

# MAT 7003 : Mathematical Foundations

(for Software Engineering)

J Paul Gibson, A207

paul.gibson@it-sudparis.eu

<http://www-public.it-sudparis.eu/~gibson/Teaching/MAT7003/>

## EvotingProblem

<http://www-public.it-sudparis.eu/~gibson/Teaching/MAT7003/L2-3-EvotingProblem.pdf>

2010: J Paul Gibson

T&MSP: *Mathematical Foundations*

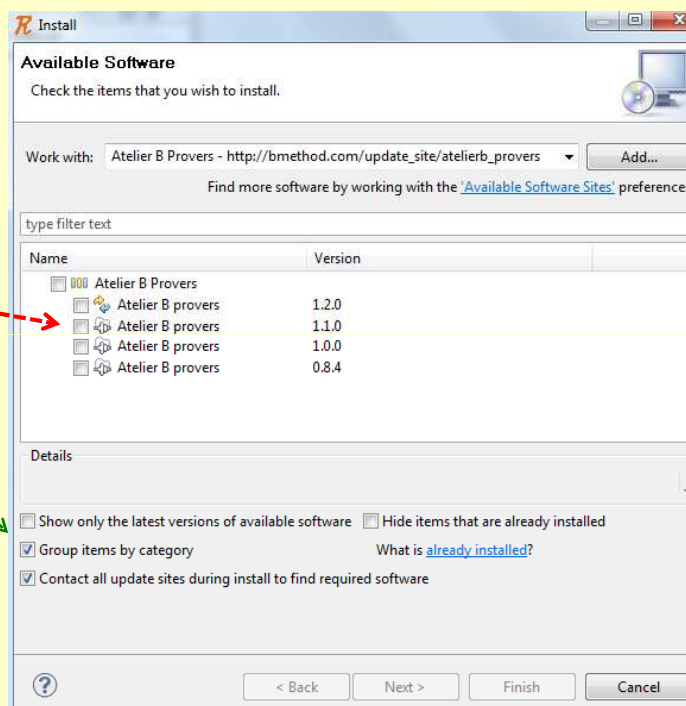
MAT7003/EvotingProblem.1

## Installing The Atelier-B Provers (RODIN Plugin)

For **Rodin 1.3.1**, you need to install Provers 1.1.0.

In the install window, uncheck "Show only the latest versions of available software", select 1.1.0 and click "Finish".

You may need to accept a license agreement, and restart RODIN



2010: J Paul Gibson

T&MSP: *Mathematical Foundations*

MAT7003/EvotingProblem.2

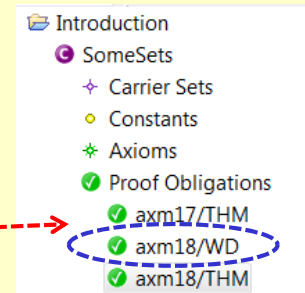
## Installing The Atelier-B Provers (RODIN Plugin)

You don't need these to complete the practical work (this week) but they are useful if you want to check that the expressions in your axioms are **well defined**.

To better understand what it means for an expression to be well defined, you should review the **SomeSets** context from last week, where we reasoned about family relationships.

For those that did not complete this work themselves, you can download a sample **SomeSets** specification from the website.

In the sample given, I have discharged the **proof obligations** using the Atelier-B proof plug-ins



