E-Voting Requirements Modelling: An Algebraic Specification Approach (With CafeOBJ)

J. Paul Gibson[1], Department of Computer Science,
National University of Ireland, Maynooth.
Date: March 2005 (updated October 2005)

Technical Report: *NUIM-CS-2005-TR-14*

Key words: ADTs, PRSTV, Validation, Verification, Objects, Classes.

Abstract

We report on the formal modelling of the PRSTV count rules as used in Ireland, and detailed in previous work[2] An algebraic approach was used following an object-based modelling *style*. This specification model was then used to compare and contrast formal requirements modelling notation and (informal) natural language. We chose to use a minimum number of semantic constructs in the specification language (CafeOBJ[9]) for the sake of simplicity and accessibility. To conclude, we give an overview of the development process (tools and techniques) used in the specification, validation and verification of the requirements model.

Contents

1 Introducing the PRSTV e-voting problem **2**

2 The role of requirements models **2**

3 The Object-Based Struture **2**

4 An Introduction to Algebraic Specification and CafeObj **3**

5 The Object Specifications As Modules **3**

6 Specification, validation and verification: the development process **3**

A The CafeObj Requirements Model **6**
A.1 ListNats Module 6
A.2 Vote Module 8
A.3 StackVotes Module 10
A.4 Parcel Module 12
A.5 ParcelStack Module 15
A.6 ArrayParcels Module 18
A.7 AllParcels Module 21
A.8 VoteTabulation Module 24

[1]email: pgibson@cs.nuim.ie
[2]*E-voting: a safety critical system*, Margaret McGaley[10].

1 Introducing the PRSTV e-voting problem

The Single Transferable Vote (STV) is an example of a class of complex algorithms for vote counting that is found in many democratic systems around the world; at institutional, regional, national and international levels. Such algorithms have drawn interest in the formal methods community [12].

The specific variation of the system in Ireland is known as PR-STV (Proportional Representation Single Transferable Vote) and it is one of the most complex algorithms (if not the most complex) to specify and implement.

We review the formal specification of PRSTV as part of an experiment in comparing formal requirements models with informal requirements models with respect to their amenability to validation. In this experiment, part of the legal document that specifies the transfer of surplus rules for the Irish electoral system was employed as the informal specification[3] The formal requirements model with which it was compared waswritten in CafeObj and is reviewed in this technical report.

2 The role of requirements models

Requirements models have three distinct roles — they are the principle media of communication between clients and requirements engineers, they are the only model upon which rigorous and automated analysis can be carried out before development begins, and they are the structural foundation upon which design and implementation depend. A major part of building requirements is the modelling of the system to be developed (or updated) together with the system environment. These models are, of course, abstractions of the real world and their operational semantics can be executed to provide a simulation model for validation: to show that the behaviour specified actually corresponds to what exists or what is required. The models also have to be verified to show their logical consistency. A formal object oriented method facilitates incremental development, where the integration of simulation (through animation), theorem proving and model checking increases confidence in model correctness.

In our approach, we emphasise the need for client-oriented models — if the client cannot understand the requirements then validation cannot be done correctly and the rest of the development process is compromised. When in doubt, the best rule is to let the client's understanding of their needs provide the underlying structure of the requirements model.

3 The Object-Based Struture

In general, we advocate an object oriented[4] approach to structuring our requirements models. Object oriented methods encompass a set of techniques which have been, and will continue to be, applied in the successful production of complex software systems [2, 3, 1, 11]. The methods are based on the simple mathematical models of abstraction, classification, refinement and polymorphism. Central to the success of object oriented techniques is the support they offer to re-use at all levels of abstraction. Re-use and structure are just as important during requirements capture as during implementation.

[3] see http:www.irishstatutebook.ie1923_12.html

[4] Please note that in this study, for simplicity, we adopt an *object-based* strategy which focuses on composing objects and does not identify any subclassing relationships.

Structure is fundamental to all stages of system development: it provides the framework upon which already developed parts of a system can be re-used. Structured analysis and requirements capture methods have been successfully applied in many different problem domains during the last twenty years[4, 5, 6]. It is clear that there is a symbiotic relationship between structure and re-use: *classification* facilitates re-use of abstractions and relations between abstract behaviours, *composition* facilitates re-use of concrete behaviour, *refinement* facilitates re-use of verification and validation, *configuration* facilitates re-use of composition mechanisms. The key to building good requirements models is to model understanding as structure and to provide facility for structural re-use.

The informal, natural language description of the vote counting requirements are — by necessity — algorithimic in nature. Furthermore, the algorithm is naturally structured as functional units. Our goal was to reproduce this structure (as well as possible) in the requirements modelling process. This would aid the validation of the requirements model — we were mostly concerned that our requirements model was complete and consistent with respect to what the e-voting system was supposed to do. This requirements model could then be refined and restructured to aid: (semi-)automated verification of properties, and/or (semi-)automated generation of efficient code.

