```
/
*****************************************************************************************
****
 *   DESCRIPTION :
 *
 * - There is one session of one meeting
 * - Members and authors of moitons are not represented
 * - Motions are not seconded
 * - Votes are abstracted in only their final outcome (adopted or not)
 * - Terminal Subsidiary Motions are instantiated
 * - Terminal Privileged Motions are instantiated
 * - SOme Terminal Incidentals are instantiated
 * - The effect of motions on the content of the
 *         motions they are applied to is abstracted (eg: amendements)
 * - The effect of motions on the status of the
 *         motions they are applied to is represented (eg: motions laid on the table)
 * - Focus is put on the motion to Amend
 *
 *
 *
*****************************************************************************************
**/
module model1

        /* *
         * An ordering over the Businesses of a sessionState and
         * over the sessionStates of a Session
         */
        //open util/ordering[SessionState] as sessionOrd

  /*****************
   * Base facts *
   ****************/
  pred BASE_FACTS{}

   fact linked_signatures{
                //there is a first and a last session state and
                //the session states are a doubly-chained list
                some SessionState => (
                        one first, last : SessionState |
                                no first.prev and no last.next
                                and (all ss : SessionState-(first) | one ss.prev ) //TODO are these 2 lines
useful when we have the reciprocity between next and prev?
                                        and      (all ss : SessionState-(last) | one ss.next ))

                //reciprocal relations
                (SessionState <: next) = ~ (SessionState <: prev)
        }
```

```
fact utility_facts{

	//by default traces are sessionprefixes and only dynamic mode is allowed
	//session_prefix and dynamic_mode_only

	//The error action induces the existance of an error flag
	( error[lastSessionState.prev, lastSessionState, Motion ] ) =>
		one ErrorFlag else no ErrorFlag
}

fact no_cycles {
	//a secondary motion cannot be inside a cycle of isAppliedTo or takesOver relations
	no sm : SecondaryMotion | sm in (sm.^takesOver + sm.^isAppliedTo)
	//a sessionState cannot be inside a cycle of next of prev relations
	no ss : SessionState | ss in (ss.^next + ss.^prev)
	//an atom cannot be inside a cycle of next of prev relations // TODO use allAux
	no atom : Business | atom in atom.^aux
//	no atom : OriginalMainMotion |	atom in atom.^aux
//	no atom : IncidentalMainMotion | atom in atom.^aux
	no atom : MainMotion |	atom in atom.^aux
//	no atom : BQAAMotion |	atom in atom.^aux
	no atom : DivisionOfQuestion |	atom in atom.^aux
	no atom : ConsiderationByParagraph | atom in atom.^aux
	no atom : CreateBlank |atom in atom.^aux
	no atom : PostponeIndefinitely | atom in atom.^aux
	no atom : AmendDebatable | atom in atom.^aux
	no atom : AmendUndebatable | atom in atom.^aux
	no atom : Commit | atom in atom.^aux
	no atom : Postpone | atom in atom.^aux
	no atom : LimitExtendDebate | atom in atom.^aux
	no atom : PreviousQuestion | atom in atom.^aux
	no atom : LayOnTheTable | atom in atom.^aux
	no atom : CallForOrdersOfDay | atom in atom.^aux
	no atom : RaiseQuestionOfPrivilege | atom in atom.^aux
	no atom : Recess | atom in atom.^aux
	no atom : Adjourn | atom in atom.^aux
	no atom : FixTimeAdjourn | atom in atom.^aux
}

fact auxiliary_atoms {

	//auxiliary atoms are either enabled or disabled
	enable_auxiliary_atoms or disable_auxiliary_atoms
	//(enable_auxiliary_atoms and not disable_auxiliary_atoms) or //unsatifiable sentences
	//(not enable_auxiliary_atoms and disable_auxiliary_atoms)

	//When auxiliary atoms are enabled : sessionState elements are main and
	//non- sessionState elements are auxiliary
	enable_auxiliary_atoms =>
```

```
		all elem : sessionStateElements |  isMain[elem] and
		all elem : nonSessionStateElements | isAuxiliary[elem]

	//If a relation comes out of /reaches a main atom the same relation comes out of/reaches its auxiliary atom;
	//exeption made for relations to Amend. In other words :
	//If a session state element but amend is related to something that is not amend than
	//the the auxiliaries of the atoms related to this element are exactly
	//the atoms related to the auxiliaries of this element
	//NOTE : case where a main atom is applyng to a sibling ? => no incidental applies to siblings in this model
all a : Amend | (
	(all elem : a.context
		| some (elem.dealsWith) =>
			(elem.dealsWith.allAux ) = (elem.allAux.dealsWith)
		else no elem.allAux.dealsWith)
	and

	(all elem :  a.context
		| some (elem.takesOver) =>(
			(some elem & Amend)=>
					( elem.takesOver.allAux = elem.yieldsTo.allAux.takesOver)
			else (
					(some elem.takesOver&Amend) =>
						( elem.takesOver.takesOver.allAux = elem.allAux.takesOver )
					else ( elem.takesOver.allAux = elem.allAux.takesOver )
				)

		)
		else no elem.allAux.takesOver)
		and
	(all elem :  a.context
		| no (elem.yieldsTo) =>
			no (elem.allAux.yieldsTo) )
	and
	(all elem :  a.context
		| some (elem.isAppliedTo) =>
			(elem.isAppliedTo.allAux) = (elem.allAux.isAppliedTo)
		else no elem.allAux.isAppliedTo)
	and
	(all elem :  a.context
		| some (elem.separatedQuestions) =>
			((elem.separatedQuestions).allAux ) = (elem.allAux.separatedQuestions)
		else no elem.allAux.separatedQuestions)
	and
```

```
                (all elem :  a.context
                                | some (elem.questionOfPrivilege) =>
                                        ((elem.questionOfPrivilege).allAux ) =
(elem.allAux.questionOfPrivilege)
                        else no elem.allAux.questionOfPrivilege)
                //NOTE: removed some code here because business links are now part of the state
                )
    }

   /**************************
    * General Predicates *
    **************************/
 pred GENERAL_PREDICATES{}

        pred whole_session {
                initialSession[firstSessionState]
                finalSession[lastSessionState]
        }

        pred session_prefix {
                initialSession[firstSessionState]
        }

        pred session_suffix {
                finalSession[lastSessionState]
        }

        pred reach_all_motions {
                no lastSessionState.toComeToOrder
   }

        pred encourage_move{
                all s : SessionState |
                        (some m1 : Motion | move_preconditions[s,m1]) =>(some m2 : Motion| move[s,
s.next, m2])
        }

        pred enable_movedOnce{
                //a motion belonging to the passed signature is moved once on top of another motion
                //§6 Motions that bring a question again before the assembly
                //Amend and RootMotions should always be movabe twice
                movedOnce[PostponeIndefinitely]
                movedOnce[Commit]
                movedOnce[Postpone]
                movedOnce[LimitExtendDebate]
                movedOnce[PreviousQuestion]
                movedOnce[LayOnTheTable]
                movedOnce[CallForOrdersOfDay]
                movedOnce[RaiseQuestionOfPrivilege]
```

```
		movedOnce[Recess]
		movedOnce[Adjourn]
		movedOnce[FixTimeAdjourn]
		movedOnce[ConsiderationByParagraph]
		movedOnce[CreateBlank]
		movedOnce[DivisionOfQuestion]



	}

	/**
	* Static and dynamic dependancies will apear
	* (static dependencies alone are not allowed)
	*/
	pred	dynamic_mode_only {

		//a sessionStateElement business is always part of some order of business**
		all b : Business | (b in sessionStateElements) => (some ss : SessionState | b in
ss.orderOfBusiness)

		//There should always exist at least one sessionState
		some SessionState

	}

	/***
	 * There are auxiliary atoms and non auxiliary atoms and
	 *	non auxiliary atoms point to auxiliary atoms.
	 *
	 * Not suited for actually enabling auxiliary atoms outside the model.
	 * Use "not disable_auxiliary_atoms"
    */
	pred enable_auxiliary_atoms {
		all atom : Business+Motion |
			(no allAux.atom /*and some atom.allAux*/) or (no atom.allAux and one
allAux.atom)
		(no a : Amend | hasTargets[a])=> no nonSessionStateElements
	}

	pred disable_auxiliary_atoms {
		//Duplicate code because of absence of overloading capabilities in alloy (but allAux
workarround!)
		all atom : Business+Motion|  no atom.allAux
		//there should exist only sessionState elements
		no nonSessionStateElements
	}
```

```
		/*************************

		 * MODELING CONCEPTS *

		 *************************/

pred MODELING_CONCEPTS {}

	one sig Aux {
		transparentAmends : set Amend,
		contexts : Amend -> ((Motion+Business)  -> (Motion+Business))
	}
	{
		//no motion should point to itself inside any amend context
		all a: Amend, m : Motion, b : Business |
			m not in m.(a.contexts) and b not in b.(a.contexts) and
			m not in (a.contexts).m and b not in (a.contexts).b
		//Amend is not contained inside its onwn context
		all a : Amend | no a.(a.contexts) and no (a.contexts).a

		//transparentAmends are the main amends who should have auxiliary worlds
//		all a : Amend | isAuxiliary[a] => no ( a & transparentAmends)
//		all a : Amend | not hasTargets[a] => no ( a & transparentAmends)

		all a : Amend |
			((not disable_auxiliary_atoms) and isMain[a] and hasTargets[a] )=>
				a in transparentAmends
			else no (a & transparentAmends)

		//contexts projected on Amends do not overlap
		all disj a1, a2 : Amend | no (a1.contexts)&(a2.contexts)
		//each element of an amend's context participates in exactly
		//one auxiliary relation inside the Aux.contexts relation
		let ta = transparentAmends |
			all a : Amend |
				let context = a.context | //getContext[a] |
					a in ta =>
						( all elem : context | one ((elem<:allAux) & (a.contexts)) )
					else ( no a.contexts )
		//Aux.contexts contains only auxiliary relations
		transparentAmends.contexts = allAux

		//contexts projected on some Amend contains this Amend's context
		//all a :Amend | a.contexts = ((a.context) <: allAux ) //TODO

	}

	/**
	* Exists in the purpose of showing there is an error
```

```
	*/
	sig ErrorFlag {}


		/*************************
		 * DOMAIN CONCEPTS *
		 *************************/
pred DOMAIN_CONCEPTS {}


	/***
	* A sessionState : the description of a session at a given 'time'.
	* Refer to :
	* §3 Call to order; Order of Business
	* §4 How a motion is brought before the Assembly
	* §5 Secondary Motions as an Underlying Concept (fundamental principle of parliamentary law)
	*/
	sig SessionState{
		//sets representing the state of motions
		immediatePendingQuestion : lone Motion, //set Motion, TODO : relax for property testing pupose ?
		disj toComeToOrder, pendingQuestions : set Motion,
		adopted : set Motion,
		//pendingQuestions2 : set Motion -> Motion,
		//sets of motions affected by a motion moved previously to this state
		laidOnTheTable : set Motion,
		underPreviousQuestionOrder : set Motion,
		underCommitOrPostponeIndefinitelyOrder : set Motion, //**should be separate sets
		//the businesses, **always at least one business to deal with
		orderOfBusiness : lone Business , //TODO : shall the order of business be an autonomous entity?
		//The current business is always part of the order of business of the sessionState;
		// there may be no current business**
		currentBusiness : lone orderOfBusiness,
		//the linked states
		next : lone SessionState,
		prev : lone SessionState
	}
	{	/* *All these constraints are to be considered inside >this< sessionState exclusively* */

		//no cycles in the sessionStates links
		this not in (next+prev)
		//If secondary motions exist in the pending questions set this must be relatively to a root motion//TODO useless? redundant?
```

```
		some (pendingQuestions & SecondaryMotion) => some (pendingQuestions &
RootMotion)
		//A secondary motion inside the pending stack of this sessionState only takes precedence
over
		// motions inside the latter stack
		pendingQuestions.takesOver in pendingQuestions
		//the ipq is one of the pending questions of the sessionState
		immediatePendingQuestion in pendingQuestions
		//If there are pendingQuestions then there is an ipq
//		some pendingQuestions  => one (immediatePendingQuestion) //TODO for cod =>
useful?
		//The ipq in the pending stack of this sessionState does not yield precedence inside the
former stack
		all i : immediatePendingQuestion |    no (i.yieldsTo & pendingQuestions)
		//If a secondary motion is applied to another secondary motion then all
		//the motions in the yieldsTo relation from this motion are applied to secondary motions
//		all sm1, sm2 : SecondaryMotion
//			| sm2 in sm1.isAppliedTo
//				=> (all m : sm1.^(yieldsTo) | m.isAppliedTo in
SecondaryMotion)//TODO find reference or remove !!
	//If a motion is applicable to a set of motions then they are consecutive amongst each other//
TODO find reference
	//(only one head and and only one tail) REMOVED : and consecutive to the former motion.
		all sm : Motion
			| some sm.isAppliedTo
				=>	(one head: (sm.isAppliedTo) |
				  (all other : (sm.isAppliedTo)|
					other in head.^(takesOver)+head))
		all sm : Motion
			| some sm.isAppliedTo
				=>	(one tail: (sm.isAppliedTo) |
				   tail.takesOver not in sm.isAppliedTo or tail in RootMotion
				)
		//The motions to be postponed definitely or indefinitely or committed are among the
pending
		//
(underCommitOrPostponeIndefinitelyOrder+underPreviousQuestionOrder+laidOnTheTable) in
pendingQuestions
	}

	/**
	* A Business
   * A business is dealt with by the means of a root motion
   * Refer to:
   * §3 Means by which business is brought before the assembly
   * §5 Classes of Motions
	*/
	 sig Business {
		isDealtWithBy : some RootMotion,
```

```
		,aux : set Business

	}
	{	//Unless there are separated questions or a question of privilege
		//Only at most one motion should deal with a Business
		(lone isDealtWithBy)or
		(some (isDealtWithBy&(DivisionOfQuestion.separatedQuestions)))or
		(some (isDealtWithBy&(RaiseQuestionOfPrivilege.questionOfPrivilege)))
	}

	/**
	* A motion
	* abstract -> a motion must have a type.
	* Refer to:
   * §5 Classes of Motions
   * §5 Secondary motion as an underlying concept
	*/
	abstract sig Motion {
		yieldsTo : set SecondaryMotion
//		,amendlessYieldsTo : set SecondaryMotion
	}
	{
		/* §12 Standard descriptive characteristics 1) */
		//no good
		//amendlessYieldsTo = yieldsTo - Amend + ((yieldsTo & Amend).@yieldsTo)
	}
	fact{
		//these relations are reciprocal
		//§5 Taking of precedence of one motion over another
		yieldsTo = ~takesOver
		//no good because there is actually no other constraint on the amendlessYieldsTo
		//appart from the fact itself that it is trying to constrain.
