact number of states is always two : a pre-state describing the initial conditions of the system and a post-state describing the final conditions after the whole process

modelled has been executed. In the case of our modelling work, the model is not limited in the number of states that can be generated. This is because our model puts an emphasis on showing how a violation of the rules happen. Our model is interested in exposing which interleaving of events led to the unwanted situation.

Regarding the analysis phase, this work puts forward two main techniques for verifying the consistency of a set of rules describing business processes : the first is the use of assertions, that is, facts that should be true about all instances to check that invariants are respected; the second is the one by one observation by a domain specialist of generated instances in order for him to validate the behaviour of the system. While analysing the Rules of Order we have indeed made use of the former technique and for the same purpose. The second technique on the other hand served us in two ways : first as a limited guidance method for incremental modelling and testing and secondly and foremost as a work around manner for discovering counter examples by running predicates.

Chapter 6

Conclusion and future work

6.1 Conclusion

With Alloy modeller and analyser as formal verification tool, we have designed two sub-models of the proceedings of parliamentary deliberative assemblies as described by the Roberts Rules Of Order newly Revised. These sub-models cover a scope that has been defined in order to allow a simplified but realistic representation of parliamentary session. In the attempt to discover incoherences with the formulation of the Rules of Order, we have chosen to decouple the static or descriptive rules from the dynamic or operational rules by designing sub-models capable of illustrating these two aspects. We have been particularly interested in the mechanism of motions that are the most formal way in which business is brought before the deliberative assembly. Due to the important number of dependencies between motions we have captured the functioning of a session in its globality in a first sub-model which models the ordinary processing of business. We then focused the modelling scope around the motion to Amend which exerted interesting characteristics in its interactions with the other motions notably regarding precedence rules. Then we have made the Alloy Analyser use each of these sub-models to produces execution traces representing the states of deliberative assembly sessions thus representing one session by the means of an abstract state machine. The property of the Alloy atoms at the basis of the Alloy relational paradigm as being uniquely distinguishable from one an other was leveraged to unambiguously represent motions and describe the state of a session.

During analysis phase, we formulated safety properties with respect to the goal of Deliberative Assemblies as described by the Robert's Rules of Order and integrated verification mechanisms to our Alloy models in order to be able to observe whether these properties where violated or not with respect to the characteristics captured by our models. Even though no violations where discovered within our modelled scope, the analysis allowed us to learn interesting facts about the structure of our models and highlight some ambiguities.

Modelling and analysing parliamentary law processes as described by the Roberts Rules Of Order using Alloy has been an interesting and challenging task on several levels.

Firstly it has allowed to conduct a reflection about what can be strictly formalized about parliamentary law processes and helped contribute to the elaboration of a formal identity of parliamentary law processes using the state of the art techniques in the field of modelling.

Secondly it has revealed itself as a highly formative experience regarding software engineering methodology. Indeed, being directly confronted to the low level rigorousness of first order logic progressively trains the descriptive skills of the modeller and proves to be a thorough training of his methodological skills regarding test driven incremental development.

The work done using the Alloy modelling tool to design a model of this level of complexity leaves us with a certain number of lesson learned.

The implementer is often tempted to be adding more and more clauses to the model in order to get it to behave the way he wants but this has also as a consequence to increase the size of the model and thus the execution time. What should be done is reformulating (refactoring) the existing clauses as much as possible before trying to add new ones which is an exercise that can prove to be complex.

6.2 Future work

We have observed the general behaviour of the most common motions used during deliberative assemblies and the behaviour of the motion to Amend due to its status of a device that is most used and subject to a number of exceptions. With these works as a reliable basis, other motions exerting even more exceptional behaviour need to be studied in the same manner, notably the motion to *Suspend the Rules* or the motion to *Reconsider* that could exert interesting navigational behaviour.

Our design of deliberative assembly sessions abstracts a certain number of sub-processes. The voting process notably is a very important process in which roles and individual rights of members are subject to specific rules regarding the notions majority of the proceeding of debate. In third sub-model, the voting box should be opened and the actions happening during a session could be abstracted into their strict effect on the process of disposing of an individual motion.