4 An Introduction to Algebraic Specification and CafeObj

A formal specification was developed using CafeOBJ. CafeOBJ is well supported by freely available tools, and it is claimed that it is good for working at the requirements and design levels of abstraction, and it maps well onto an Object Oriented implementation which is the paradigm most familiar to the students.

5 The Object Specifications As Modules

We follow the object-based algebraic style of specification as reported in [7, 8]:

- a module-type corresponds to a class
- an object is a value of a particular type, corresponding to a state
- operations can be used to construct new objects, change the state of an object or access some value of the state of the object
- a special invariant operation defines the sfae states of each object
- the operations define the object's interface
- the equations define the meaning of the operations
- the specification is executed through applying the operations as re-write rules

6 Specification, validation and verification: the development process

The system was developed in a bottom-up fashion:

- (1a) A List of Natural Numbers was used to represent a vote.

- (1b) A Vote was a list of natural numbers constrained in such a way as to guarantee vote validity
- (2a) A Stack of Votes was used to represent a pile of votes
- (2b) A Parcel was a Stack of Votes constrained in such a way as to guarantee a valid pile
- (3a) An Array of Parcels was used to represent all the piles of votes in the vote counting process
- (3b) AllParcels was used to constrain the array of parcels in order to guarantee a valid count state
- (4a) VoteTabulation was the model used to represent the vote counting as it goes through a number of rounds
- (4b) VoteResult models a VoteTabulation where constraints are used to guarantee a valid vote from round to round.

Please note the *pair-wise* increments of development — where each representation of a part of the real-world requirements (part (a)) is followed by a more rigorous representation (part b) where additional constraints are used to guarantee invariant (safety) properties.

For each of these 4 model pairs, the CafeObj re-write tool was used to validate very simple (voting) scenarios: limited usually to only a handful of candidates and votes. These were chosen in order to ensure that every possible conditional branch in the algorithmic behaviour was taken at least once. Furthermore, they increase our confidence that each module behaves as required.

The operations were proven to respect the invariants: so that if an invariant is true before an operation is applied then the invariant is guaranteed to be true after the operation has executed. This was usually done using structural induction over the internal data structures as defined by the constructor operators. We also proved that the initial state of the system (components) respected the invariant.

In order to more exhaustively test our model, we chose to rapidly prototype executable code (in Java) for each of the modules. This facilitated a much more comprehensive set of more-realistic scenarios being tested. As the process of going from the abstract algebraic specification of composed classes and objects to a concrete implementation was straightforward and one with which we had much experience, we were reassured that the original specification of requirements was correct through automated testing of the implementation which we derived from it.

Acknowledgements

Many thanks to Margaret McGaley and Deirdre Carew who were the main motivation behind me carrying out this case study; and writing up this technical report.

References

[1] G. Booch. *Object oriented design with applications.* Benjamin Cummings, 1991.
[2] P. Coad and E. Yourdon. *Object oriented analysis.* Prentice-Hall (Yourdon Press), 1990.
[3] P. Coad and E. Yourdon. *Object oriented design.* Prentice-Hall (Yourdon Press), 1990.
[4] L. Constantine. Beyond the madness of methods: System structure methods and converging design. In *Software Development 1989.* Miller-Freeman, 1989.
[5] G. Cutts. *Structured system analysis and design method.* Blackwell Scientific Publishers, 1991.

[6] T. DeMarco. *Structured analysis and system specification.* Prentice-Hall, 1979.

[7] J. Gibson. *Formal Object Oriented Development of Software Systems Using LOTOS.* Thesis csm-114, Stirling University, Aug. 1993.

[8] P. Gibson, G. Hamilton, and D. Méry. Requirements integration problems in telephone feature development. In A. G. Keijiro Araki and K. Taguchi, editors, *Integrated Formal Methods conference, IFM99*, pages 129–151. Springer Verlag, 1999.

[9] J. Goguen, T. Winkler, J. Meseguer, K. Futatsugi, and J.-P. Jouannaud. Introducing OBJ. In J. Goguen, editor, *Applications of Algebraic Specification using OBJ.* Cambridge, 1993.

[10] M. McGaley. Electronic voting: A safety critical system, 2003.

[11] B. Meyer. Re-usability: the case for object oriented design. *IEE Software Engineering*, Mar. 1987.

[12] M. Poppleton. The single transferable voting system: Functional decomposition in formal specifications. In G. O'Regan and S. Flynn, editors, *1st Irish Workshop on Formal Methods*, Dublin, Ireland, July 1997. Irish Formal Methods Special Interest Group (IFMSIG), Springer Verlag. http://ewic.springer.co.uk/.