## Installing The Atelier-B Provers (RODIN Plugin)

### CONTEXT

SomeSets

### SETS

PERSON

### CONSTANTS

Male

Female

Mothers

Fathers

Parents

MotherOf

FatherOf

ParentOf

ChildOf

DaughterOf

SonOf

Siblings

SisterOf

BrotherOf

```

axm1 : Male ⊆ PERSON // All males are persons
axm2 : Female = PERSON \ Male // Any person who is not male is female
axm3 : Mothers ⊆ Female // Only females can be mothers
axm4 : Fathers ⊆ Male // Only males can be fathers
axm5 : Parents = Mothers ∪ Fathers // Parents are mothers or fathers
axm6 : MotherOf ∈ PERSON → Mothers // All persons have a single Mother
axm7 : FatherOf ∈ PERSON → Fathers // All persons have a single Father
axm8 : ParentOf = MotherOf ∪ FatherOf // Your parent is either:
// your mother or your father
axm9 : ChildOf = ParentOf~ // Links ChildOf and ParentOf
axm10 : DaughterOf = ChildOf ▷ Female // Daughters are female children
axm11 : SonOf = ChildOf ▷ Male // Sons are male children
axm12 : Siblings ⊆ PERSON × PERSON // Siblings defines a binary relation between PERSONS
axm13 : ∀p. pe PERSON ⇒ // Nonreflexive; ie a person cannot be their own parent
      ¬( p▷p ∈ ParentOf)
axm14 : ∀p1,p2. ((p1▷p2) ∈ Siblings) ⇔ // Siblings have at least one parent in common
      ((p1▷p2) ∧ // And you are not your own sibling
      (ParentOf ▷ {p1} )n(ParentOf ▷ {p2} ) ≠∅)
axm15 : SisterOf = Siblings▷Female // Sisters are female siblings
axm16 : BrotherOf = Siblings▷Male // Brothers are male siblings
axm17 : ChildOf = DaughterOf ∪ SonOf // Theorem: A child is either a son or a daughter
axm18 : ∀p. pe PERSON ⇒ // Theorem: All persons have precisely 2 parents
      card({p}◁ParentOf)=2
    
```

Once you understand this, move on to the electronic voting problems that follow

## An Electronic Voting Problem – step 1: a referendum

Create a RODIN context that models a referendum, using the following sets and constants:

*Do you agree that ...*  
**YES**   
**NO**

### CONTEXT

**Referendum** // To model a referendum where there is a single yes/no response

### SETS

**RESPONSE** // The set of choices offered to the VOTERS during the vote

**VOTERS** // The finite set of people entitled to record a vote, but not all do so

### CONSTANTS

**yes** // one possible response of two

**no** // one possible response of two

**AllVotes** // Linking voters to responses

**CountYes** // The integer count of the yes votes

**CountNo** // The integer count of the no votes

Complete the **Referendum** context model with the **axioms** that specify the properties commented in **green** above

## An Electronic Voting Problem – step 1: a referendum

Now we can test your axioms with the following theorems:

```
AllVotes = { v1→ yes, v2→ no } ⇒ CountYes = 1 // A single yes vote count check  
AllVotes = { v1→ yes } ⇒ CountNo = 0 // A zero no vote count check
```

To add these you need to add constants  $v1$  and  $v2$ , and to specify axioms that they are different VOTERS.

You can use the **RODIN wizards** to help you build such a set of VOTERS

Note: these theorems can be thought of as test cases for validation

**OPTIONAL: Try to use RODIN to prove these theorems**



## An Electronic Voting Problem – step 2: a presidential election

Create a RODIN context that models a **presidential election**, using the following sets and constants:

**Smith**  
**Jones**  
**Gibson**      **X**  
**Park**  
**Green**

### CONTEXT

**Presidential**      // *In a presidential election, each voter who chooses to vote  
                         // must select a single candidate*

### SETS

**VOTERS**      // *The finite set of people entitled to vote in the election,  
                         // but who do not necessarily do so*

### CONSTANTS

**AllVotes**      // *A function linking each voter, who recorded a vote, to their selected candidate*  
**CountVote**      // *A function linking each candidate to the number of votes they have received*  
**Candidates**      // *The subset of voters who run as candidates in the election*

Complete the **Presidential** context model with the **axioms** that specify the properties commented in **green** above

## An Electronic Voting Problem – step 2: a presidential election

Test your **Presidential** context model with the theorem:

$$\text{AllVotes} = \{v1 \mapsto v2, v2 \mapsto v2\} \Rightarrow \text{CountVote}(v2) = 2$$

Here, both voters (v1 and v2) have voted for candidate v2

## An Electronic Voting Problem – step 3: a list election

Create a RODIN context that models a *valid vote* in a **list election**, that use the following constants:

<b>Smith</b>	<b>X</b>
<b>Jones</b>	
<b>Gibson</b>	<b>X</b>
<b>Park</b>	<b>X</b>
<b>Green</b>	

### CONSTANTS

```
Candidates // A subset of VOTERS
ValidVotes // All valid (non-empty) candidate lists which follow the minimum and maximum requirements
UnrestrictedValidVotes // No restriction from minimum or maximum requirements
```