		//all m : Motion, a : Amend | some m.yieldsTo & a and some m.amendlessYieldsTo &
a.yieldsTo
   }

/*****************************************
**********
 * THE GENEREAL CATEGORIES OF
MOTIONS  *

************************************************/
pred
THE_GENEREAL_CATEGORIES_OF_MOTIONS {}

/*****
***************
 *
ROOTMOTIONS *
```

**********/

pred ROOTMOTIONS {}

```
	/**
	 * A motion that is at the root of a stack of pending questions
	 * Refer to :
    * §5 Order of precedence of motions
	 */
	 abstract sig RootMotion extends Motion{
		dealsWith : one Business //A root motion exists because it deals with a business**
	 }
	 {
		//a root motion in the pending questions of some sessionState must be dealing with some business
		//and this business must be a current one in that sessionState.**
		//all ss: SessionState | ((this in ss.pendingQuestions and no ss.specialOrder) => //NOTE : modified for cod
		//dealsWith in ss.currentBusiness )
		//**All RootMotions are amendable and debatable
		this in AmendableMotion
		this in DebatableMotion
	 }
	fact{
		//reciprocal relations
		dealsWith = ~isDealtWithBy
	}

	/**
	* A main motion
   * abstract -> a main motion has to have a class
   * §5 Classes of Motions
	*/
	sig MainMotion extends RootMotion {
		aux : set MainMotion
  }


	/**
	* A motion that can bring a question again
	* before the assembly
	*/
	/*sig BQAAMotion extends RootMotion {
		aux : set BQAAMotion
	}
  {}*/
```

```
/*********************************

        *  SECONDARY_MOTIONS     *

        *********************************/

        pred SECONDARY_MOTIONS {}


        /**
        *A Secondary Motion
        *abstract -> a sm has to belong to a class.
    *§5 Taking of precedence by one motion over another
        */
        abstract sig SecondaryMotion extends Motion{
                takesOver : lone Motion,
                isAppliedTo : set Motion
        }
        {
                //A secondary motion exists because it takes precedence over another motion**
                some takesOver
        }
        /**
     * An incidental motion (Secondary)
     *
     */
        abstract sig IncidentalMotion extends SecondaryMotion {}//TODO : abstract ?
        {
                //An incidental motion that is linked to no session state and that is not an auxiliary atom
                //is involved in no "applied to" relation** ; one inside has to be in at least one.
                this not in (sessionStateElements + sessionStateElements.allAux)=>
                        no isAppliedTo
                                else ( some isAppliedTo )
                //Incidental motions except DivisionOfTheAssembly yield to the privileged motions
                // and one subsidiary motion(LayOnTheTable)**
                // §6: conditions under which incidental motions take precedence over, or yield to, other
motions.
                yieldsTo in (PrivilegedMotion + LayOnTheTable )
    }
    fact{
        //A IncidentalMotion in the sessionState's pending stack is only applied to
                // a lower motion in the takesOver transitive closure or a 'sibling'
                // motion in the precedence tree.
                // see §6: characteristics of incidental motions as a class.
                all im : IncidentalMotion, m : Motion |
                        (m in im.isAppliedTo )
                                => (m in (im.^(takesOver) + im.takesOver.yieldsTo))
                //If an incidental motion is applied to a parent in the precedence structure,
                //then this parent must be a pendingQuestion in the sessionState where p is pending
```

```
		all i : IncidentalMotion, ses : SessionState |
			(i in ses.pendingQuestions) =>
				(i.isAppliedTo & i.^takesOver) in ses.pendingQuestions//TODO why this ? to find out, remove and run the contrary predicate
	}

		/**
	* A Subsidiary motion
	*/
		abstract sig SubsidiaryMotion extends SecondaryMotion {}
		{
			//A subsidiary motion that is linked to no session state and that is not an auxiliary atom
			//is involved in no "applied to" relation** ; one inside has to be in at least one.
			this not in (sessionStateElements + sessionStateElements.allAux)=>
				no isAppliedTo
					else ( some isAppliedTo )
		}
		fact{
			//A SusidiaryMotion in the sessionState's pending stack is only applied to
			//a lower motion in the takesOver transitive closure
			all sm : SubsidiaryMotion, m : Motion |
				(m in sm.isAppliedTo )
					=> (m in sm.^(takesOver))
			//A Subsidiary motion inside a pending stack only applies to motions inside this pending stack
			all s : SubsidiaryMotion , ses : SessionState|
				(s in ses.pendingQuestions) => s.isAppliedTo in ses.pendingQuestions
			//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)
		}

		/**
	* A privileged motion
	*/
		abstract sig PrivilegedMotion extends SecondaryMotion {}
		{
			//Privileged motions don't apply to other motions (their matter is 'self-contained')
			//§6: Privileged motions
			no isAppliedTo
			this not in DebatableMotion
		}
		fact{
			//A Privileged motion inside a pending stack only applies to motions inside this pending stack
			all p : PrivilegedMotion, ses : SessionState |
				(p in ses.pendingQuestions) => p.isAppliedTo in ses.pendingQuestions
```

```
        }

        /**
    *A motion that can be debated
    */
        sig DebatableMotion in Motion {}

        /**
    *A motion that can be amended
    */
        sig AmendableMotion in Motion {}

                                                                                /***********
*****************************
                                                                                 *
TERMINAL-INCIDENTAL-MOTIONS *

****************************************/
                                                                                pred
TERMINAL_INCIDENTAL_MOTIONS {}

/* *
 * A terminal IncidentalMotion
 * §12 Standard descriptive characteristics 2.
 * §27 Standard descriptive characteristics
 * §27 Further rules and explanations, specification of the manner
 * §27 Division of a Question, last paragraph "The procedure is then the same..."
 * in which the question is to be divided
 * ** A formal motion to divide the question must always be made
 * ** The eventually separated questions are already visible in the
 * precedence tree.
 * ** The division of the amendments along with their rootMotion is not modeled.
 *
 */
sig DivisionOfQuestion extends IncidentalMotion {
        separatedQuestions : some Motion
        ,aux : set DivisionOfQuestion
}
{
        //by definition of this motion:
        one isAppliedTo
        this not in separatedQuestions
        //** When adopted the divided question becomes one of
        //the new independent questions that are rendered reachable.
        isAppliedTo in separatedQuestions
        //** a question is at most separated in two other
        one (separatedQuestions-isAppliedTo)
        //the separatedQuestions are siblings in the precedence tree
        isAppliedTo in SecondaryMotion
```

```
        => separatedQuestions in isAppliedTo.@takesOver.@yieldsTo
    isAppliedTo in RootMotion
        =>  separatedQuestions in isAppliedTo.@dealsWith.@isDealtWithBy
    //separated questions are of same type
    haveSameType[separatedQuestions]
    //§27 Standard descriptive characteristics 1. //GOOD EXAMPLE
    //Takes precedence over the main motion and over the subsidiary motion to Postpone
indefinitely
    //If applied to an amendement it also takes precedence over that amendment
    some (isAppliedTo&Amend) =>
    takesOver in (RootMotion+PostponeIndefinitely+(isAppliedTo&Amend))

    else
    takesOver in (RootMotion+PostponeIndefinitely)
    //A motion to divide the main question cannot be made while an amendement to the main
question is pending.
    //(Meaning there is no amendement to the main question in the this ^takesOver set)
    no (takesOver.*@takesOver) & ((@isAppliedTo.(getRoot[this]))&Amend)
    //it yields to all subsidiary motions exept PstponeIndefinitely, Amend and LimitExtendDebate,
to all Privileged and to all applicable Incidental
    yieldsTo in (SubsidiaryMotion-(PostponeIndefinitely+Amend+LimitExtendDebate)
                                    +PrivilegedMotion
                                    +IncidentalMotion)
    //A motion to divide can be made at any time that the main motion , an amendment
    // which it is proposed to divide or the motion to PostponeIndefinitely is immediately pending
    //("even after the previous question has been ordered "=> unnecessary to model)
    takesOver in (RootMotion+PostponeIndefinitely+isAppliedTo&Amend)
    //§27 Standard descriptive characteristics 2.
    //Can be applied to main motions and their amendments if they are scuceptible to dvision.
//  isAppliedTo in RootMotion+((@isAppliedTo.RootMotion)&Amend)
/*  (some root : RootMotion | root in isAppliedTo) => (
        (some root : RootMotion | (root + (root.(~@isAppliedTo))&Amend) in isAppliedTo)
or//suceptible to division
        (some root : RootMotion | (root + in isAppliedTo) //not suceptible to division

                                                                )*/
    //no Subsidiary can be aplied to it except Amend and PreviousQuestion
    some (@isAppliedTo.this) => @isAppliedTo.this not in (SubsidiaryMotion - (Amend+
PreviousQuestion))
/*  //§27 Standard descriptive characteristics 1.
    //about precedence
    takesOver in (RootMotion+PostponeIndefinitely+Amend)
    some (isAppliedTo & RootMotion)
        => no a : Amend| some(a.@isAppliedTo & RootMotion) and a in this.^@takesOver
    yieldsTo in (SubsidiaryMotion-(PostponeIndefinitely+Amend+LimitExtendDebate)
                                    +PrivilegedMotion
                                    +IncidentalMotion)
    //§27 Standard descriptive characteristics 2.
    //about application
```

```
		isAppliedTo in
			((RootMotion + Amend)&takesOver)
			+ ( ((takesOver&PostponeIndefinitely).@takesOver) & (RootMotion + Amend) )
*/
		//debate and amendment
		this not in DebatableMotion
		this in AmendableMotion
		//does not apply to siblings
		isAppliedTo in this.^@takesOver
}
fact{
//no overlapped divided questions
all disj doq1, doq2 : DivisionOfQuestion | no (doq1.separatedQuestions & doq2.separatedQuestions)
//no unchronological dependencies
//all disj doq1, doq2 : DivisionOfQuestion | not (doq1.separatedQuestions = doq2.separatedQuestions)
}

/* *
 * A terminal IncidentalMotion
 * §12 Standard descriptive characteristics 2.
 * §28 Standard descriptive characteristics
 * ** This motion is modeled as one motion with multiple debate
 * and voting rounds allowed. Its internal characteristics are
 * not modeled.
 */

sig ConsiderationByParagraph extends IncidentalMotion {
		aux : set ConsiderationByParagraph
}
{
		//precedence
		takesOver in (RootMotion+PostponeIndefinitely+Amend)
		takesOver in Amend => takesOver = isAppliedTo
		all root : RootMotion, a : Amend |
			(a in @isAppliedTo.root) => not (this in a.^@yieldsTo and this in @isAppliedTo.root)
		yieldsTo in (IncidentalMotion
							+PrivilegedMotion
							+SubsidiaryMotion
							-PostponeIndefinitely
							-Amend
							-LimitExtendDebate)
		//application
		one isAppliedTo //TODO is it always so?
		isAppliedTo in (RootMotion + Amend)
		no (@isAppliedTo.this & SubsidiaryMotion-Amend-PreviousQuestion)
		//debate and amendments
		this not in DebatableMotion
		this in AmendableMotion
		//does not apply to siblings
```

```
        isAppliedTo in this.^@takesOver
        //auxiliaries are from the same set
        //aux in ConsiderationByParagraph
}

/**
 * A terminal IncidentalMotion
 * §12 Creating a blank b) and last paragraph "When a blank exists..."
 * §12 Filling blanks, last paragraph "The particular circumstances..."
 * ** The proposals to fill a blank are considered as being part of the motion
 * to create this blank so that the motion CreateBlank is viewed as
 * one motion with multiple voting rounds allowed.
 * ** The motions to CloseSuggestion and CloseNominations will not be modeled.
 */

sig CreateBlank extends IncidentalMotion {
        aux : set CreateBlank
}
{
        this not in DebatableMotion
        this not in AmendableMotion
        //not in order when PreviousQuestion has been moved before it
        all root : RootMotion, pq : PreviousQuestion |
                ((this+pq in root.^@yieldsTo) and (pq in @isAppliedTo.root))
                        => this not in pq.^@yieldsTo
        //Can apply directly or to the same motion than a primary or secondary amendment
        //§12 FIlling Blanks, CREATING A BLANK b)
        isAppliedTo in (takesOver+(takesOver&Amend).@takesOver
                                                        +
(((takesOver&Amend).@takesOver)&Amend).@takesOver)
        //cannot be applied soleley the PreviousQuestion
        //§12 FIlling Blanks, CREATING A BLANK b)
        all pq : PreviousQuestion&@isAppliedTo.this | isAppliedTo in pq.@isAppliedTo
        //does not apply to siblings
        isAppliedTo in this.^@takesOver
        //applies to one motion at a time
        one isAppliedTo
}
                                                                        /***********
*****************************
                                                                         *
TERMINAL-SUBSIDIARY-MOTIONS *

*****************************************/
                                                                        pred
TERMINAL_SUBSIDIARY_MOTIONS {}
        /* *
    *An individual Subsidiary Motion
    *Rank 1
```

```
    *§6:Listing of individual subsidiary motions 1)
    */
        sig PostponeIndefinitely extends SubsidiaryMotion {
                aux : set PostponeIndefinitely
        }
        {
                //*§6:cases when one subsidiary can be applied ton another
                this not in AmendableMotion
                this in DebatableMotion
                one isAppliedTo
                //auxiliaries are from the same set
                //aux in PostponeIndefinitely
        }
        fact{

                //Natural ranking priorities
                //(sometimes also imply how the motion can be applied since
                //if a motion cannot take precedence over another it cannot be applied to it)
                no (PostponeIndefinitely.^takesOver & SubsidiaryMotion)
        }

        /**
    *An individual Subsidiary Motion
    * Rank 2
    * §6:Listing of individual subsidiary motions 2)
    * §6: cases when one subsidiary can be applied ton another =>
    * NOTE : It seems that Amend can take precedence over any motion**
    */
        abstract sig Amend extends SubsidiaryMotion {
                //concretetakesOver : set Motion
                context : set Motion+Business
        }
        {
                //auxiliary atoms : each main amend that has targets knows about its context
                let tree = getTree[this] ,
                        children = this+(this.^@yieldsTo),
                        targets = getTargets[this] |
                                (not disable_auxiliary_atoms and hasTargets[this] and isMain[this])
=> //TODO : refactor: put this on top?