A The CafeObj Requirements Model

A.1 ListNats Module

```
--> FILE: ListNats.mod
--> CONTENTS: ADT specification of  a list of Nats
--> PROJECT: E-Voting
--> AUTHOR: J Paul Gibson

-- to select the lastly loaded module automatically
    set auto context on

-- to show parse trees
-- set verbose on

--> ********************************************************
--> defining the ListNats module by instantiating List with Nat
-->
--> extended with a subtraction operator
--> ********************************************************
-->
mod! ListNats
{
  protecting (NAT)
  protecting (BOOL)

  [ListNats]

  op empty : -> ListNats
  op isempty : ListNats -> Bool
  op add : ListNats Nat -> ListNats
  op length : ListNats -> Nat
  op head : ListNats -> Nat
  op tail : ListNats -> ListNats
  op contains : ListNats Nat -> Bool

  op _-_: Nat Nat -> Nat

  var l : ListNats
  var e : Nat
  var n : Nat

  ceq e - n = 0 if (e == n) .
  ceq e - n = 1 + (e - (n + 1) ) if ( e > n ) .

  eq head (add(empty,e)) = e .
  eq head (add(l,e)) = head (l) .
```

```
  eq tail (add(empty,e)) = empty .
  eq tail (add(l,e)) = add( tail(l), e) .

  eq isempty (empty) = true .
  eq isempty (add(l,e)) = false .

  eq length(empty) = 0 .
  eq length(add(l,e)) = 1 + length(l) .

  eq contains (empty, n) = false .
  ceq contains (add(l,e), n) = true if (e == n) .
  ceq contains (add(l,e), n) = contains (l,n) if (not(e == n)) .
}
-->
--> *************************
--> testing the ListNats module
--> *************************
-->
open ListNats
--> test ListNats ops
-->
red isempty(empty) .
-->
red isempty(add(l,1)) .
-->
red length(empty) .
-->
red length(add(empty,1)) .
-->
red head (add(add(empty,1),2)) .
-->
red tail (add(add(empty,1),2)) .
-->
red contains( add((add(add(empty,1),2)), 4),  3) .
-->
red contains( add((add(add(empty,1),2)), 4),  4) .
-->
red contains( add((add(add(empty,1),2)), 4),  1) .
-->
red 7 - 2 .
close
```

A.2 Vote Module

```
--> FILE: Vote.mod
--> CONTENTS: ADT specification of  a Vote as a specialisation of a list of Nats
--> PROJECT: E-Voting
--> AUTHOR: J Paul Gibson

--> to select the lastly loaded module automatically
set auto context on
--> to show parse trees
set verbose on

--> *************************
--> defining the Vote module
--> *************************

mod! Vote {

  [Vote]
  protecting(NAT)
  protecting(ListNats)

  op AVote : ListNats  -> Vote
  op invariant : Vote -> Bool

  op Pref2Cand : Vote Nat -> Nat
  op Cand2Pref : Vote Nat -> Nat
  op CandNoPref : Vote Nat -> Bool
  op MaxCand : Vote -> Nat
  op MaxPref : Vote -> Nat

  op maximum : Nat Nat -> Nat

  var v : Vote
  var ln : ListNats
  var n1 : Nat
  var n2 : Nat

  ceq maximum (n1 ,n2) = n1 if n1 > n2 .
  ceq maximum (n1 ,n2) = n2 if n2 >= n1 .

  ceq invariant ( AVote(ln) ) = true if isempty(ln) .
  ceq invariant ( AVote(ln) ) = false if contains(tail(ln), head(ln) ) .
  ceq invariant ( AVote(ln) ) = false if ( not (invariant(AVote(tail(ln))))) .
   eq invariant ( AVote(ln) ) = true .

  ceq MaxCand ( AVote (ln) ) = head (ln) if ( (length(ln)) == 1) .
   eq MaxCand ( AVote (ln) ) = maximum ( head(ln) , MaxCand (AVote(tail (ln)) )  ) .
```

```
  eq MaxPref (AVote (ln) ) = length (ln) .

  ceq CandNoPref( AVote(ln), n1 ) = true if isempty(ln) .
  ceq CandNoPref( AVote(ln), n1 ) = false if (head (ln) == n1) .
   eq CandNoPref( AVote(ln), n1 ) = CandNoPref (AVote(tail(ln)), n1) .

  ceq Cand2Pref ( AVote(ln), n1 ) = 0 if CandNoPref(AVote(ln), n1) .
  ceq Cand2Pref ( AVote(ln), n1 ) = 1 if (head (ln) == n1) .
  ceq Cand2Pref ( AVote(ln), n1 ) = 1 +  Cand2Pref(AVote(tail(ln)), n1) if (not(head (ln) == n1)) .


  ceq Pref2Cand ( v, n1 ) = 0 if n1 < 1 .
  ceq Pref2Cand ( v, n1 ) = 0 if n1 > MaxPref (v) .
  ceq Pref2Cand ( AVote(ln), n1 ) = 0 if isempty(ln) .
  ceq Pref2Cand ( AVote(ln), n1 ) = head(ln) if n1 == 1 .
   eq Pref2Cand ( AVote(ln), n1 ) = Pref2Cand( AVote(tail(ln)), n1 - 1) .

}
--> *************************
--> testing the Vote module
--> *************************

open Vote .

--> test MaxCand

let ln1 = add(add(add(add(add(add( empty, 1), 4), 2),4),6),3) .
let ln2 = add(add(add(add(add(add( empty, 1), 4), 2),5),6),3) .
let v1 = AVote (ln1) .
let v2 = AVote (ln2) .
red MaxCand ( v1 ) .
red MaxPref ( v1 ) .

red Cand2Pref(v1, 1) .
red Cand2Pref(v1, 2) .
red Cand2Pref(v1, 6) .

red Pref2Cand(v1, 3) .
red Pref2Cand(v1, 5) .
red Pref2Cand(v1, 7) .

red invariant (v1) .
red invariant (v2) .
close
```