An ongoing deliberative assembly is comparable to the execution of a virtual machine with a stack of sub-routines that consist in the settling of motions, motions are added on the stack, with an environment composed of dependencies with other elements of the session, settled and then removed from the stack, returning the context to the previous activity. This stands out in the fact that a state machine appeared to us as the most interesting way of representing the Robert's rules of Order parliamentary proceedings. The major difference resides in the fact that a virtual machine executes instructions that can be known beforehand whereas a deliberative assembly discovers, as the opinion of members shapes themselves, the resolutions that will be made, adopted or rejected. The scheduling task which is the role of the moderator is also different since the criteria for discriminating fairly between the persons that have the right to speak at a given moment is not trivially formalizable since it depends largely on how humans succeed to understand each other. However, and as mentioned in the case study reviewed about the modelling of parliamentary workflows, the degree to which the formal characteristics of parliamentary proceedings can also be formally enforced is not fully reached. In future work more studies should be conducted in order to provide the human component of these proceedings tools that help them integrate their action to the formal context of parliamentary law. Beyond modelling and post-analysis, user-friendly real-time simulation tools should be the next step.

Bibliography

[1] Robert Henry M. III, Honemann Daniel H.,Balch Thomas J. with the assistance of Seabold Daniel E., Gerber Shmuel. *The Robert's Rules of Order Newly Revised, 11th edition*. Da Capo Press, 2011.

[2] Sanghavi Alok, 21 May 2010. *"What is formal verification?"*(Online).EE Times Asia.

[3] Jackson Daniel. *Software Abstractions: logic, language, and analysis*. The MIT Press, 2012.

[4] `http://alloy.mit.edu/alloy/`.2012.

[5] `http://robertsrules.com/interpretations.html`. Official interpretations by the authors of robert's rules of order newly revised, © Copyright 2011.

[6] `http://www.omg.org/oceb/defbusinessprocess.htm`. Jon Siegel, Ph.D., Vice President, Technology Transfer, Object Management Group™. *In OMG's OCEB Certification Program, What is the Definition of Business Process?*, An OCEB Certification Program White Paper, May, 2008.

[7] Ould Martyn A, Venice Consulting Ltd. May 2005. Text of the lecture given at the launch of the book "Business Process Management – A Rigorous Approach" on 25 January 2005 in London. (Online) `http://www.bptrends.com/publicationfiles/05-05%20ART%20BPM%20A%20Rigorous%20App%20-%20Ould.pdf`

[8] *Workflow Management Coalition Terminology & Glossary*, WFMC-TC-1011, Issue 3, p. 8,p. 10, `http://www.wfmc.org/standards/docs/TC-1011_term_glossary_v3.pdf`, February 1999.

[9] Ould A. Martyn. *Business Process Management: A Rigorous Approach*.Meghan- Kiffer Press, 2005.

[10] Ponsard Christophe, Deberdt Gaetan, Tournemenne Joël. *Modelling Parliamentary Workflows a Case Study in Belgian Parliaments*. Août 2009

[11] Bajic-Bizumic Biljana, Rychkova Irina, Wegmann Alain. *Simulation-Driven Approach for Business Rules Discovery* In: Advanced Information Systems Engineering Workshops, CAiSE 2013 International Workshops, Valencia, Spain, June 17-21, 2013.

[12] `http://www.uml-diagrams.org/state-machine-diagrams.html`, *State Machine Diagrams*. Copyright © 2009-2016 uml-diagrams.org.

[13] Börger Egon, Dipartimento di Informatica, Universita di Pisa, Italy. *The Abstract State Machines Method for High-Level System Design and Analysis* In: Theoretical Computer Science, Volume 336, Issues 2–3, 26 May 2005.

[14] Darwen Hugh. *An Introduction to Relational Database Theory*.Ventus Publishing ApS, 2010.

[15] Cunha Alcino, HASLab - High Assurance Software Laboratory INESC TEC & Universidade do Minho, Braga, Portugal. *Bounded Model Checking of Temporal Formulas with Alloy* In : *Abstract State Machines, Alloy, B, TLA, VDM, and Z*. Y. Ait Ameur and K.-D. Schewe (Eds.): ABZ 2014, LNCS 8477, pp. 303–308, 2014, Springer-Verlag Berlin Heidelberg, 2014.

Appendix

Solving times samples

The multiple runs results regaring the solving times of which the table presented in the above chapters is a sample.