                                        context = tree - children +targets
                                else no context
                //context elements must have auxiliaries
                all elem : context | some elem.allAux
                //concretetakesOver = skipAmend[this, Motion.takesOver]
                this in AmendableMotion
                one isAppliedTo
                //Amend applies only to Amendable motions and RootMotions
                //§6: cases when one subsidiary can be applied ton another
                isAppliedTo in (RootMotion + AmendableMotion)
```

```
		// Motion that, due to their characteristics and particularities, end up
		// not yielding to Amend. See:
		// §16 Standard descriptive characteristics 1)
		no (takesOver & (PreviousQuestion + LayOnTheTable))
		//no (concretetakesOver & (PreviousQuestion + LayOnTheTable))
		//§12 Standard descriptive characteristics 1. a)
		//rule when amend is applied to a RootMotion
		some (isAppliedTo & RootMotion)
			=> takesOver in (RootMotion + PostponeIndefinitely)
			  and yieldsTo in
			  (SubsidiaryMotion-PostponeIndefinitely+PrivilegedMotion+IncidentalMotion)
			  and (some yieldsTo & Amend => (all a :yieldsTo & Amend | this in
a.@isAppliedTo))
		/* §12 Standard descriptive characteristics 1. b) */
		//rule when amend is applied to a secondary motion
		some (isAppliedTo & SecondaryMotion)
			=> takesOver = isAppliedTo
				and yieldsTo in (PrivilegedMotion+SubsidiaryMotion+IncidentalMotion)
				and ((some (yieldsTo & Amend+LimitExtendDebate+PreviousQuestion))
					=> (all m : (yieldsTo &
Amend+LimitExtendDebate+PreviousQuestion)| this in m.@isAppliedTo))
				//draft, see §12 Standard descriptive characteristics 1. b)
				//and (some atoms : set univ | isAppliedTo.yieldsTo in atoms
				//                              => yieldsTo in atoms)
		/*  §12 Standard descriptive characteristics 2. */
		//what can apply to amend
		@isAppliedTo.this in (Amend
                                        +LimitExtendDebate
                                        +PreviousQuestion
                                        +DivisionOfQuestion
                                        +ConsiderationByParagraph
                                        +CreateBlank)
		no (@isAppliedTo.this & (Commit
                                        +Postpone
                                        +LayOnTheTable
                                        +PostponeIndefinitely))
		/*****BINGO1?******/
		all m : (@isAppliedTo.this) & (LimitExtendDebate+PreviousQuestion) |
                                this in m.@takesOver // BINGO? : for the PreviousQuestion to
applly to Amend, Amend must be immediately pending
		/*****BINGO1?******/		/*  §12 Standard descriptive characteristics 6. */
		//secondary amendments cannot be amended
		some (isAppliedTo & Amend) => no ( @isAppliedTo.this & Amend)
	}

	/**
   * Debatable Amend for debatables
   * §6: cases when one subsidiary can be applied ton another
   * Rank 2
```

```
*/
	sig AmendDebatable extends Amend {
		aux : set AmendDebatable
	}
	{
		isAppliedTo in DebatableMotion
		this in DebatableMotion
		//auxiliaries are from same set
		//aux in AmendDebatable
	}

	/**
* Undebatable Amend for undebatables
* §6: cases when one subsidiary can be applied ton another
* Rank 2
*/
	sig AmendUndebatable extends Amend {
		aux : set AmendUndebatable
	}
	{
		no (isAppliedTo & DebatableMotion) //isAppliedTo not in DebatableMotion
		this not in DebatableMotion
		//auxiliaries are from same set
		//aux in AmendUndebatable

	}

	/**
* An individual Subsidiary Motion
* §6:Listing of individual subsidiary motions 3)
* Rank 3
*/
	sig Commit extends SubsidiaryMotion {
		aux : set Commit
	}
	{
		this in AmendableMotion
		this in DebatableMotion
		one isAppliedTo
		//§13 Standard Descriptive Characteristics 2.
		//cannot be applied to any subsidiary motion
		no isAppliedTo&SubsidiaryMotion
	}
	fact{
		//natural ranking priorities over other subsidiary motions
		//(sometimes also imply how the motion can be applied since
		//if a motion cannot take precedence over another it cannot be applied to it)
		no (Commit.^(takesOver)
			&  (LayOnTheTable + PreviousQuestion + LimitExtendDebate + Postpone +
```

```
Commit ))
	}

	/**
	*An individual Subsidiary Motion
	*§6:Listing of individual subsidiary motions 3)
	*Rank 4
	*/
	sig Postpone extends SubsidiaryMotion {
		aux : set Postpone
	}
	{
		this in AmendableMotion
		this in DebatableMotion
		one isAppliedTo
		//§13 Standartd DEscriptive caracteristics 2.
		isAppliedTo in RootMotion
		//auxiliaries are from same set
		//aux in Postpone
		//movedOnce[this]
		//appliesOnce[this]
	}
	fact{

		//natural ranking priorities over other subsidiary motions
		//(sometimes also imply how the motion can be applied since
		//if a motion cannot take precedence over another it cannot be applied to it)
		no (Postpone.^takesOver
			&  (LayOnTheTable + PreviousQuestion + LimitExtendDebate + Postpone))
	}

	/**
	*A individual Subsidiary Motion
	*§6:Listing of individual subsidiary motions 5)
	*Rank 5
	*/
	sig LimitExtendDebate extends SubsidiaryMotion {
		aux : set LimitExtendDebate
	}
	{
		this in AmendableMotion
		this not in DebatableMotion
		//can apply to a series
		some isAppliedTo
		//§15 Standard descriptive characteristics 2. The immediatePending question should
		// be part of the motion this is applied to
		/*****BINGO1?******/
		takesOver in isAppliedTo
		/*****BINGO1?******/
```

```
		//auxiliaries are from same set
		//aux in LimitExtendDebate
		//movedOnce[this]
		//appliesOnce[this]
	}
	fact{
		//natural ranking priorities over other subsidiary motions
		//(sometimes also imply how the motion can be applied since
		//if a motion cannot take precedence over another it cannot be applied to it)
		no (LimitExtendDebate.^takesOver
			&  (LayOnTheTable + PreviousQuestion + LimitExtendDebate))
	}

	/**
* An individual Subsidiary Motion
* §6:Listing of individual subsidiary motions 6)
* Rank 6
*/
	sig PreviousQuestion extends SubsidiaryMotion {
		aux : set PreviousQuestion
	}
	{
		//§6: cases when one subsidiary can be applied ton another
		this not in AmendableMotion
		this not in DebatableMotion
		some isAppliedTo
		//§16 Standard descriptive characteristics 2. The immediatePending question should
		// be part of the motion this is applied to
		/*****BINGO1?******/
		takesOver in isAppliedTo
		/*****BINGO1?******/
	}
	fact{
		//natural ranking priorities over other subsidiary motions
		//(sometimes also imply how the motion can be applied since
		//if a motion cannot take precedence over another it cannot be applied to it)
		no (PreviousQuestion.^takesOver
			&  (LayOnTheTable + PreviousQuestion ))
	}

	/**
* An individual Subsidiary Motion
* §6:Listing of individual subsidiary motions 7)
* Rank 7
*/
	sig LayOnTheTable extends SubsidiaryMotion {
		aux  : set LayOnTheTable
	}
	{
```

```
		//*§6:cases when one subsidiary can be applied ton another
		this not in AmendableMotion
		this not in DebatableMotion
		//§7 : Standard descriptive characteristics 1.
		isAppliedTo in RootMotion
		//auxiliaries are from same set
		//aux in LayOnTheTable
		//movedOnce[this]
		//appliesOnce[this]
	}
	fact{
		//natural ranking priorities over other subsidiary motions
		//(sometimes also imply how the motion can be applied since
		//if a motion cannot take precedence over another it cannot be applied to it)
		no (LayOnTheTable.^takesOver &  LayOnTheTable)
	}

/************
**********************************
 *
TERMINAL-PRIVILEDGED-MOTIONS     *

*********************************************/
pred
TERMINAL_PRIVILEDGED_MOTIONS {}

	/**
	 * Rank 1
	 */
	sig CallForOrdersOfDay extends PrivilegedMotion {
		aux : set CallForOrdersOfDay
	}
	{
		this not in DebatableMotion
		this not in AmendableMotion
		//auxiliaries are from same set
		//aux in CallForOrdersOfDay
		//movedOnce[this]
		//appliesOnce[this]
	}
	fact {
		//natural ranking priorities over other privileged motions
		no (CallForOrdersOfDay.^(takesOver) & PrivilegedMotion)
	}

	/**
	 * Rank 2
	 * NOTE :a privileged motion can be moved like a main motion.
	 *		§6: Incidental main motions corresponding to privileged motions
```

```
	 *                §19 Raise a question of privilege
	 *             §19 steps in raising and disposing of a question of privilege
	 */
	sig RaiseQuestionOfPrivilege extends PrivilegedMotion {
		questionOfPrivilege : one MainMotion, // **(can also be a request)//TODO is this a s.h.?
		aux : set RaiseQuestionOfPrivilege
	}
	{
		//The question of privilege dealsWith the same business this motion is on top of
		some (questionOfPrivilege.dealsWith)&(this.(^@takesOver).dealsWith)&(Business)
		//the questionOfPrivilege is a new question
		no (questionOfPrivilege&this.(^@takesOver))
		this not in DebatableMotion
		this not in AmendableMotion
		//questions of privilege do not overlap
		lone @questionOfPrivilege.questionOfPrivilege
	}
	fact{
		//natural ranking priorities over other privileged motions
		no (RaiseQuestionOfPrivilege.^(takesOver) & (RaiseQuestionOfPrivilege + Recess +
Adjourn + FixTimeAdjourn))
	}
	/**
	 * Rank 3
	 */
	sig Recess extends PrivilegedMotion {
		aux : set Recess
	}
	{
		this not in DebatableMotion
		this in AmendableMotion
		//auxiliaries are from same set
		//aux in Recess
		//movedOnce[this]
		//appliesOnce[this]
	}
	fact {
		//natural ranking priorities over other privileged motions
		no (Recess.^(takesOver) & (Recess + Adjourn + FixTimeAdjourn))
	}

	/**
	 * Rank 4
	 */
	sig Adjourn extends PrivilegedMotion {
		aux : set Adjourn
	}
	{
		this not in DebatableMotion
```

```
		this not in AmendableMotion
		//auxiliaries are from same set
		//aux in Adjourn
		//movedOnce[this]
		//appliesOnce[this]
	}
	fact {
		//natural ranking priorities over other privileged motions
		no (Adjourn.^(takesOver) & (Adjourn + FixTimeAdjourn))
	}

	/**
	 * Rank 5
	 * Can take precedence over anything
	 */
	sig FixTimeAdjourn extends PrivilegedMotion {
		aux : set FixTimeAdjourn
	}
	{
		this not in DebatableMotion
		this in AmendableMotion
		//auxiliaries are from same set
		//aux in FixTimeAdjourn
		//movedOnce[this]
		//appliesOnce[this]
	}
	fact {
		//natural ranking priorities over other privileged motions
		no (FixTimeAdjourn.^(takesOver) & (FixTimeAdjourn))
	}

                                                                /************************
*********
                                                                 * AUXILIARY
FUNCTIONS     *

*********************************/
                                                                pred
AUXILLIARY_FUNCTIONS {}

	fun pendingStack(ss : SessionState): set Motion {
		(ss.immediatePendingQuestion + ss.immediatePendingQuestion.^(takesOver))
	}

	pred movedOnce[ motions : set univ] {
		no disj m1, m2 : motions |
			(some m1.takesOver => m1.takesOver = m2.takesOver) and
			(some m1.isAppliedTo => m1.isAppliedTo = m2.isAppliedTo)
	}
```

```
	fun firstSessionState : one SessionState{
		SessionState - (SessionState & prev.SessionState)
	}

	fun lastSessionState : one SessionState{
		SessionState - (SessionState & next.SessionState)
	}

	fun primaryAmendments : set Amend {
		Amend.isAppliedTo & Amend
	}

	fun secondaryAmendments : set Amend {
		isAppliedTo.Amend & Amend
	}

	pred isAuxiliary[atom : univ] {
		// An atom isAuxiliary when it is pointed at
		some  allAux.atom
	}

	fun sessionStateElements : set univ {
	//A sessionStateElement is one reachable from some SessionState
			SessionState.orderOfBusiness
		+SessionState.orderOfBusiness.isDealtWithBy.*yieldsTo
//		+	SessionState.toComeToOrder
//		+SessionState.toComeToOrder.^yieldsTo

	}

	pred isMain [atom : univ]  {
		not isAuxiliary[atom]
   }

	fun nonSessionStateElements : set univ {
		(Motion + Business) - sessionStateElements
  }


	fun allAux : (Motion+Business) -> (Motion+Business) {
		( Business <: aux) +
		( MainMotion <: aux) +
//		( OriginalMainMotion <: aux) +
//		( IncidentalMainMotion <: aux) +
//		( BQAAMotion<: aux) +
		( DivisionOfQuestion <: aux) +
		( ConsiderationByParagraph<: aux) +
		( CreateBlank <: aux) +
```

```
                ( PostponeIndefinitely <: aux) +
                ( AmendDebatable<: aux) +
                ( AmendUndebatable <: aux) +
                ( Commit <: aux) +
                ( Postpone<: aux) +
                ( LimitExtendDebate<: aux) +
                ( PreviousQuestion<: aux) +
                ( LayOnTheTable<: aux) +
                ( CallForOrdersOfDay<: aux) +
                ( RaiseQuestionOfPrivilege <: aux) +
                ( Recess<: aux) +
                ( Adjourn<: aux) +
                ( FixTimeAdjourn <: aux)
        }

        pred haveSameType [ motions : Motion ] {
//                motions in OriginalMainMotion  or
//                motions in IncidentalMainMotion  or
                motions in MainMotion  or
//                motions in BQAAMotion or
                motions in DivisionOfQuestion  or
                motions in ConsiderationByParagraph or
                motions in CreateBlank  or
                motions in PostponeIndefinitely  or
                motions in AmendDebatable or
                motions in AmendUndebatable  or
                motions in Commit  or
                motions in Postpone or
                motions in LimitExtendDebate or
                motions in PreviousQuestion or
                motions in LayOnTheTable or
                motions in CallForOrdersOfDay or
                motions in RaiseQuestionOfPrivilege  or
                motions in Recess or
                motions in Adjourn or
                motions in FixTimeAdjourn
   }

        fun getTree [a : Motion+Business ] : set Motion+Business {
                ((
                (a.(^takesOver))
                                        .dealsWith)
                                                .isDealtWithBy)
                                                        .(^yieldsTo+dealsWith+iden)
        }

/***
 * A 'target' of a motion to amend is a child that
 * applies to a parent of this amend
```

```
 */
pred hasTargets[a : Amend] {

	some (a.yieldsTo.isAppliedTo)&(a.^takesOver)
}
/**
*returns motions eventualy making this Amend transparent
*/
fun getTargets[a : Amend] : set SecondaryMotion {
	(isAppliedTo.( (a.yieldsTo.isAppliedTo)&(a.^takesOver) )) & (a.yieldsTo )
}

fun getBusiness[motion : Motion] : one Business{
	(motion.^takesOver.dealsWith)+(motion.dealsWith)
}

/**
 * All motions motion applies to and
 * other concerned motions when motion is applied to a RootMotion
 */
fun getAppliedTo[motion : SecondaryMotion] : set Motion {
                                                motion.isAppliedTo
                                                + ((RootMotion & motion.isAppliedTo).^yieldsTo) &
motion.^takesOver
}

/**
*
*/
fun getSiblings[motion : Motion] : set Motion {
	(motion.(dealsWith+takesOver).(isDealtWithBy+yieldsTo))-motion
}

fun getRoot[motion: Motion] : lone RootMotion {
	(motion.*takesOver)&RootMotion
}
                                                                                /***********
*
                                                                                 * TRACES *

**********/
                                                                        pred TRACES {}

	pred initialSession[firstState : SessionState]{
		//A session starts with no questions being pending**
		no firstState.pendingQuestions
		// the session starts with a business being current
		some firstState.currentBusiness
		//nothing is adopted**
```

```
		no firstState.adopted
		//A session starts with no questions being under some order**
		no firstSessionState.