A.3 StackVotes Module

```
-- FILE: StackVotes.mod
-- CONTENTS: ADT specification of a stack of votes to be used as a Parcel
--> PROJECT: E-Voting
--> AUTHOR: J Paul Gibson

--> to select the lastly loaded module automatically
set auto context on
--> to show parse trees
set verbose on

--> ********************************************************
--> defining the StackVotes module as a  Vote Stack (LIFO)
--> extra operation for removing topn n elements of Vote Stack
--> ********************************************************

mod! StackVotes {

  [StackVotes]
  protecting(Vote)
  protecting(ListNats)
  protecting(NAT)

  op emptySV :  -> StackVotes

  op isempty : StackVotes -> Bool
  op size : StackVotes -> Nat
  op head : StackVotes -> Vote
  op topVotes : StackVotes Nat -> StackVotes
  op bottomVotes : StackVotes Nat -> StackVotes

  op push : StackVotes Vote -> StackVotes
  op pop : StackVotes -> StackVotes

  var SV : StackVotes
  var v1 : Vote
  var v2 : Vote
  var n : Nat

  eq isempty (emptySV) = true .
  eq isempty (push(SV,v1)) = false .

  eq size(emptySV) = 0 .
  eq size(push(SV,v1)) = 1 + size(SV) .

  eq head (push(SV,v1)) = v1 .
```

```
  eq pop (emptySV) = emptySV .
  eq pop(push(SV,v1)) = SV .

   eq bottomVotes (emptySV, n ) = emptySV .
  ceq bottomVotes (push(SV,v1), n ) = SV if (n == 1) .
  ceq bottomVotes (SV, n ) = SV if (n < 1) .
  ceq bottomVotes (push(SV,v1), n ) = bottomVotes(SV, (n - 1) ) if not(n == 1) .

  eq topVotes (emptySV, n) = emptySV .
  ceq topVotes (push(SV,v1), n ) = push(emptySV, v1) if (n == 1) .
  ceq topVotes (SV, n ) = SV if (n < 1) .
  ceq topVotes (push(SV,v1), n ) = push (topVotes(SV, (n - 1) ), v1) if not(n == 1) .
}

--> *************************
--> testing the StackVotes module
--> *************************

open StackVotes .

let ln1 = add(add(add(add(add( empty, 1), 4), 2),4),6) .
let ln2 = add(add(add(add(add(add( empty, 2), 4), 2),5),6),3) .
let ln3 = add(add(add( empty, 3), 4), 2) .
let ln4 = add( empty, 4) .

let v1 = AVote (ln1) .
let v2 = AVote (ln2) .
let v3 = AVote (ln3) .
let v4 = AVote (ln4) .

let SV1 = push(push (push(push (emptySV, v1), v2), v3),v4) .

reduce size (SV1) .
reduce isempty (SV1) .
reduce head (SV1) .
reduce pop (SV1) .

reduce bottomVotes (SV1, 1) .
reduce topVotes (SV1, 1) .

reduce bottomVotes (SV1, 2) .
reduce topVotes (SV1, 2) .

reduce bottomVotes (SV1, 3) .
reduce topVotes (SV1, 3) .

close
```