Verify the following theorem:

```
Candidates = {v1,v2,v3} ⇒
{v1,v3} ∈ UnrestrictedValidVotes
```

**DO NOT** attempt to specify the count mechanism – you are not yet able to do this and we will come back to this in later classes

### Step 4 : A constrained list election

Add axioms to ensure a **minimum** number of choices and a **maximum** number of choices in **ValidVotes**

## An Electronic Voting Problem – step 5: a preferential election

Create a RODIN context that models a *valid vote* in a **preferential election**

<b>Smith</b>	<b>2</b>
<b>Jones</b>	
<b>Gibson</b>	<b>1</b>
<b>Park</b>	<b>3</b>
<b>Green</b>	

**NOTE:** You are not to get any assistance from the lecturer for this challenge!

Write axioms that validate the correctness of your specification of a vote being valid

**DO NOT** attempt to specify the count mechanism – you are not yet able to do this and we will come back to this in later classes

### Step 6 : A constrained preferential election

Add axioms to ensure a **minimum** number of choices and a **maximum** number of choices

## An Electronic Voting Problem optional step – Vote Parcels

In preferential elections, the count process usually requires votes to be stored in parcels (eg in Ireland). Then, as candidates are elected (or eliminated from the count) votes are transferred from one parcel to the next.

In computing terms, parcels correspond to a Stack (of votes) with LIFO behaviour

RODIN contexts can be used to specify behavioural requirements algebraically. We look at the example of a Stack of Votes; where you are required to specify the axioms that define – the count of the number of Votes in a Stack (parcel) and the transfer of a single vote from the top of one Stack to the top of a second.

## An Electronic Voting Problem optional step – Vote Parcels

### CONTEXT

VoteParcel // *A Stack (LIFO) of Votes*

### SETS

VOTES  
PARCEL

### CONSTANTS

noVotes // *An empty Parcel*  
addVote // *To any parcel we can add a vote*  
removeVote // *We can remove the last vote added to any nonempty Parcel*  
top // *We can access/read the vote on top of the Parcel*  
countVotes // *The number of votes in a Parcel*  
transferVote // *Move a vote from the top of the 1st Parcel to the top of the 2nd Parcel*

The AXIOMS are needed to give meaning (semantics) to the concepts.

## An Electronic Voting Problem optional step – Vote Parcels

Take some time to understand the **AXIOMS** given, below:

### AXIOMS

axm1 :  $\text{noVotes} \in \text{PARCEL}$   
axm2 :  $\text{addVote} \in (\text{PARCEL} \times \text{VOTES}) \rightarrow (\text{PARCEL} \setminus \emptyset)$   
axm3 :  $\text{removeVote} \in (\text{PARCEL} \setminus \emptyset) \rightarrow \text{PARCEL}$   
axm4 :  $\forall p, v. p \in \text{PARCEL} \wedge v \in \text{VOTES} \Rightarrow$   
 $\text{removeVote}(\text{addVote}(p \mapsto v)) = p$   
axm5 :  $\text{top} \in (\text{PARCEL} \setminus \emptyset) \rightarrow \text{VOTES}$   
axm6 :  $\forall p, v. p \in \text{PARCEL} \wedge v \in \text{VOTES} \Rightarrow$   
 $\text{top}(\text{addVote}(p \mapsto v)) = v$   
axm7 :  $\text{countVotes} \in \text{PARCEL} \rightarrow \mathbb{N}$   
axm8 :  $\text{countVotes}(\text{noVotes}) = 0$   
axm9 :  $\text{transferVote} \in ((\text{PARCEL} \setminus \emptyset) \times \text{PARCEL}) \rightarrow (\text{PARCEL} \times (\text{PARCEL} \setminus \emptyset))$

The context specification is incomplete.

TO DO: complete the specification by adding some axioms?

## Electronic Voting: Use Specification to Develop an Implementation

**TO DO:** FOR NEXT WEEK (teams or individually):

For any one of the specified machines, write a program in the language of your choice to:

- 1) Represent a valid vote
- 2) Count valid votes

Verify that your code is *correct* against the mathematical specification