(underCommitOrPostponeIndefinitelyOrder+laidOnTheTable+underPreviousQuestionOrder)
		//the eventually separated motions and the auxiliary motions are not toComeToOrder
		//every other motion is toComeToOrder
//max : all (DivisionOfQuestion.isAppliedTo) are in tCO
		firstState.toComeToOrder =
			(firstState.orderOfBusiness.isDealtWithBy.*yieldsTo)
		 -
			(	((DivisionOfQuestion.separatedQuestions-
(DivisionOfQuestion.isAppliedTo)).*yieldsTo)
		  +(RaiseQuestionOfPrivilege.questionOfPrivilege.*yieldsTo)
			)
/*//min : no (DivisionOfQuestion.isAppliedTo) is in tCO
			(firstState.orderOfBusiness.isDealtWithBy.*yieldsTo)
		 -
			(	((DivisionOfQuestion.separatedQuestions).*yieldsTo)
		  +(RaiseQuestionOfPrivilege.questionOfPrivilege.*yieldsTo)
			) in firstState.toComeToOrder
//constraint about separatedQuestions
all dq1, dq2 : DivisionOfQuestion | (some dq1.separatedQuestions&dq2.separatedQuestions)=>(
			one
((dq1.separatedQuestions&dq2.separatedQuestions)&(firstState.toComeToOrder)&((dq1+dq2).isAppli
edTo))
			and
((dq1.separatedQuestions&dq2.separatedQuestions)&(firstState.toComeToOrder)).^yieldsTo in
firstState.toComeToOrder
			)
*/
	}

	pred finalSession[lastState : SessionState]{
		//A session ends with no questions being pending
//		no (lastState.pendingQuestions)&RootMotion
		//An ended session has no more motions to come to order
		//activating this makes the predicate become equivalent to reach_all_motions
//		no lastState.toComeToOrder
		//A session is over when there is no more current businesses;
		//because of the implementation the disposeOf action this means
		//that the last business has been dealt with in the previous state
		no (lastState.currentBusiness)
	}

	fact sessionTrace{
//			initialSession[firstSessionState]//[sessionOrd/first]
		all sBefore : SessionState - lastSessionState | //sessionOrd/last |
		 let sAfter  = sBefore.next |
			(	one motion : Motion |
```

```
                                        move[sBefore, sAfter, motion] or   //TODO : does the order of
these matter?
                                        adopt[sBefore, sAfter, motion] or
                                        disposeOf[sBefore, sAfter, motion ])

//                      finalSession[lastSessionState] //[sessionOrd/last]
        }

                                                                                /***********
**********
                                                                                 * BASIC
ACTIONS *

**********************/
                                                                        pred
BASIC_ACTIONS {}


        pred move_preconditions[sBefore : SessionState, motion : Motion ]{
                //if the RootMotion of the before state is under some order then move is false
//TODO :remove
                no ( (sBefore.underCommitOrPostponeIndefinitelyOrder)
                                & RootMotion)
                no ( (sBefore.laidOnTheTable)
                                & RootMotion)
                //if the ipq of the before state is under PreviousQuestion order then move is false
for subsidiary Motions
                //exception made for LayOnTheTable §16 PreviousQuestion 1) and 2)
            (motion in (SubsidiaryMotion-LayOnTheTable))=>
                no (sBefore.immediatePendingQuestion
                        & (sBefore.underPreviousQuestionOrder))
                //if the ipq of the before state is adopted move is false (proposals to fill blanks are
not modelled)
                //§4 The handeling of a motion , chair's announcement of the voting result;
verifiaction procedures
                //and cases where the chair votes.
                no (sBefore.immediatePendingQuestion & sBefore.adopted)
                //if there is some separatedQuestions still to come to order among the before state
                //ipq's children then the moved motion must be one of them.
                //§27 Division of the Question, Form and example
                some (sBefore.immediatePendingQuestion.yieldsTo)
                        & (DivisionOfQuestion.separatedQuestions)
                        & (sBefore.toComeToOrder)
                        => motion in (sBefore.immediatePendingQuestion.yieldsTo)
                                        & (DivisionOfQuestion.separatedQuestions)
                                        & (sBefore.toComeToOrder)
            motion in sBefore.toComeToOrder
            motion in RootMotion => motion.dealsWith = sBefore.currentBusiness
        }
```

```
	/* *
	 * A non second requiring motion is moved by a member who has
	 * obtained the floor for it to become a pending question.
	 * It doesn't need to be seconded by another member to become pending.
    */
	pred move[sBefore, sAfter:SessionState, motion : Motion]{
		/** initial situation **/
		move_preconditions[sBefore, motion]
		/** situation after the action **/
		motion = sAfter.immediatePendingQuestion
		motion.takesOver = sBefore.immediatePendingQuestion
		sAfter.pendingQuestions = sBefore.pendingQuestions + motion
		/** everything else stays the same **/
		sAfter.toComeToOrder = sBefore.toComeToOrder - motion
		sAfter.adopted = sBefore.adopted
		sAfter.orderOfBusiness = sBefore.orderOfBusiness
		sAfter.currentBusiness = sBefore.currentBusiness
		sAfter.underCommitOrPostponeIndefinitelyOrder =
sBefore.underCommitOrPostponeIndefinitelyOrder
		sAfter.underPreviousQuestionOrder = sBefore.underPreviousQuestionOrder
		sAfter.laidOnTheTable = sBefore.laidOnTheTable
	}

	pred adopt_preconditions[sBefore : SessionState, motion: Motion]{
		motion = sBefore.immediatePendingQuestion
		motion not in sBefore.adopted
			//adopt is false when there is some separated motion still toComeToOrder
			//among the children of the to be adopted motion
		no (sBefore.immediatePendingQuestion.yieldsTo)
			& (DivisionOfQuestion.separatedQuestions)
			& (sBefore.toComeToOrder)
	}

	/**
	 * A motion has to be adopted in order to take effect
	 * either by :
	 * - a vote §4:Putting the question
	 * - directly because of the motion's nature
	 * - upon a ruling of the chair
	 */
	pred adopt[sBefore, sAfter:SessionState, motion : Motion] {
		/** initial situation **/
		adopt_preconditions[sBefore, motion]
		/** situation after the action **/
		motion in sAfter.adopted
			//If the ipq is Postpone or Commit then the motions they are
			//applied to are in the underCommitOrPostponeIndefinitelyOrder set of the after
sessionState
			//if the motion is applied to a RootMotion then all the motions between the
```

```
                    //root and the motion are under the former order too.
                motion in (PostponeIndefinitely + Commit)
                            => (sAfter.underCommitOrPostponeIndefinitelyOrder =
sBefore.underCommitOrPostponeIndefinitelyOrder + getAppliedTo[motion] and
                                    sAfter.underPreviousQuestionOrder =
sBefore.underPreviousQuestionOrder - getAppliedTo[motion] )
                            else(sAfter.underCommitOrPostponeIndefinitelyOrder =
sBefore.underCommitOrPostponeIndefinitelyOrder and
                                    //If the ipq is the PreviousQuestion then the motions it is
                                    //applied to are in the underPreviousQuestionOrder set of
the after sessionState
                                    //if the motion is applied to a RootMotion then all the
motions between the
                                    //root and the motion are under the former order too.
                                    (motion in (PreviousQuestion) //NOTE : maybe useless
putting both together
                                            => sAfter.underPreviousQuestionOrder
                                                    =
(sBefore.underPreviousQuestionOrder + getAppliedTo[motion])
                                            else(
                        (motion in (DivisionOfQuestion) and some
motion.isAppliedTo&sBefore.underPreviousQuestionOrder)
                    => sAfter.underPreviousQuestionOrder =
sBefore.underPreviousQuestionOrder+motion.separatedQuestions
                                                else (sAfter.underPreviousQuestionOrder =
sBefore.underPreviousQuestionOrder)
                            )
                                        )
                                    )
                //If the ipq is LayOnTheTable then the motions they are
                //applied to are in the laidOnTheTable set of the after sessionState
                //if the motion is applied to a RootMotion then all the motions between the
                //root and the motion are under the former order too.
                motion in (LayOnTheTable)
                            => sAfter.laidOnTheTable     = (sBefore.laidOnTheTable

                    + motion.isAppliedTo //all parents if applied to the root

                    + ((RootMotion & motion.isAppliedTo).^yieldsTo) & motion.^takesOver)
                            else (sAfter.laidOnTheTable = sBefore.laidOnTheTable)
                //if the motion adopted is RaiseQuestionOfPrivilege the question of privilege raised
                //becomes immediately pending //TODO ref
                motion in (RaiseQuestionOfPrivilege)
                            => (sAfter.immediatePendingQuestion = motion.questionOfPrivilege and
                                    sAfter.pendingQuestions =
sBefore.pendingQuestions+motion.questionOfPrivilege and
                                    sAfter.toComeToOrder =
sBefore.toComeToOrder+motion.questionOfPrivilege.yieldsTo) and
                                    sAfter.currentBusiness = sBefore.currentBusiness
```

```
                    else(
                //if the motion adopted id Division of the Question then all the resulting new
questions
                //should eventually come to order §27 Division of the Question, Form and
example
        motion in DivisionOfQuestion
                    => sAfter.toComeToOrder =
                    (sBefore.toComeToOrder
                     + (motion.separatedQuestions- motion.isAppliedTo)
                     + (
                          ( motion.separatedQuestions- motion.isAppliedTo).^yieldsTo
                        - ( (DivisionOfQuestion-motion).separatedQuestions -
(DivisionOfQuestion - motion).isAppliedTo )
                        )
         ) and
             sAfter.immediatePendingQuestion = sBefore.immediatePendingQuestion and
                        sAfter.pendingQuestions = sBefore.pendingQuestions and
                        sAfter.currentBusiness = sBefore.currentBusiness
        else(
                    //Limited effect of adopting CallForOrdersOfDay
                    //(a skipped business or one due a this time).
                            (motion in CallForOrdersOfDay
                            => (
                                    sAfter.currentBusiness =
sBefore.currentBusiness and
                              sAfter.toComeToOrder =
sBefore.toComeToOrder and
                                    sAfter.pendingQuestions =
sBefore.pendingQuestions-motion
                                        )
                            else(
                                    //limited effect of adopting
lay on the table+ never taken back again
                                    motion in LayOnTheTable
                                    => (

          sAfter.currentBusiness = none
                                    and no
sAfter.immediatePendingQuestion
                                                and
sAfter.toComeToOrder = sBefore.toComeToOrder
                                                and
sAfter.pendingQuestions = sBefore.pendingQuestions - motion

                - motion.isAppliedTo //all parents if applied to the root

                - (((RootMotion & motion.isAppliedTo).^yieldsTo) & motion.^takesOver)
                            )
                                    else(
```

```
                                                                    //limited effect of
adopting Postpone
                                                                    motion in Postpone
                                                                    =>(

        sAfter.toComeToOrder = sBefore.toComeToOrder and

        sAfter.currentBusiness = sBefore.currentBusiness
                                                                                and
sAfter.pendingQuestions = sBefore.pendingQuestions-motion
                                                                                )
                                                                    else (

        (motion in (Commit+PostponeIndefinitely)

        => (

                        sAfter.toComeToOrder = sBefore.toComeToOrder and

                        sAfter.currentBusiness = none and

                        sAfter.pendingQuestions = sBefore.pendingQuestions - motion -
motion.isAppliedTo

                                                                    - (((RootMotion &
motion.isAppliedTo).^yieldsTo) & motion.^takesOver)

                          and no sAfter.immediatePendingQuestion

                )

        else( sAfter.toComeToOrder = sBefore.toComeToOrder and

                        sAfter.immediatePendingQuestion = sBefore.immediatePendingQuestion
and

                    sAfter.currentBusiness = sBefore.currentBusiness and

                        sAfter.pendingQuestions = sBefore.pendingQuestions

                    )

         )
                                                                                )
                                                                            )
                                                                    )
            )
           )
          )
```

```
		//everything else stays the same
		sAfter.orderOfBusiness = sBefore.orderOfBusiness
		sAfter.adopted = sBefore.adopted + motion
	}

	pred disposeOf_preconditions [sBefore: SessionState, motion : Motion] {
		/**initial situation **/
		//disposeOf is true only if the motion is immediately pending //TODO ref
		motion = sBefore.immediatePendingQuestion
			//disposeOf is false when there is some separated motion still toComeToOrder among the
			//children of the to be disposedOf motion  §27 Division of the Question, Form and example
		no (sBefore.immediatePendingQuestion.yieldsTo)
			& (DivisionOfQuestion.separatedQuestions)
			& (sBefore.toComeToOrder)
	}

	/**
	 * A motion must be disposed of before
	 * the next business or motion can be
	 * dealtWith.
	 * note : this action lets the definition
	 * of the ipq determine what will be the
	 * new ipq. The new ipq is thus not described
	 * explicitly here.
	 */
	 pred disposeOf[sBefore, sAfter :SessionState, motion : Motion]{
		/** initial situation **/
		disposeOf_preconditions[sBefore, motion]
		/** situation after the action **/
		motion not in sAfter.pendingQuestions //TODO redundant ?