A.4 Parcel Module

```
-- FILE: Parcel.mod
-- CONTENTS: ADT specification of a Parcel as a constrained type of Stack of Votes
--> PROJECT: E-Voting
--> AUTHOR: J Paul Gibson

--> to select the lastly loaded module automatically
set auto context on
--> to show parse trees
set verbose on

--> ************************************************************
--> defining the Parcel module as a  constrained Vote Stack
--> where the number of candidates is the constraint parameter
--> and all Votes in Stack must be 'safe' ie have a true invariant
--> ************************************************************

mod! Parcel {

  [Parcel]
  protecting(StackVotes)
  protecting(Vote)
  protecting(ListNats)
  protecting(NAT)

  op AParcel : StackVotes Nat -> Parcel

  op invariant : Parcel -> Bool
  op head : Parcel -> Vote
  op tail : Parcel -> Parcel
  op isempty : Parcel -> Bool
  op numCands : Parcel -> Nat
  op size : Parcel -> Nat

  op topVotes : Parcel Nat -> Parcel
  op bottomVotes : Parcel Nat -> Parcel

  op addP : Parcel Parcel -> Parcel
  op addV : Parcel Vote -> Parcel

  var SV : StackVotes
  var n  : Nat
  var count : Nat
  var v : Vote
  var p1 : Parcel
  var p2 : Parcel
```

```
  eq addV (AParcel(SV,n) , v) =  AParcel(push(SV,v), n) .
  eq size (AParcel(SV,n) ) = size(SV) .

  ceq addP (p1, p2) = p1 if isempty(p2) .
  ceq addP (p1, p2) = addP( addV (  p1, head(p2) ), tail(p2) ) if numCands(p1) == numCands(p2).

  eq isempty (AParcel(SV,n)) = isempty (SV) .
  eq head (AParcel(SV,n)) = head(SV) .
  eq tail (AParcel (SV, n)) = AParcel( pop (SV) , n) .
  eq numCands (AParcel(SV,n)) = n .

  eq topVotes (AParcel(SV,n), count) = AParcel(topVotes(SV,count), n) .
  eq bottomVotes (AParcel(SV,n), count) = AParcel(bottomVotes(SV,count), n) .

  ceq invariant (p1) = true if isempty(p1) .
  ceq invariant (p1) = false if not(invariant(head(p1))) .
  ceq invariant (p1) = false if MaxCand(head(p1)) > numCands(p1) .
  ceq invariant (p1) = false if not(invariant(tail(p1))) .
  ceq invariant (p1) = true if invariant(tail(p1)) and invariant(head(p1)) .
}

--> *************************
--> testing the Parcel module
--> *************************

open Parcel .

let ln1 = add(add(add(add(add( empty, 1), 4), 2),5),6) .
let ln2 = add(add(add(add(add(add( empty, 1), 4), 2),4),6),3) .
let ln3 = add(add(add( empty, 3), 4), 7) .
let ln4 = add( empty, 4) .

let v1 = AVote (ln1) .
let v2 = AVote (ln2) .
let v3 = AVote (ln3) .
let v4 = AVote (ln4) .

let SV1 = push (push (push(push (emptySV, v1), v2), v3),v4) .
let SV2 = push (push (emptySV, v1), v3) .
let SV3 = push (push (emptySV, v1), v4) .

let P1 = AParcel(SV1, 8) .
let P2 = AParcel (SV2, 7) .
let P3 = AParcel (SV3, 3) .
let P4 = AParcel (SV3, 6) .
let P5 = addP (P4, P4) .
```

```
red invariant(P1) .
red invariant(P2) .
red invariant(P3) .
red invariant(P4) .
red P5 .

red size (P5) .
close
```