Command	Time (ms)
Run show for 1	858
Run show for 2	375
Run show for 3	513
Run show for 4	1410
Run show for 5	1162
Run show for 6	8575
Run show for 7	5435
Run show for 8	6266
Run show for 9	17241
Run show for 10	36906
Run show for 11	3647
Run show for 12	52886
Run show for 13	339343
Run show for 14	66972
Run show for 15	31106

Command	Time (ms)
Run show for 1	736
Run show for 2	350
Run show for 3	421
Run show for 4	1306
Run show for 5	919
Run show for 6	10278
Run show for 7	6453
Run show for 8	6159
Run show for 9	20135
Run show for 10	39468
Run show for 11	3547
Run show for 12	56483
Run show for 13	376959
Run show for 14	71662
Run show for 15	31580

Command	Time (ms)
Run show for 1	747
Run show for 2	512
Run show for 3	623
Run show for 4	1374
Run show for 5	1135
Run show for 6	8249
Run show for 7	5335
Run show for 8	6355
Run show for 9	17311
Run show for 10	35124
Run show for 11	3385
Run show for 12	51393
Run show for 13	337792
Run show for 14	66889
Run show for 15	30699

Figure 6.1: Solving times for incremental global scope values in 3 consecutive runs in model 1

Back cover page

Command	Time (ms)
Run show for 9 but 1 SessionState	2274
Run show for 9 but 2 SessionState	15980
Run show for 9 but 3 SessionState	2359
Run show for 9 but 4 SessionState	3069
Run show for 9 but 5 SessionState	2855
Run show for 9 but 6 SessionState	5244
Run show for 9 but 7 SessionState	9506
Run show for 9 but 8 SessionState	17200
Run show for 9 but 9 SessionState	19126
Run show for 9 but 10 SessionState	35636
Run show for 9 but 11 SessionState	28503
Run show for 9 but 12 SessionState	19765
Run show for 9 but 13 SessionState	68581
Run show for 9 but 14 SessionState	117213
Run show for 9 but 15 SessionState	13915
Run show for 9 but 16 SessionState	26057
Run show for 9 but 17 SessionState	9430
Run show for 9 but 18 SessionState	43450
Run show for 9 but 19 SessionState	423407
Run show for 9 but 20 SessionState	536587

Command	Time (ms)
Run show for 9 but 1 SessionState	2855
Run show for 9 but 2 SessionState	13714
Run show for 9 but 3 SessionState	2031
Run show for 9 but 4 SessionState	2488
Run show for 9 but 5 SessionState	2979
Run show for 9 but 6 SessionState	4368
Run show for 9 but 7 SessionState	9757
Run show for 9 but 8 SessionState	15343
Run show for 9 but 9 SessionState	17414
Run show for 9 but 10 SessionState	31915
Run show for 9 but 11 SessionState	24581
Run show for 9 but 12 SessionState	17401
Run show for 9 but 13 SessionState	64079
Run show for 9 but 14 SessionState	112772
Run show for 9 but 15 SessionState	13868
Run show for 9 but 16 SessionState	24794
Run show for 9 but 17 SessionState	8769
Run show for 9 but 18 SessionState	39975
Run show for 9 but 19 SessionState	436311
Run show for 9 but 20 SessionState	557511

Command	Time (ms)
Run show for 9 but 1 SessionState	2711
Run show for 9 but 2 SessionState	13379
Run show for 9 but 3 SessionState	2405
Run show for 9 but 4 SessionState	2037
Run show for 9 but 5 SessionState	3101
Run show for 9 but 6 SessionState	4576
Run show for 9 but 7 SessionState	8913
Run show for 9 but 8 SessionState	15267
Run show for 9 but 9 SessionState	18173
Run show for 9 but 10 SessionState	34873
Run show for 9 but 11 SessionState	23999
Run show for 9 but 12 SessionState	18081
Run show for 9 but 13 SessionState	61912
Run show for 9 but 14 SessionState	108042
Run show for 9 but 15 SessionState	13044
Run show for 9 but 16 SessionState	24194
Run show for 9 but 17 SessionState	9252
Run show for 9 but 18 SessionState	39045
Run show for 9 but 19 SessionState	392748
Run show for 9 but 20 SessionState	531219

Figure 6.2: Solving times for incremental SessionState scope values in 3 consecutive runs for model 1

Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve **www.uclouvain.be/epl**