			//whatever we are taking out of the pendingQuestions set,
			//if it was under some order then it shouldn't be anymore //TODO : this is never used for c, pi and lat since order took effect immediately
			//§16 Previous Question
		motion not in (sAfter.underCommitOrPostponeIndefinitelyOrder
					+sAfter.underPreviousQuestionOrder
					+sAfter.laidOnTheTable)
			//if an adopted division of the question is disposed of
			//all motions between it and its divided questions should be lost//TODO : ref
		(motion in DivisionOfQuestion and motion in sBefore.adopted and not motion.isAppliedTo = motion.takesOver )
				=> (sAfter.pendingQuestions = sBefore.pendingQuestions

				-motion

				-(motion.takesOver)
```

```
                -(motion.^takesOver) + ((motion.isAppliedTo).*takesOver)) and
                        sAfter.immediatePendingQuestion = motion.isAppliedTo
                else (
        motion in (RaiseQuestionOfPrivilege.questionOfPrivilege)
                => (sAfter.pendingQuestions = sBefore.pendingQuestions-
(motion+questionOfPrivilege.motion)and
                        sAfter.immediatePendingQuestion =
questionOfPrivilege.motion.takesOver)
            else (sAfter.pendingQuestions = sBefore.pendingQuestions-motion and
                        sAfter.immediatePendingQuestion =
sBefore.immediatePendingQuestion.takesOver)
                        )
        //transition from one business to another NOTE: since the last has no next
currentBusiness will be empty
        motion in RootMotion =>(
            sAfter.currentBusiness = sBefore.currentBusiness
                                                                                )
        else (   sAfter.currentBusiness = sBefore.currentBusiness)
        //everything else stays the same
        sAfter.orderOfBusiness = sBefore.orderOfBusiness
        sAfter.adopted = sBefore.adopted
        sAfter.toComeToOrder = sBefore.toComeToOrder
        sAfter.underCommitOrPostponeIndefinitelyOrder =
sBefore.underCommitOrPostponeIndefinitelyOrder-motion
        sAfter.underPreviousQuestionOrder = sBefore.underPreviousQuestionOrder-motion
        sAfter.laidOnTheTable = sBefore.laidOnTheTable-motion
    }


    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]
                                                        )
    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 othe ratributes that are not specified by the final state
    sAfter.immediatePendingQuestion = sBefore.immediatePendingQuestion
    sAfter.orderOfBusiness = sBefore.orderOfBusiness
    sAfter.adopted = sBefore.adopted
    sAfter.toComeToOrder = sBefore.toComeToOrder
    sAfter.underCommitOrPostponeIndefinitelyOrder =
sBefore.underCommitOrPostponeIndefinitelyOrder
    sAfter.underPreviousQuestionOrder = sBefore.underPreviousQuestionOrder
    sAfter.laidOnTheTable = sBefore.laidOnTheTable
```

```
	}

/*************
 * SHOW   *
 *************/
pred SHOW {}

	pred show {
		//not disable_auxiliary_atoms
		//disable_auxiliary_atoms
		//whole_session
		session_prefix
		//session_suffix
		//enable_movedOnce
		//reach_all_motions
		//no ErrorFlag
	}

run show for 10 but 5 Int

run show for 1 but   8 SessionState, 5 Int
run show for 2 but   8 SessionState,5 Int
run show for 3 but   8 SessionState,5 Int
run show for 4 but   8 SessionState,5 Int
run show for 5 but   8 SessionState,5 Int
run show for 6 but   8 SessionState,5 Int
run show for 7 but   8 SessionState,5 Int
run show for 8 but   8 SessionState,5 Int
run show for 9 but   8 SessionState,5 Int
run show for 10 but   8 SessionState,5 Int

//---------------DRAFT-TESTS--------------//

/
*********************************************************************************
****
 *   DESCRIPTION :
 *
 * - There is one session of one meeting
 * - Members and authors of moitons are not represented
 * - Motions are not seconded
 * - Votes are abstracted in only their final outcome (adopted or not)
 * - Terminal Subsidiary Motions are instantiated
 * - Terminal Privileged Motions are instantiated
 * - SOme Terminal Incidentals are instantiated
 * - The effect of motions on the content of the
 *         motions they are applied to is abstracted (eg: amendements)
 * - The effect of motions on the status of the
 *         motions they are applied to is represented (eg: motions laid on the table)
```

```
 * - Focus is put on the motion to Amend
 *
 *
 *
*****************************************************************************
**/
module model1

	/* *
	 * An ordering over the Businesses of a sessionState and
	 * over the sessionStates of a Session
	 */
	//open util/ordering[SessionState] as sessionOrd

  /*****************
   * Base facts *
   *****************/
  pred BASE_FACTS{}

   fact linked_signatures{
		//there is a first and a last session state and
		//the session states are a doubly-chained list
		some SessionState => (
			one first, last : SessionState |
				no first.prev and no last.next
				and (all ss : SessionState-(first) | one ss.prev ) //TODO are these 2 lines
useful when we have the reciprocity between next and prev?
					and	(all ss : SessionState-(last) | one ss.next ))

		//reciprocal relations
		(SessionState <: next) = ~ (SessionState <: prev)
	}

	fact utility_facts{

		//by default traces are sessionprefixes and only dynamic mode is allowed
		//session_prefix and dynamic_mode_only

		//The error action induces the existance of an error flag
		( error[lastSessionState.prev, lastSessionState, Motion ] ) =>
			one ErrorFlag else no ErrorFlag
	}

	fact no_cycles {
		//a secondary motion cannot be inside a cycle of isAppliedTo or takesOver relations
		no sm : SecondaryMotion | sm in (sm.^takesOver + sm.^isAppliedTo)
		//a sessionState cannot be inside a cycle of next of prev relations
		no ss : SessionState | ss in (ss.^next + ss.^prev)
		//an atom cannot be inside a cycle of next of prev relations // TODO use allAux
```

```
		no atom : Business | atom in atom.^aux
//		no atom : OriginalMainMotion |	atom in atom.^aux
//		no atom : IncidentalMainMotion | atom in atom.^aux
		no atom : MainMotion |	atom in atom.^aux
//		no atom : BQAAMotion |	atom in atom.^aux
		no atom : DivisionOfQuestion |	atom in atom.^aux
		no atom : ConsiderationByParagraph | atom in atom.^aux
		no atom : CreateBlank |atom in atom.^aux
		no atom : PostponeIndefinitely | atom in atom.^aux
		no atom : AmendDebatable | atom in atom.^aux
		no atom : AmendUndebatable | atom in atom.^aux
		no atom : Commit | atom in atom.^aux
		no atom : Postpone | atom in atom.^aux
		no atom : LimitExtendDebate | atom in atom.^aux
		no atom : PreviousQuestion | atom in atom.^aux
		no atom : LayOnTheTable | atom in atom.^aux
		no atom : CallForOrdersOfDay | atom in atom.^aux
		no atom : RaiseQuestionOfPrivilege | atom in atom.^aux
		no atom : Recess | atom in atom.^aux
		no atom : Adjourn | atom in atom.^aux
		no atom : FixTimeAdjourn | atom in atom.^aux
	}

	fact auxiliary_atoms {

		//auxiliary atoms are either enabled or disabled
		enable_auxiliary_atoms or disable_auxiliary_atoms
		//(enable_auxiliary_atoms and not disable_auxiliary_atoms) or //unsatifiable sentences
		//(not enable_auxiliary_atoms and disable_auxiliary_atoms)

		//When auxiliary atoms are enabled : sessionState elements are main and
		//non- sessionState elements are auxiliary
		enable_auxiliary_atoms =>
			all elem : sessionStateElements |  isMain[elem] and
			all elem : nonSessionStateElements | isAuxiliary[elem]

		//If a relation comes out of /reaches a main atom the same relation comes out of/reaches
its auxiliary atom;
		//exeption made for relations to Amend. In other words :
		//If a session state element but amend is related to something that is not amend than
		//the the auxiliaries of the atoms related to this element are exactly
		//the atoms related to the auxiliaries of this element
		//NOTE : case where a main atom is applyng to a sibling ? => no incidental applies to
siblings in this model
all a : Amend | (
		(all elem : a.context
				| some (elem.dealsWith) =>
					(elem.dealsWith.allAux ) = (elem.allAux.dealsWith)
			else no elem.allAux.dealsWith)
```

```
                and

                (all elem :  a.context
                        | some (elem.takesOver) =>(
                                (some elem & Amend)=>
                                                ( elem.takesOver.allAux =
elem.yieldsTo.allAux.takesOver)
                                else (
                                                        (some elem.takesOver&Amend) =>
                                                                ( elem.takesOver.takesOver.allAux =
elem.allAux.takesOver )
                                                        else ( elem.takesOver.allAux =
elem.allAux.takesOver )
                                                )

                        )
                        else no elem.allAux.takesOver)
                and
                (all elem :  a.context
                        | no (elem.yieldsTo) =>
                                no (elem.allAux.yieldsTo) )
                and
                (all elem :  a.context
                        | some (elem.isAppliedTo) =>
                                (elem.isAppliedTo.allAux) = (elem.allAux.isAppliedTo)
                        else no elem.allAux.isAppliedTo)
                and
                (all elem :  a.context
                        | some (elem.separatedQuestions) =>
                                ((elem.separatedQuestions).allAux ) =
(elem.allAux.separatedQuestions)
                        else no elem.allAux.separatedQuestions)
                and
                (all elem :  a.context
                        | some (elem.questionOfPrivilege) =>
                                ((elem.questionOfPrivilege).allAux ) =
(elem.allAux.questionOfPrivilege)
                        else no elem.allAux.questionOfPrivilege)
                //NOTE: removed some code here because business links are now part of the state
                )
   }

  /*************************
   * General Predicates *
   *************************/
 pred GENERAL_PREDICATES{}

        pred whole_session {
                initialSession[firstSessionState]
```

```
		finalSession[lastSessionState]
	}

	pred session_prefix {
		initialSession[firstSessionState]
	}

	pred session_suffix {
		finalSession[lastSessionState]
	}

	pred reach_all_motions {
		no lastSessionState.toComeToOrder
  }

	pred encourage_move{
		all s : SessionState |
			(some m1 : Motion | move_preconditions[s,m1]) =>(some m2 : Motion| move[s,
s.next, m2])
	}

	pred enable_movedOnce{
		//a motion belonging to the passed signature is moved once on top of another motion
		//§6 Motions that bring a question again before the assembly
		//Amend and RootMotions should always be movabe twice
		movedOnce[PostponeIndefinitely]
		movedOnce[Commit]
		movedOnce[Postpone]
		movedOnce[LimitExtendDebate]
		movedOnce[PreviousQuestion]
		movedOnce[LayOnTheTable]
		movedOnce[CallForOrdersOfDay]
		movedOnce[RaiseQuestionOfPrivilege]
		movedOnce[Recess]
		movedOnce[Adjourn]
		movedOnce[FixTimeAdjourn]
		movedOnce[ConsiderationByParagraph]
		movedOnce[CreateBlank]
		movedOnce[DivisionOfQuestion]



	}

	/**
	* Static and dynamic dependancies will apear
	* (static dependencies alone are not allowed)
	*/
	pred    dynamic_mode_only {
```

```
		//a sessionStateElement business is always part of some order of business**
		all b : Business | (b in sessionStateElements) => (some ss : SessionState | b in
ss.orderOfBusiness)

		//There should always exist at least one sessionState
		some SessionState

	}

	/***
	 * There are auxiliary atoms and non auxiliary atoms and
	 *	non auxiliary atoms point to auxiliary atoms.
	 *
	 * Not suited for actually enabling auxiliary atoms outside the model.
	 * Use "not disable_auxiliary_atoms"
   */
	pred enable_auxiliary_atoms {
		all atom : Business+Motion |
			(no allAux.atom /*and some atom.allAux*/) or (no atom.allAux and one
allAux.atom)
		(no a : Amend | hasTargets[a])=> no nonSessionStateElements
	}

	pred disable_auxiliary_atoms {
		//Duplicate code because of absence of overloading capabilities in alloy (but allAux
workarround!)
		all atom : Business+Motion|  no atom.allAux
		//there should exist only sessionState elements
		no nonSessionStateElements
	}


		/**************************

		 * MODELING CONCEPTS *

		 *************************/

pred MODELING_CONCEPTS {}

	one sig Aux {
		transparentAmends : set Amend,
		contexts : Amend -> ((Motion+Business)  -> (Motion+Business))
	}
	{
		//no motion should point to itself inside any amend context
		all a: Amend, m : Motion, b : Business |
			m not in m.(a.contexts) and b not in b.(a.contexts) and
```

```
                m not in (a.contexts).m and b not in (a.contexts).b
        //Amend is not contained inside its onwn context
        all a : Amend | no a.(a.contexts) and no (a.contexts).a

        //transparentAmends are the main amends who should have auxiliary worlds
//      all a : Amend | isAuxiliary[a] => no ( a & transparentAmends)
//      all a : Amend | not hasTargets[a] => no ( a & transparentAmends)

        all a : Amend |
                ((not disable_auxiliary_atoms) and isMain[a] and hasTargets[a] )=>
                        a in transparentAmends
                else no (a & transparentAmends)

        //contexts projected on Amends do not overlap
        all disj a1, a2 : Amend | no (a1.contexts)&(a2.contexts)
        //each element of an amend's context participates in exactly
        //one auxiliary relation inside the Aux.contexts relation
        let ta = transparentAmends |
                all a : Amend |
                        let context = a.context | //getContext[a] |
                                a in ta =>
                                        ( all elem : context | one ((elem<:allAux) & (a.contexts)) )
                                else ( no a.contexts )
        //Aux.contexts contains only auxiliary relations
        transparentAmends.contexts = allAux

        //contexts projected on some Amend contains this Amend's context
        //all a :Amend | a.contexts = ((a.context) <: allAux ) //TODO

}

/**
* Exists in the purpose of showing there is an error
*/
sig ErrorFlag {}



        /**************************

         * DOMAIN CONCEPTS *

         **************************/

pred DOMAIN_CONCEPTS {}


/***
* A sessionState : the description of a session at a given 'time'.
```

```
	* Refer to :
	* §3 Call to order; Order of Business
	* §4 How a motion is brought before the Assembly
	* §5 Secondary Motions as an Underlying Concept (fundamental principle of parliamentary
law)
	*/
	sig SessionState{
		//sets representing the state of motions
		immediatePendingQuestion : lone Motion, //set Motion, TODO : relax for property
testing pupose ?
		disj toComeToOrder, pendingQuestions : set Motion,
		adopted : set Motion,
		//pendingQuestions2 : set Motion -> Motion,
		//sets of motions affected by a motion moved previously to this state
		laidOnTheTable : set Motion,
		underPreviousQuestionOrder : set Motion,
		underCommitOrPostponeIndefinitelyOrder : set Motion, //**should be separate sets
		//the businesses, **always at least one business to deal with
		orderOfBusiness : lone Business , //TODO : shall the order of business be an
autonomous entity?