A.5 ParcelStack Module

```
-- FILE: ParcelStack.mod
-- CONTENTS: ADT specification of a stack of parcels
-- AUTHOR: J Paul Gibson

--> to select the lastly loaded module automatically
set auto context on
--> to show parse trees
set verbose on

--> ******************************************************
--> defining the ParcelStack  module as a  Stack of Parcels
--> ******************************************************

mod! ParcelStack {

  [ParcelStack]
  protecting(Parcel)
  protecting(Vote)
  protecting(ListNats)
  protecting(NAT)

  op emptyPS :  Nat -> ParcelStack

  op isempty : ParcelStack -> Bool
  op roundOut : ParcelStack -> Nat
  op totallength : ParcelStack -> Nat
  op numberofparcels : ParcelStack -> Nat

  op lengthofParcelfromTop : ParcelStack Nat -> Nat

  op lengthofParcelfromBottom : ParcelStack Nat -> Nat

  op head : ParcelStack -> Parcel
  op push : ParcelStack Parcel -> ParcelStack
  op pop : ParcelStack -> ParcelStack

  op headVote : ParcelStack -> Vote
  op pushVote : ParcelStack Vote -> ParcelStack
  op popVote : ParcelStack -> ParcelStack

  var PS : ParcelStack
  var p1 : Parcel
  var p2 : Parcel
  var v1 : Vote
  var round : Nat
```

```
  var parcelnum : Nat

  eq roundOut (emptyPS(round)) = round .
  eq roundOut (push(PS,p1)) = roundOut(PS) .

  eq isempty (emptyPS(round)) = true .
  eq isempty (push(PS,p1)) = false .

  eq head (push(PS,p1)) = p1 .
  eq pop (push(PS,p1)) = PS .

  eq headVote (push(PS,p1)) = head(p1)  .
  eq popVote (push(PS,p1)) = push(PS, pop(p1) ) .
  eq pushVote ((push(PS,p1)), v1) = push(PS, push(p1,v1)) .

  eq totallength (emptyPS(round)) = 0 .
  eq totallength (push(PS,p1)) = (totallength(PS) + length(p1)) .

  eq numberofparcels (emptyPS(round)) = 0 .
  eq numberofparcels (push(PS,p1)) = 1 + numberofparcels (PS) .

  eq lengthofParcelfromTop (emptyPS(round), parcelnum) = 0 .
  ceq lengthofParcelfromTop (push(PS,p1), parcelnum) =  length(p1) if parcelnum == 1 .
  ceq lengthofParcelfromTop (push(PS,p1), parcelnum) =
      lengthofParcelfromTop(PS, (parcelnum - 1) ) if not (parcelnum == 1) .

  eq lengthofParcelfromBottom (PS, parcelnum) =
     lengthofParcelfromTop (PS, ( 1 + (numberofparcels(PS) - parcelnum) ) ) .
}

--> *******************************
--> testing the ParcelStack  module
--> *******************************

open ParcelStack .

let vote1a = empty(3) .
let vote1b = addP(vote1a, 1) .
let vote1a = addP(vote1b, 3) .

let vote2a = empty(3) .
let vote2b = addP(vote2a, 2) .
let vote2a = addP(vote2b, 1) .

let vote3a = empty(3) .
let vote3b = addP(vote3a, 3) .
let vote3a = addP(vote3b, 2) .
```

```
let parcel1a = emptyP(3) .
let parcel1b = push( parcel1a, vote1a) .
let parcel1a = push( parcel1b, vote2a) .
let parcel1b = push( parcel1a, vote3a) .

--> reduce length (parcel1a) .
reduce length (parcel1a) .

--> reduce length (parcel1b) .
reduce length (parcel1b) .

let parcelstack1a = emptyPS(1) .

let parcelstack1b = push (parcelstack1a, parcel1a) .
let parcelstack1ab = push (parcelstack1b, parcel1a) .

--> reduce totallength (parcelstack1b) .
reduce totallength (parcelstack1b) .

--> reduce totallength (parcelstack1ab) .
reduce totallength (parcelstack1ab) .

--> reduce numberofparcels (parcelstack1ab) .
reduce numberofparcels (parcelstack1ab) .

--> reduce lengthofParcelfromBottom (parcelstack1ab, 1) .
reduce lengthofParcelfromBottom (parcelstack1ab, 1) .

--> reduce lengthofParcelfromTop (parcelstack1ab, 1) .
reduce lengthofParcelfromTop (parcelstack1ab, 1) .

--> reduce headVote (parcelstack1ab) .
reduce headVote (parcelstack1ab) .

close
```

A.6 ArrayParcels Module

```
-- FILE: ArrayParcels.mod
-- CONTENTS: ADT specification of an Array of Parcels (Stacks of votes)
--> PROJECT: E-Voting
--> AUTHOR: J Paul Gibson

--> to select the lastly loaded module automatically
set auto context on
--> to show parse trees
set verbose on

--> ************************************************************
--> defining an 'array' of Parcels for storing candidate votes
--> ************************************************************

mod! ArrayParcels {

  [ArrayParcels]
  protecting(ParcelStack)
  protecting(Parcel)
  protecting(StackVotes)
  protecting(Vote)
  protecting(ListNats)
  protecting(NAT)

  op emptyAPs : -> ArrayParcels
  op addP : ArrayParcels Parcel -> ArrayParcels

  op addVtoC : ArrayParcels Vote Nat -> ArrayParcels
  op popVfromC : ArrayParcels Nat -> ArrayParcels
  op headVofC : ArrayParcels Nat -> Vote

  op addPtoC : ArrayParcels Parcel Nat -> ArrayParcels

  op numCands : ArrayParcels -> Nat

  op size : ArrayParcels Nat -> Nat

  var SV : StackVotes
  var n  : Nat
  var count : Nat
  var v : Vote
  var p1 : Parcel
  var p2 : Parcel
  var APs : ArrayParcels

  eq numCands(emptyAPs) = 0 .
```

```
  eq numCands(addP(APs, p1)) = 1 + numCands(APs) .

  eq addVtoC(emptyAPs, v, n) = emptyAPs .
  ceq addVtoC(APs, v, n) = APs if n < 1 .
  ceq addVtoC( (addP(APs, p1)) , v, n) = addP( APs, addV(p1,v) ) if n == 1 .
  ceq addVtoC( (addP(APs, p1)) , v, n) = addP( addVtoC (APs, v, n - 1), p1 ) if n > 1 .

  ceq size( (addP(APs, p1)) , n) =   size(p1) if n == 1 .
  ceq size( (addP(APs, p1)) , n) =   size( APs  , n - 1 ) if n > 1 .


  ceq headVofC( (addP(APs, p1)) , n) =   head(p1) if n == 1 .
  ceq headVofC( (addP(APs, p1)) , n) =   headVofC( APs  , n - 1 ) if n > 1 .

  ceq popVfromC( (addP(APs, p1)) , n) =   addP (APs, tail(p1)) if n == 1 .
  ceq popVfromC( (addP(APs, p1)) , n) =   addP (popVfromC(APs, n - 1 ), p1 ) if n > 1 .

  eq addPtoC(emptyAPs, p1, n) = emptyAPs .
  ceq addPtoC(APs, p1, n) = APs if n < 1 .
  ceq addPtoC( (addP(APs, p1)) , p2, n) = addP( APs, addP(p1,p2) ) if n == 1 .
  ceq addPtoC( (addP(APs, p1)) , p2, n) = addP( addPtoC (APs, p2, n - 1), p1 ) if n > 1 .
}

--> ******************************
--> testing the ArrayParcels module
--> ******************************

open ArrayParcels .

let ln1 = add(add(add(add(add( empty, 1), 4), 2),5),6) .
let ln2 = add(add(add(add(add(add( empty, 1), 4), 2),4),6),3) .
let ln3 = add(add(add( empty, 3), 4), 7) .
let ln4 = add( empty, 3) .

let v1 = AVote (ln1) .
let v2 = AVote (ln2) .
let v3 = AVote (ln3) .
let v4 = AVote (ln4) .

let P1 = AParcel(emptySV, 8) .
let P2 = AParcel (emptySV, 8) .

let APs2 = addP (addP(emptyAPs, P1), P2) .

let APs2v1 = addVtoC( APs2, v1, 1) .
let APs2v2 = addVtoC(APs2v1, v2, 1) .
let APs2v3 = addVtoC(APs2v2, v3, 2) .
```

```
let APs2v4 = addVtoC(APs2v3, v4, 2) .

red headVofC(APs2v4, 1) .
red popVfromC(APs2v4, 1) .
red headVofC(APs2v4, 2) .
red popVfromC(APs2v4, 2) .

red size (APs2v4, 2) .
close
```

A.7 AllParcels Module

```
-- FILE: AllParcels.mod
-- CONTENTS: ADT specification of all Vote Parcels involved in vote tabulation process
--> PROJECT: E-Voting
--> AUTHOR: J Paul Gibson

--> to select the lastly loaded module automatically
set auto context on
--> to show parse trees
set verbose on

--> ************************************************************
--> The data structure for all vote parcels during tabulation ---
--> 1) An array of n parcels for the n candidates (initially empty)
--> 2) A parcel for the nontransferrable votes
--> 3) A parcel for the votes currently being processed
--> ************************************************************

mod! AllParcels {

  [AllParcels]
  protecting (ArrayParcels)
  protecting(ParcelStack)
  protecting(Parcel)
  protecting(StackVotes)
  protecting(Vote)
  protecting(ListNats)
  protecting(NAT)

  op makeAllParcels : Nat -> AllParcels
  op makeArrayParcels : Nat Nat -> ArrayParcels
  op anAllParcels : ArrayParcels Parcel Parcel -> AllParcels

  op introduceVote : AllParcels Vote -> AllParcels

  op countVotes : AllParcels Nat -> Nat

  var c : Nat
  var n : Nat
  var v : Vote
  var APs : ArrayParcels
  var nontr : Parcel
  var pr : Parcel
  var cand : Nat

  ceq makeArrayParcels (c, n) = emptyAPs if n < 1 .
  ceq makeArrayParcels (c, n) = addP (emptyAPs, AParcel(emptySV,c) ) if n == 1 .
```

```
  ceq makeArrayParcels (c, n) = addP (makeArrayParcels(c, n - 1 ), AParcel(emptySV,c) ) if n > 1 .

  eq makeAllParcels (n) = anAllParcels(makeArrayParcels(n, n), AParcel(emptySV, n), AParcel(emptySV, n) ).


  ceq introduceVote (anAllParcels(APs, nontr, pr), v) = anAllParcels(APs, addV(nontr,v), pr)
                                                        if Pref2Cand(v, 1) == 0 .

  ceq introduceVote (anAllParcels(APs, nontr, pr), v) = anAllParcels(APs, addV(nontr,v), pr)
                                                        if Pref2Cand(v, 1) > numCands(APs) .

  ceq introduceVote (anAllParcels(APs, nontr, pr), v) = anAllParcels( addVtoC(APs, v, Pref2Cand(v, 1)), nontr
                                                        if Pref2Cand(v, 1) > 0 .

  ceq countVotes (anAllParcels(APs, nontr, pr), cand) =   size(APs, cand) if isempty(pr) .
}

--> ******************************
--> testing the AllParcels module
--> ******************************

open AllParcels .

let ln1 = add(add(add(add(add( empty, 1), 4), 2),5),6) .
let ln2 = add(add(add(add(add(add( empty, 1), 4), 2),4),6),3) .
let ln3 = add(add(add( empty, 3), 4), 7) .
let ln4 = add( empty, 3) .

let ln5 = add (empty, 10) .

let v1 = AVote (ln1) .
let v2 = AVote (ln2) .
let v3 = AVote (ln3) .
let v4 = AVote (ln4) .
let v5 = AVote (ln5) .

let C3 =  introduceVote(introduceVote(makeAllParcels(3), v1 ), v2 ) .

red C3 .

let C4 = introduceVote(introduceVote(C3, v3 ), v4 ) .

red C4 .

let C5 = introduceVote (C4, v5) .

red countVotes (C5, 1)  .
```

```
red countVotes (C5, 2)  .
red countVotes (C5, 3)  .
close
```