		//The current business is always part of the order of business of the sessionState;
		// there may be no current business**
		currentBusiness : lone orderOfBusiness,
		//the linked states
		next : lone SessionState,
		prev : lone SessionState
	}
	{	/* *All these constraints are to be considered inside >this< sessionState exclusively* */

		//no cycles in the sessionStates links
		this not in (next+prev)
		//If secondary motions exist in the pending questions set this must be relatively to a root
motion//TODO useless? redundant?
		some (pendingQuestions & SecondaryMotion) => some (pendingQuestions &
RootMotion)
		//A secondary motion inside the pending stack of this sessionState only takes precedence
over
		// motions inside the latter stack
		pendingQuestions.takesOver in pendingQuestions
		//the ipq is one of the pending questions of the sessionState
		immediatePendingQuestion in pendingQuestions
		//If there are pendingQuestions then there is an ipq
//		some pendingQuestions  => one (immediatePendingQuestion) //TODO for cod =>
useful?
		//The ipq in the pending stack of this sessionState does not yield precedence inside the
former stack
		all i : immediatePendingQuestion |    no (i.yieldsTo & pendingQuestions)
		//If a secondary motion is applied to another secondary motion then all
		//the motions in the yieldsTo relation from this motion are applied to secondary motions
```

```
//                all sm1, sm2 : SecondaryMotion
//                        | sm2 in sm1.isAppliedTo
//                                => (all m : sm1.^(yieldsTo) | m.isAppliedTo in
SecondaryMotion)//TODO find reference or remove !!
            //If a motion is applicable to a set of motions then they are consecutive amongst each other//
TODO find reference
            //(only one head and and only one tail) REMOVED : and consecutive to the former motion.
                all sm : Motion
                        | some sm.isAppliedTo
                                =>      (one head: (sm.isAppliedTo) |
                                  (all other : (sm.isAppliedTo)|
                                          other in head.^(takesOver)+head))
                all sm : Motion
                        | some sm.isAppliedTo
                                =>      (one tail: (sm.isAppliedTo) |
                                   tail.takesOver not in sm.isAppliedTo or tail in RootMotion
                                )
                //The motions to be postponed definitely or indefinitely or committed are among the
pending
                //
(underCommitOrPostponeIndefinitelyOrder+underPreviousQuestionOrder+laidOnTheTable) in
pendingQuestions
        }

        /**
        * A Business
   * A business is dealt with by the means of a root motion
   * Refer to:
   * §3 Means by which business is brought before the assembly
   * §5 Classes of Motions
        */
         sig Business {
                isDealtWithBy : some RootMotion,
                ,aux : set Business

        }
        {       //Unless there are separated questions or a question of privilege
                //Only at most one motion should deal with a Business
                (lone isDealtWithBy)or
                (some (isDealtWithBy&(DivisionOfQuestion.separatedQuestions)))or
                (some (isDealtWithBy&(RaiseQuestionOfPrivilege.questionOfPrivilege)))
        }

        /**
        * A motion
        * abstract -> a motion must have a type.
        * Refer to:
   * §5 Classes of Motions
   * §5 Secondary motion as an underlying concept
```

```
	*/
	abstract sig Motion {
		yieldsTo : set SecondaryMotion
//		,amendlessYieldsTo : set SecondaryMotion
	}
	{
		/* §12 Standard descriptive characteristics 1) */
		//no good
		//amendlessYieldsTo = yieldsTo - Amend + ((yieldsTo & Amend).@yieldsTo)
	}
	fact{
		//these relations are reciprocal
		//§5 Taking of precedence of one motion over another
		yieldsTo = ~takesOver
		//no good because there is actually no other constraint on the amendlessYieldsTo
		//appart from the fact itself that it is trying to constrain.
		//all m : Motion, a : Amend | some m.yieldsTo & a and some m.amendlessYieldsTo &
a.yieldsTo
   }

                                                /*****************************************
**********
                                                 * THE GENEREAL CATEGORIES OF
MOTIONS  *

************************************************/
                                                pred
THE_GENEREAL_CATEGORIES_OF_MOTIONS {}

                                                                                        /*****
****************
                                                                                         *
ROOTMOTIONS *

*********************/
                                                                                        pred
ROOTMOTIONS {}

	/**
	 * A motion that is at the root of a stack of pending questions
	 * Refer to :
    * §5 Order of precedence of motions
	 */
	 abstract sig RootMotion extends Motion{
		dealsWith : one Business //A root motion exists because it deals with a business**
	 }
	 {
		//a root motion in the pending questions of some sessionState must be dealing with some
business
```

```
		//and this business must be a current one in that sessionState.**
		//all ss: SessionState | ((this in ss.pendingQuestions and no ss.specialOrder) => //NOTE :
modified for cod
		//dealsWith in ss.currentBusiness )
		//**All RootMotions are amendable and debatable
		this in AmendableMotion
		this in DebatableMotion
	 }
	fact{
		//reciprocal relations
		dealsWith = ~isDealtWithBy
	}

	/**
	* A main motion
   * abstract -> a main motion has to have a class
   * §5 Classes of Motions
	*/
	sig MainMotion extends RootMotion {
		aux : set MainMotion
  }


	/**
	* A motion that can bring a question again
	* before the assembly
	*/
	/*sig BQAAMotion extends RootMotion {
		aux : set BQAAMotion
	}
  {}*/


/*********************************

	*  SECONDARY_MOTIONS     *

	*********************************/

	pred SECONDARY_MOTIONS {}


	/**
	*A Secondary Motion
	*abstract -> a sm has to belong to a class.
   *§5 Taking of precedence by one motion over another
	*/
	abstract sig SecondaryMotion extends Motion{
		takesOver : lone Motion,
```

```
		isAppliedTo : set Motion
	}
	{
		//A secondary motion exists because it takes precedence over another motion**
		some takesOver
	}
	/**
    * An incidental motion (Secondary)
    *
    */
	abstract sig IncidentalMotion extends SecondaryMotion {}//TODO : abstract ?
	{
		//An incidental motion that is linked to no session state and that is not an auxiliary atom
		//is involved in no "applied to" relation** ; one inside has to be in at least one.
		this not in (sessionStateElements + sessionStateElements.allAux)=>
			no isAppliedTo
				else ( some isAppliedTo )
		//Incidental motions except DivisionOfTheAssembly yield to the privileged motions
		// and one subsidiary motion(LayOnTheTable)**
		// §6: conditions under which incidental motions take precedence over, or yield to, other
motions.
		yieldsTo in (PrivilegedMotion + LayOnTheTable )
    }
    fact{
	//A IncidentalMotion in the sessionState's pending stack is only applied to
		// a lower motion in the takesOver transitive closure or a 'sibling'
		// motion in the precedence tree.
		// see §6: characteristics of incidental motions as a class.
		all im : IncidentalMotion, m : Motion |
			(m in im.isAppliedTo )
				=> (m in (im.^(takesOver) + im.takesOver.yieldsTo))
		//If an incidental motion is applied to a parent in the precedence structure,
		//then this parent must be a pendingQuestion in the sessionState where p is pending
		all i : IncidentalMotion, ses : SessionState |
			(i in ses.pendingQuestions) =>
				(i.isAppliedTo & i.^takesOver) in ses.pendingQuestions//TODO why
this ? to find out, remove and run the contrary predicate
    }

	/**
    * A Subsidiary motion
    */
	abstract sig SubsidiaryMotion extends SecondaryMotion {}
	{
		//A subsidiary motion that is linked to no session state and that is not an auxiliary atom
		//is involved in no "applied to" relation** ; one inside has to be in at least one.
		this not in (sessionStateElements + sessionStateElements.allAux)=>
			no isAppliedTo
				else ( some isAppliedTo )
```

```
	}
	fact{
		//A SusidiaryMotion in the sessionState's pending stack is only applied to
		//a lower motion in the takesOver transitive closure
		all sm : SubsidiaryMotion, m : Motion |
			(m in sm.isAppliedTo )
				=> (m in sm.^(takesOver))
		//A Subsidiary motion inside a pending stack only applies to motions inside this pending
stack
		all s : SubsidiaryMotion , ses : SessionState|
			(s in ses.pendingQuestions) => s.isAppliedTo in ses.pendingQuestions
		//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)
	}

	/**
   * A privileged motion
   */
	abstract sig PrivilegedMotion extends SecondaryMotion {}
	{
		//Privileged motions don't apply to other motions (their matter is 'self-contained')
		//§6: Privileged motions
		no isAppliedTo
		this not in DebatableMotion
	}
	fact{
		//A Privileged motion inside a pending stack only applies to motions inside this pending
stack
		all p : PrivilegedMotion, ses : SessionState |
			(p in ses.pendingQuestions) => p.isAppliedTo in ses.pendingQuestions
	}

	/**
   *A motion that can be debated
   */
	sig DebatableMotion in Motion {}

	/**
   *A motion that can be amended
   */
	sig AmendableMotion in Motion {}

                                                                        /***********
******************************
                                                                         *
TERMINAL-INCIDENTAL-MOTIONS *
```

```
****************************************/
                                                                        pred
TERMINAL_INCIDENTAL_MOTIONS {}

/* *
 * A terminal IncidentalMotion
 * §12 Standard descriptive characteristics 2.
 * §27 Standard descriptive characteristics
 * §27 Further rules and explanations, specification of the manner
 * §27 Division of a Question, last paragraph "The procedure is then the same..."
 * in which the question is to be divided
 * ** A formal motion to divide the question must always be made
 * ** The eventually separated questions are already visible in the
 * precedence tree.
 * ** The division of the amendments along with their rootMotion is not modeled.
 *
 */
sig DivisionOfQuestion extends IncidentalMotion {
        separatedQuestions : some Motion
        ,aux : set DivisionOfQuestion
}
{
        //by definition of this motion:
        one isAppliedTo
        this not in separatedQuestions
        //** When adopted the divided question becomes one of
        //the new independent questions that are rendered reachable.
        isAppliedTo in separatedQuestions
        //** a question is at most separated in two other
        one (separatedQuestions-isAppliedTo)
        //the separatedQuestions are siblings in the precedence tree
        isAppliedTo in SecondaryMotion
                => separatedQuestions in isAppliedTo.@takesOver.@yieldsTo
        isAppliedTo in RootMotion
                =>  separatedQuestions in isAppliedTo.@dealsWith.@isDealtWithBy
        //separated questions are of same type
        haveSameType[separatedQuestions]
        //§27 Standard descriptive characteristics 1. //GOOD EXAMPLE
        //Takes precedence over the main motion and over the subsidiary motion to Postpone
indefinitely
        //If applied to an amendement it also takes precedence over that amendment
        some (isAppliedTo&Amend) =>
        takesOver in (RootMotion+PostponeIndefinitely+(isAppliedTo&Amend))

        else
        takesOver in (RootMotion+PostponeIndefinitely)
        //A motion to divide the main question cannot be made while an amendement to the main
question is pending.
```

```
	//(Meaning there is no amendement to the main question in the this ^takesOver set)
	no (takesOver.*@takesOver) & ((@isAppliedTo.(getRoot[this]))&Amend)
	//it yields to all subsidiary motions exept PstponeIndefinitely, Amend and LimitExtendDebate,
to all Privileged and to all applicable Incidental
	yieldsTo in (SubsidiaryMotion-(PostponeIndefinitely+Amend+LimitExtendDebate)
						+PrivilegedMotion
						+IncidentalMotion)
	//A motion to divide can be made at any time that the main motion , an amendment
	// which it is proposed to divide or the motion to PostponeIndefinitely is immediately pending
	//("even after the previous question has been ordered "=> unnecessary to model)
	takesOver in (RootMotion+PostponeIndefinitely+isAppliedTo&Amend)
	//§27 Standard descriptive characteristics 2.
	//Can be applied to main motions and their amendments if they are scuceptible to dvision.
//	isAppliedTo in RootMotion+((@isAppliedTo.RootMotion)&Amend)
/*	(some root : RootMotion | root in isAppliedTo) => (
		(some root : RootMotion | (root + (root.(~@isAppliedTo))&Amend) in isAppliedTo)
or//suceptible to division
		(some root : RootMotion | (root + in isAppliedTo) //not suceptible to division

)*/
	//no Subsidiary can be aplied to it except Amend and PreviousQuestion
	some (@isAppliedTo.this) => @isAppliedTo.this not in (SubsidiaryMotion - (Amend+
PreviousQuestion))
/*	//§27 Standard descriptive characteristics 1.
	//about precedence
	takesOver in (RootMotion+PostponeIndefinitely+Amend)
	some (isAppliedTo & RootMotion)
		=> no a : Amend| some(a.@isAppliedTo & RootMotion) and a in this.^@takesOver
	yieldsTo in (SubsidiaryMotion-(PostponeIndefinitely+Amend+LimitExtendDebate)
						+PrivilegedMotion
						+IncidentalMotion)
	//§27 Standard descriptive characteristics 2.
	//about application
	isAppliedTo in
		((RootMotion + Amend)&takesOver)
		+ ( ((takesOver&PostponeIndefinitely).@takesOver) & (RootMotion + Amend) )
*/
	//debate and amendment
	this not in DebatableMotion
	this in AmendableMotion
	//does not apply to siblings
	isAppliedTo in this.^@takesOver
}
fact{
//no overlapped divided questions
all disj doq1, doq2 : DivisionOfQuestion | no (doq1.separatedQuestions & doq2.separatedQuestions)
//no unchronological dependencies
//all disj doq1, doq2 : DivisionOfQuestion | not (doq1.separatedQuestions = doq2.separatedQuestions)
}
```

```
/* *
 * A terminal IncidentalMotion
 * §12 Standard descriptive characteristics 2.
 * §28 Standard descriptive characteristics
 * ** This motion is modeled as one motion with multiple debate
 * and voting rounds allowed. Its internal characteristics are
 * not modeled.
 */

sig ConsiderationByParagraph extends IncidentalMotion {
	aux : set ConsiderationByParagraph
}
{
	//precedence
	takesOver in (RootMotion+PostponeIndefinitely+Amend)
	takesOver in Amend => takesOver = isAppliedTo
	all root : RootMotion, a : Amend |
		(a in @isAppliedTo.root) => not (this in a.^@yieldsTo and this in @isAppliedTo.root)
	yieldsTo in (IncidentalMotion
						+PrivilegedMotion
						+SubsidiaryMotion
						-PostponeIndefinitely
						-Amend
						-LimitExtendDebate)
	//application
	one isAppliedTo //TODO is it always so?
	isAppliedTo in (RootMotion + Amend)
	no (@isAppliedTo.this & SubsidiaryMotion-Amend-PreviousQuestion)
	//debate and amendments
	this not in DebatableMotion
	this in AmendableMotion
	//does not apply to siblings
	isAppliedTo in this.^@takesOver
	//auxiliaries are from the same set
	//aux in ConsiderationByParagraph
}

/**
 * A terminal IncidentalMotion
 * §12 Creating a blank b) and last paragraph "When a blank exists..."