A.8 VoteTabulation Module

```
-- FILE: VoteTabulation.mod
-- CONTENTS: ADT specification of PRSTV tabulation process
--> PROJECT: E-Voting
--> AUTHOR: J Paul Gibson

--> to select the lastly loaded module automatically
set auto context on
--> to show parse trees
set verbose on

--> ******************************************************************************
--> Using AllParcels data structure for all vote parcels during tabulation ---
--> 1) An array of n parcels for the n candidates (initially empty)
--> 2) A parcel for the nontransferrable votes
--> 3) A parcel for the votes currently being processed
--> Together with the number of votes that a candidate needs to be deemed elected
--> (one greater than the threshold specified) and the current round
--> It assumes that the votes have been correctly allocated for the 1st round
--> in the AllParcels data structure
--> ******************************************************************************

mod! VoteTabulation {

  [VoteTabulation]
  protecting (AllParcels)
  protecting (ArrayParcels)
  protecting(Parcel)
  protecting(StackVotes)
  protecting(Vote)
  protecting(ListNats)
  protecting(NAT)

  op StartCounting : AllParcels Nat -> VoteTabulation
  op AVoteTabulation : AllParcels Nat Nat ->  VoteTabulation
  op electedRound : VoteTabulation Nat -> Nat

  op currentRound : VoteTabulation -> Nat
  op voteThreshold : VoteTabulation -> Nat

  var APs : AllParcels
  var thr : Nat
  var round : Nat
  var cand : Nat

  eq StartCounting (APs, thr) = AVoteTabulation(APs, thr, 1) .
```

```
  eq currentRound (AVoteTabulation(APs, thr, round)) = round .
  eq voteThreshold(AVoteTabulation(APs, thr, round))  = thr  .

  ceq electedRound ( AVoteTabulation(APs, thr, round), cand ) = round if countVotes(APs, cand) > thr .
  ceq electedRound ( AVoteTabulation(APs, thr, round), cand ) = 0 if countVotes(APs, cand) <= thr .

}

--> ******************************
--> testing the VoteTabulation module
--> ******************************

open VoteTabulation .

let ln1 = add(add(add(add(add( empty, 1), 4), 2),5),6) .
let ln2 = add(add(add(add(add(add( empty, 1), 4), 2),4),6),3) .
let ln3 = add(add(add( empty, 3), 4), 7) .
let ln4 = add( empty, 3) .

let ln5 = add (empty, 10) .

let v1 = AVote (ln1) .
let v2 = AVote (ln2) .
let v3 = AVote (ln3) .
let v4 = AVote (ln4) .
let v5 = AVote (ln5) .

let C3 =  introduceVote(introduceVote(makeAllParcels(3), v1 ), v2 ) .

let C4 = introduceVote(introduceVote(C3, v3 ), v4 ) .

let C5 = introduceVote (C4, v5) .

red C5 .

let VT = StartCounting (C5, 1) .

red VT .

let ER1 = electedRound(VT, 1) .
red ER1 .

let ER2 = electedRound(VT, 2) .
red ER2 .

let ER3 = electedRound(VT, 3) .
red ER3 .
```

```
close
```