 * §12 Filling blanks, last paragraph "The particular circumstances..."
 * ** The proposals to fill a blank are considered as being part of the motion
 * to create this blank so that the motion CreateBlank is viewed as
 * one motion with multiple voting rounds allowed.
 * ** The motions to CloseSuggestion and CloseNominations will not be modeled.
 */

sig CreateBlank extends IncidentalMotion {
```

```
	aux : set CreateBlank
}
{
	this not in DebatableMotion
	this not in AmendableMotion
	//not in order when PreviousQuestion has been moved before it
	all root : RootMotion, pq : PreviousQuestion |
		((this+pq in root.^@yieldsTo) and (pq in @isAppliedTo.root))
			=> this not in pq.^@yieldsTo
	//Can apply directly or to the same motion than a primary or secondary amendment
	//§12 FIlling Blanks, CREATING A BLANK b)
	isAppliedTo in (takesOver+(takesOver&Amend).@takesOver
                                                        +
(((takesOver&Amend).@takesOver)&Amend).@takesOver)
	//cannot be applied soleley the PreviousQuestion
	//§12 FIlling Blanks, CREATING A BLANK b)
	all pq : PreviousQuestion&@isAppliedTo.this | isAppliedTo in pq.@isAppliedTo
	//does not apply to siblings
	isAppliedTo in this.^@takesOver
	//applies to one motion at a time
	one isAppliedTo
}
                                                                        /***********
*****************************
                                                                         *
TERMINAL-SUBSIDIARY-MOTIONS *

****************************************/
                                                                        pred
TERMINAL_SUBSIDIARY_MOTIONS {}
	/* *
   *An individual Subsidiary Motion
   *Rank 1
   *§6:Listing of individual subsidiary motions 1)
   */
	sig PostponeIndefinitely extends SubsidiaryMotion {
		aux : set PostponeIndefinitely
	}
	{
		//*§6:cases when one subsidiary can be applied ton another
		this not in AmendableMotion
		this in DebatableMotion
		one isAppliedTo
		//auxiliaries are from the same set
		//aux in PostponeIndefinitely
	}
	fact{

		//Natural ranking priorities
```

```
		//(sometimes also imply how the motion can be applied since
		//if a motion cannot take precedence over another it cannot be applied to it)
		no (PostponeIndefinitely.^takesOver & SubsidiaryMotion)
	}

	/**
   *An individual Subsidiary Motion
   * Rank 2
   * §6:Listing of individual subsidiary motions 2)
   * §6: cases when one subsidiary can be applied ton another =>
   * NOTE : It seems that Amend can take precedence over any motion**
   */
	abstract sig Amend extends SubsidiaryMotion {
		//concretetakesOver : set Motion
		context : set Motion+Business
	}
	{
		//auxiliary atoms : each main amend that has targets knows about its context
		let tree = getTree[this] ,
			children = this+(this.^@yieldsTo),
			targets = getTargets[this] |
				(not disable_auxiliary_atoms and hasTargets[this] and isMain[this])
=> //TODO : refactor: put this on top?
					context = tree - children +targets
				else no context
		//context elements must have auxiliaries
		all elem : context | some elem.allAux
		//concretetakesOver = skipAmend[this, Motion.takesOver]
		this in AmendableMotion
		one isAppliedTo
		//Amend applies only to Amendable motions and RootMotions
		//§6: cases when one subsidiary can be applied ton another
		isAppliedTo in (RootMotion + AmendableMotion)
		// Motion that, due to their characteristics and particularities, end up
		// not yielding to Amend. See:
		// §16 Standard descriptive characteristics 1)
		no (takesOver & (PreviousQuestion + LayOnTheTable))
		//no (concretetakesOver & (PreviousQuestion + LayOnTheTable))
		//§12 Standard descriptive characteristics 1. a)
		//rule when amend is applied to a RootMotion
		some (isAppliedTo & RootMotion)
			=> takesOver in (RootMotion + PostponeIndefinitely)
			  and yieldsTo in
			  (SubsidiaryMotion-PostponeIndefinitely+PrivilegedMotion+IncidentalMotion)
			  and (some yieldsTo & Amend => (all a :yieldsTo & Amend | this in
a.@isAppliedTo))
		/* §12 Standard descriptive characteristics 1. b) */
		//rule when amend is applied to a secondary motion
		some (isAppliedTo & SecondaryMotion)
```

```
                => takesOver = isAppliedTo
                        and yieldsTo in (PrivilegedMotion+SubsidiaryMotion+IncidentalMotion)
                        and ((some (yieldsTo & Amend+LimitExtendDebate+PreviousQuestion))
                                => (all m : (yieldsTo &
Amend+LimitExtendDebate+PreviousQuestion)| this in m.@isAppliedTo))
                        //draft, see §12 Standard descriptive characteristics 1. b)
                        //and (some atoms : set univ | isAppliedTo.yieldsTo in atoms
                        //                              => yieldsTo in atoms)
            /*  §12 Standard descriptive characteristics 2. */
            //what can apply to amend
            @isAppliedTo.this in (Amend
                                                +LimitExtendDebate
                                                +PreviousQuestion
                                                +DivisionOfQuestion
                                                +ConsiderationByParagraph
                                                +CreateBlank)
            no (@isAppliedTo.this & (Commit
                                                +Postpone
                                                +LayOnTheTable
                                                +PostponeIndefinitely))
            /*****BINGO1?******/
            all m : (@isAppliedTo.this) & (LimitExtendDebate+PreviousQuestion) |
                                this in m.@takesOver // BINGO? : for the PreviousQuestion to
applly to Amend, Amend must be immediately pending
            /*****BINGO1?******/            /*  §12 Standard descriptive characteristics 6. */
            //secondary amendments cannot be amended
            some (isAppliedTo & Amend) => no ( @isAppliedTo.this & Amend)
        }

        /**
    * Debatable Amend for debatables
    * §6: cases when one subsidiary can be applied ton another
    * Rank 2
    */
        sig AmendDebatable extends Amend {
            aux : set AmendDebatable
        }
        {
            isAppliedTo in DebatableMotion
            this in DebatableMotion
            //auxiliaries are from same set
            //aux in AmendDebatable
        }

        /**
    * Undebatable Amend for undebatables
    * §6: cases when one subsidiary can be applied ton another
    * Rank 2
    */
```

```
	sig AmendUndebatable extends Amend {
		aux : set AmendUndebatable
	}
	{
		no (isAppliedTo & DebatableMotion) //isAppliedTo not in DebatableMotion
		this not in DebatableMotion
		//auxiliaries are from same set
		//aux in AmendUndebatable

	}

	/**
    * An individual Subsidiary Motion
    * §6:Listing of individual subsidiary motions 3)
    * Rank 3
    */
	sig Commit extends SubsidiaryMotion {
		aux : set Commit
	}
	{
		this in AmendableMotion
		this in DebatableMotion
		one isAppliedTo
		//§13 Standard Descriptive Characteristics 2.
		//cannot be applied to any subsidiary motion
		no isAppliedTo&SubsidiaryMotion
	}
	fact{
		//natural ranking priorities over other subsidiary motions
		//(sometimes also imply how the motion can be applied since
		//if a motion cannot take precedence over another it cannot be applied to it)
		no (Commit.^(takesOver)
			&  (LayOnTheTable + PreviousQuestion + LimitExtendDebate + Postpone +
Commit ))
	}

	/**
    *An individual Subsidiary Motion
    *§6:Listing of individual subsidiary motions 3)
    *Rank 4
    */
	sig Postpone extends SubsidiaryMotion {
		aux : set Postpone
	}
	{
		this in AmendableMotion
		this in DebatableMotion
		one isAppliedTo
		//§13 Standartd DEscriptive caracteristics 2.
```

```
		isAppliedTo in RootMotion
		//auxiliaries are from same set
		//aux in Postpone
		//movedOnce[this]
		//appliesOnce[this]
	}
	fact{

		//natural ranking priorities over other subsidiary motions
		//(sometimes also imply how the motion can be applied since
		//if a motion cannot take precedence over another it cannot be applied to it)
		no (Postpone.^takesOver
			&  (LayOnTheTable + PreviousQuestion + LimitExtendDebate + Postpone))
	}

	/**
*A individual Subsidiary Motion
*§6:Listing of individual subsidiary motions 5)
*Rank 5
*/
	sig LimitExtendDebate extends SubsidiaryMotion {
		aux : set LimitExtendDebate
	}
	{
		this in AmendableMotion
		this not in DebatableMotion
		//can apply to a series
		some isAppliedTo
		//§15 Standard descriptive characteristics 2. The immediatePending question should
		// be part of the motion this is applied to
		/*****BINGO1?******/
		takesOver in isAppliedTo
		/*****BINGO1?******/
		//auxiliaries are from same set
		//aux in LimitExtendDebate
		//movedOnce[this]
		//appliesOnce[this]
	}
	fact{
		//natural ranking priorities over other subsidiary motions
		//(sometimes also imply how the motion can be applied since
		//if a motion cannot take precedence over another it cannot be applied to it)
		no (LimitExtendDebate.^takesOver
			&  (LayOnTheTable + PreviousQuestion + LimitExtendDebate))
	}

	/**
* An individual Subsidiary Motion
* §6:Listing of individual subsidiary motions 6)
```

```
* Rank 6
*/
    sig PreviousQuestion extends SubsidiaryMotion {
            aux : set PreviousQuestion
    }
    {
            //§6: cases when one subsidiary can be applied ton another
            this not in AmendableMotion
            this not in DebatableMotion
            some isAppliedTo
            //§16 Standard descriptive characteristics 2. The immediatePending question should
            // be part of the motion this is applied to
            /*****BINGO1?******/
            takesOver in isAppliedTo
            /*****BINGO1?******/
    }
    fact{
            //natural ranking priorities over other subsidiary motions
            //(sometimes also imply how the motion can be applied since
            //if a motion cannot take precedence over another it cannot be applied to it)
            no (PreviousQuestion.^takesOver
                    &  (LayOnTheTable + PreviousQuestion ))
    }

    /**
* An individual Subsidiary Motion
* §6:Listing of individual subsidiary motions 7)
* Rank 7
*/
    sig LayOnTheTable extends SubsidiaryMotion {
            aux  : set LayOnTheTable
    }
    {
            //*§6:cases when one subsidiary can be applied ton another
            this not in AmendableMotion
            this not in DebatableMotion
            //§7 : Standard descriptive characteristics 1.
            isAppliedTo in RootMotion
            //auxiliaries are from same set
            //aux in LayOnTheTable
            //movedOnce[this]
            //appliesOnce[this]
    }
    fact{
            //natural ranking priorities over other subsidiary motions
            //(sometimes also imply how the motion can be applied since
            //if a motion cannot take precedence over another it cannot be applied to it)
            no (LayOnTheTable.^takesOver &  LayOnTheTable)
    }
```

```
                                                                    /***********
*********************************
                                                                     *
TERMINAL-PRIVILEDGED-MOTIONS     *

*********************************************/
                                                                    pred
TERMINAL_PRIVILEDGED_MOTIONS {}

        /**
         * Rank 1
         */
        sig CallForOrdersOfDay extends PrivilegedMotion {
                aux : set CallForOrdersOfDay
        }
        {
                this not in DebatableMotion
                this not in AmendableMotion
                //auxiliaries are from same set
                //aux in CallForOrdersOfDay
                //movedOnce[this]
                //appliesOnce[this]
        }
        fact {
                //natural ranking priorities over other privileged motions
                no (CallForOrdersOfDay.^(takesOver) & PrivilegedMotion)
        }

        /**
         * Rank 2
         * NOTE :a privileged motion can be moved like a main motion.
         *                 §6: Incidental main motions corresponding to privileged motions
         *                 §19 Raise a question of privilege
         *             §19 steps in raising and disposing of a question of privilege
         */
        sig RaiseQuestionOfPrivilege extends PrivilegedMotion {
                questionOfPrivilege : one MainMotion, // **(can also be a request)//TODO is this a s.h.?
                aux : set RaiseQuestionOfPrivilege
        }
        {
                //The question of privilege dealsWith the same business this motion is on top of
                some (questionOfPrivilege.dealsWith)&(this.(^@takesOver).dealsWith)&(Business)
                //the questionOfPrivilege is a new question
                no (questionOfPrivilege&this.(^@takesOver))
                this not in DebatableMotion
                this not in AmendableMotion
                //questions of privilege do not overlap
                lone @questionOfPrivilege.questionOfPrivilege
```

```
	}
	fact{
		//natural ranking priorities over other privileged motions
		no (RaiseQuestionOfPrivilege.^(takesOver) & (RaiseQuestionOfPrivilege + Recess +
Adjourn + FixTimeAdjourn))
	}
	/**
	 * Rank 3
	 */
	sig Recess extends PrivilegedMotion {
		aux : set Recess
	}
	{
		this not in DebatableMotion
		this in AmendableMotion
		//auxiliaries are from same set
		//aux in Recess
		//movedOnce[this]
		//appliesOnce[this]
	}
	fact {
		//natural ranking priorities over other privileged motions
		no (Recess.^(takesOver) & (Recess + Adjourn + FixTimeAdjourn))
	}

	/**
	 * Rank 4
	 */
	sig Adjourn extends PrivilegedMotion {
		aux : set Adjourn
	}
	{
		this not in DebatableMotion
		this not in AmendableMotion
		//auxiliaries are from same set
		//aux in Adjourn
		//movedOnce[this]
		//appliesOnce[this]
	}
	fact {
		//natural ranking priorities over other privileged motions
		no (Adjourn.^(takesOver) & (Adjourn + FixTimeAdjourn))
	}

	/**
	 * Rank 5
	 * Can take precedence over anything
	 */
	sig FixTimeAdjourn extends PrivilegedMotion {
```

```
		aux : set FixTimeAdjourn
	}
	{
		this not in DebatableMotion
		this in AmendableMotion
		//auxiliaries are from same set
		//aux in FixTimeAdjourn
		//movedOnce[this]
		//appliesOnce[this]
	}
	fact {
		//natural ranking priorities over other privileged motions
		no (FixTimeAdjourn.^(takesOver) & (FixTimeAdjourn))
	}

/************************
*********
 * AUXILIARY
FUNCTIONS     *

*********************************/
pred
AUXILLIARY_FUNCTIONS {}

	fun pendingStack(ss : SessionState): set Motion {
		(ss.immediatePendingQuestion + ss.immediatePendingQuestion.^(takesOver))
	}

	pred movedOnce[ motions : set univ] {
		no disj m1, m2 : motions |
			(some m1.takesOver => m1.takesOver = m2.takesOver) and
			(some m1.isAppliedTo => m1.isAppliedTo = m2.isAppliedTo)
	}

	fun firstSessionState : one SessionState{
		SessionState - (SessionState & prev.SessionState)
	}

	fun lastSessionState : one SessionState{
		SessionState - (SessionState & next.SessionState)
	}

	fun primaryAmendments : set Amend {
		Amend.isAppliedTo & Amend
	}

	fun secondaryAmendments : set Amend {
		isAppliedTo.Amend & Amend
	}
```

```
	pred isAuxiliary[atom : univ] {
		// An atom isAuxiliary when it is pointed at
		some  allAux.atom
	}

	fun sessionStateElements : set univ {
	//A sessionStateElement is one reachable from some SessionState
			SessionState.orderOfBusiness
		+SessionState.orderOfBusiness.isDealtWithBy.*yieldsTo
//		+	SessionState.toComeToOrder
//		+SessionState.toComeToOrder.^yieldsTo

	}

	pred isMain [atom : univ]  {
		not isAuxiliary[atom]
   }

	fun nonSessionStateElements : set univ {
		(Motion + Business) - sessionStateElements
  }


	fun allAux : (Motion+Business) -> (Motion+Business) {
		( Business <: aux) +
		( MainMotion <: aux) +
//		( OriginalMainMotion <: aux) +
//		( IncidentalMainMotion <: aux) +
//		( BQAAMotion<: aux) +
		( DivisionOfQuestion <: aux) +
		( ConsiderationByParagraph<: aux) +
		( CreateBlank <: aux) +
		( PostponeIndefinitely <: aux) +
		( AmendDebatable<: aux) +
		( AmendUndebatable <: aux) +
		( Commit <: aux) +
		( Postpone<: aux) +
		( LimitExtendDebate<: aux) +
		( PreviousQuestion<: aux) +
		( LayOnTheTable<: aux) +
		( CallForOrdersOfDay<: aux) +
		( RaiseQuestionOfPrivilege <: aux) +
		( Recess<: aux) +
		( Adjourn<: aux) +
		( FixTimeAdjourn <: aux)
	}

	pred haveSameType [ motions : Motion ] {
```

```
//                motions in OriginalMainMotion  or
//                motions in IncidentalMainMotion  or
                  motions in MainMotion  or
//                motions in BQAAMotion or
                  motions in DivisionOfQuestion  or
                  motions in ConsiderationByParagraph or
                  motions in CreateBlank  or
                  motions in PostponeIndefinitely  or
                  motions in AmendDebatable or
                  motions in AmendUndebatable  or
                  motions in Commit  or
                  motions in Postpone or
                  motions in LimitExtendDebate or
                  motions in PreviousQuestion or
                  motions in LayOnTheTable or
                  motions in CallForOrdersOfDay or
                  motions in RaiseQuestionOfPrivilege  or
                  motions in Recess or
                  motions in Adjourn or
                  motions in FixTimeAdjourn
  }

        fun getTree [a : Motion+Business ] : set Motion+Business {
                ((
                (a.(^takesOver))
                                                .dealsWith)
                                                        .isDealtWithBy)
                                                                .(^yieldsTo+dealsWith+iden)
        }

/***
 * A 'target' of a motion to amend is a child that
 * applies to a parent of this amend
 */
pred hasTargets[a : Amend] {

        some (a.yieldsTo.isAppliedTo)&(a.^takesOver)
}
/**
*returns motions eventualy making this Amend transparent
*/
fun getTargets[a : Amend] : set SecondaryMotion {
        (isAppliedTo.( (a.yieldsTo.isAppliedTo)&(a.^takesOver) )) & (a.yieldsTo )
}

fun getBusiness[motion : Motion] : one Business{
        (motion.^takesOver.dealsWith)+(motion.dealsWith)
}
```

```
/**
 * All motions motion applies to and
 * other concerned motions when motion is applied to a RootMotion
 */
fun getAppliedTo[motion : SecondaryMotion] : set Motion {
                                                motion.isAppliedTo
                                                + ((RootMotion & motion.isAppliedTo).^yieldsTo) &
motion.^takesOver
}

/**
*
*/
fun getSiblings[motion : Motion] : set Motion {
        (motion.(dealsWith+takesOver).(isDealtWithBy+yieldsTo))-motion
}

fun getRoot[motion: Motion] : lone RootMotion {
        (motion.*takesOver)&RootMotion
}
                                                                                        /***********
*
                                                                                         * TRACES *

************/
                                                                                pred TRACES {}

        pred initialSession[firstState : SessionState]{
                //A session starts with no questions being pending**
                no firstState.pendingQuestions
                // the session starts with a business being current
                some firstState.currentBusiness
                //nothing is adopted**
                no firstState.adopted
                //A session starts with no questions being under some order**
                no firstSessionState.
(underCommitOrPostponeIndefinitelyOrder+laidOnTheTable+underPreviousQuestionOrder)
                //the eventually separated motions and the auxiliary motions are not toComeToOrder
                //every other motion is toComeToOrder
//max : all (DivisionOfQuestion.isAppliedTo) are in tCO
                firstState.toComeToOrder =
                        (firstState.orderOfBusiness.isDealtWithBy.*yieldsTo)
                  -
                        (       ((DivisionOfQuestion.separatedQuestions-
(DivisionOfQuestion.isAppliedTo)).*yieldsTo)
                    +(RaiseQuestionOfPrivilege.questionOfPrivilege.*yieldsTo)
                        )
/*//min : no (DivisionOfQuestion.isAppliedTo) is in tCO
                        (firstState.orderOfBusiness.isDealtWithBy.*yieldsTo)
```

```
                -
                        (       ((DivisionOfQuestion.separatedQuestions).*yieldsTo)
                    +(RaiseQuestionOfPrivilege.questionOfPrivilege.*yieldsTo)
                        ) in firstState.toComeToOrder
//constraint about separatedQuestions
all dq1, dq2 : DivisionOfQuestion | (some dq1.separatedQuestions&dq2.separatedQuestions)=>(
                        one
((dq1.separatedQuestions&dq2.separatedQuestions)&(firstState.toComeToOrder)&((dq1+dq2).isAppli
edTo))
                        and
((dq1.separatedQuestions&dq2.separatedQuestions)&(firstState.toComeToOrder)).^yieldsTo in
firstState.toComeToOrder
                        )
*/
        }

        pred finalSession[lastState : SessionState]{
                //A session ends with no questions being pending
//              no (lastState.pendingQuestions)&RootMotion
                //An ended session has no more motions to come to order
                //activating this makes the predicate become equivalent to reach_all_motions
//              no lastState.toComeToOrder
                //A session is over when there is no more current businesses;
                //because of the implementation the disposeOf action this means
                //that the last business has been dealt with in the previous state
                no (lastState.currentBusiness)
        }

        fact sessionTrace{
//                      initialSession[firstSessionState]//[sessionOrd/first]
                all sBefore : SessionState - lastSessionState | //sessionOrd/last |
                        let sAfter  = sBefore.next |
                                (       one motion : Motion |
                                        move[sBefore, sAfter, motion] or   //TODO : does the order of
these matter?
                                        adopt[sBefore, sAfter, motion] or
                                        disposeOf[sBefore, sAfter, motion ])

//                      finalSession[lastSessionState] //[sessionOrd/last]
        }

                                                                                /***********
**********
                                                                                 * BASIC
ACTIONS *

**********************/
                                                                        pred
BASIC_ACTIONS {}
```

```
	pred move_preconditions[sBefore : SessionState, motion : Motion ]{
			//if the RootMotion of the before state is under some order then move is false
//TODO :remove
			no ( (sBefore.underCommitOrPostponeIndefinitelyOrder)
					& RootMotion)
			no ( (sBefore.laidOnTheTable)
					& RootMotion)
			//if the ipq of the before state is under PreviousQuestion order then move is false
for subsidiary Motions
			//exception made for LayOnTheTable §16 PreviousQuestion 1) and 2)
		(motion in (SubsidiaryMotion-LayOnTheTable))=>
			no (sBefore.immediatePendingQuestion
				& (sBefore.underPreviousQuestionOrder))
			//if the ipq of the before state is adopted move is false (proposals to fill blanks are
not modelled)
			//§4 The handeling of a motion , chair's announcement of the voting result;
verifiaction procedures
			//and cases where the chair votes.
			no (sBefore.immediatePendingQuestion & sBefore.adopted)
			//if there is some separatedQuestions still to come to order among the before state
			//ipq's children then the moved motion must be one of them.
			//§27 Division of the Question, Form and example
			some (sBefore.immediatePendingQuestion.yieldsTo)
				& (DivisionOfQuestion.separatedQuestions)
				& (sBefore.toComeToOrder)
				=> motion in (sBefore.immediatePendingQuestion.yieldsTo)
							& (DivisionOfQuestion.separatedQuestions)
							& (sBefore.toComeToOrder)
		motion in sBefore.toComeToOrder
		motion in RootMotion => motion.dealsWith = sBefore.currentBusiness
	}
	/* *
	 * A non second requiring motion is moved by a member who has
	 * obtained the floor for it to become a pending question.
	 * It doesn't need to be seconded by another member to become pending.
   */
	pred move[sBefore, sAfter:SessionState, motion : Motion]{
		/** initial situation **/
		move_preconditions[sBefore, motion]
		/** situation after the action **/
		motion = sAfter.immediatePendingQuestion
		motion.takesOver = sBefore.immediatePendingQuestion
		sAfter.pendingQuestions = sBefore.pendingQuestions + motion
		/** everything else stays the same **/
		sAfter.toComeToOrder = sBefore.toComeToOrder - motion
		sAfter.adopted = sBefore.adopted
		sAfter.orderOfBusiness = sBefore.orderOfBusiness
```

```
		sAfter.currentBusiness = sBefore.currentBusiness
		sAfter.underCommitOrPostponeIndefinitelyOrder =
sBefore.underCommitOrPostponeIndefinitelyOrder
		sAfter.underPreviousQuestionOrder = sBefore.underPreviousQuestionOrder
		sAfter.laidOnTheTable = sBefore.laidOnTheTable
	}

	pred adopt_preconditions[sBefore : SessionState, motion: Motion]{
		motion = sBefore.immediatePendingQuestion
		motion not in sBefore.adopted
			//adopt is false when there is some separated motion still toComeToOrder
			//among the children of the to be adopted motion
		no (sBefore.immediatePendingQuestion.yieldsTo)
			& (DivisionOfQuestion.separatedQuestions)
			& (sBefore.toComeToOrder)
	}

	/**
	 * A motion has to be adopted in order to take effect
	 * either by :
	 * - a vote §4:Putting the question
	 * - directly because of the motion's nature
	 * - upon a ruling of the chair
	 */
	pred adopt[sBefore, sAfter:SessionState, motion : Motion] {
		/** initial situation **/
		adopt_preconditions[sBefore, motion]
		/** situation after the action **/
		motion in sAfter.adopted
			//If the ipq is Postpone or Commit then the motions they are
			//applied to are in the underCommitOrPostponeIndefinitelyOrder set of the after
sessionState
			//if the motion is applied to a RootMotion then all the motions between the
			//root and the motion are under the former order too.
		motion in (PostponeIndefinitely + Commit)
				=> (sAfter.underCommitOrPostponeIndefinitelyOrder =
sBefore.underCommitOrPostponeIndefinitelyOrder + getAppliedTo[motion] and
					sAfter.underPreviousQuestionOrder =
sBefore.underPreviousQuestionOrder - getAppliedTo[motion] )
				else(sAfter.underCommitOrPostponeIndefinitelyOrder =
sBefore.underCommitOrPostponeIndefinitelyOrder and
					//If the ipq is the PreviousQuestion then the motions it is
					//applied to are in the underPreviousQuestionOrder set of
the after sessionState
					//if the motion is applied to a RootMotion then all the
motions between the
					//root and the motion are under the former order too.
					(motion in (PreviousQuestion) //NOTE : maybe useless
putting both together
```

```
                                                    => sAfter.underPreviousQuestionOrder
                                                                =
(sBefore.underPreviousQuestionOrder + getAppliedTo[motion])
                                                    else(
                          (motion in (DivisionOfQuestion) and some
motion.isAppliedTo&sBefore.underPreviousQuestionOrder)
                    => sAfter.underPreviousQuestionOrder =
sBefore.underPreviousQuestionOrder+motion.separatedQuestions
                                                        else (sAfter.underPreviousQuestionOrder =
sBefore.underPreviousQuestionOrder)
                          )
                                                )
                                               )
                    //If the ipq is LayOnTheTable then the motions they are
                    //applied to are in the laidOnTheTable set of the after sessionState
                    //if the motion is applied to a RootMotion then all the motions between the
                    //root and the motion are under the former order too.
                    motion in (LayOnTheTable)
                              => sAfter.laidOnTheTable     = (sBefore.laidOnTheTable

                         + motion.isAppliedTo //all parents if applied to the root

                         + ((RootMotion & motion.isAppliedTo).^yieldsTo) & motion.^takesOver)
                              else (sAfter.laidOnTheTable = sBefore.laidOnTheTable)
                    //if the motion adopted is RaiseQuestionOfPrivilege the question of privilege raised
                    //becomes immediately pending //TODO ref
                    motion in (RaiseQuestionOfPrivilege)
                              => (sAfter.immediatePendingQuestion = motion.questionOfPrivilege and
                                        sAfter.pendingQuestions =
sBefore.pendingQuestions+motion.questionOfPrivilege and
                                                  sAfter.toComeToOrder =
sBefore.toComeToOrder+motion.questionOfPrivilege.yieldsTo) and
                                                  sAfter.currentBusiness = sBefore.currentBusiness
                              else(
                         //if the motion adopted id Division of the Question then all the resulting new
questions
                         //should eventually come to order §27 Division of the Question, Form and
example
        motion in DivisionOfQuestion
                                    => sAfter.toComeToOrder =
                                    (sBefore.toComeToOrder
                                     + (motion.separatedQuestions- motion.isAppliedTo)
                                     + (
                                              ( motion.separatedQuestions- motion.isAppliedTo).^yieldsTo
                                             - ( (DivisionOfQuestion-motion).separatedQuestions -
(DivisionOfQuestion - motion).isAppliedTo )
                                          )
         ) and
              sAfter.immediatePendingQuestion = sBefore.immediatePendingQuestion and
```

```
                                    sAfter.pendingQuestions = sBefore.pendingQuestions and
                                    sAfter.currentBusiness = sBefore.currentBusiness
        else(
                            //Limited effect of adopting CallForOrdersOfDay
                            //(a skipped business or one due a this time).
                                        (motion in CallForOrdersOfDay
                                        => (
                                                sAfter.currentBusiness =
sBefore.currentBusiness and
                                          sAfter.toComeToOrder =
sBefore.toComeToOrder and
                                                sAfter.pendingQuestions =
sBefore.pendingQuestions-motion
                                                    )
                                        else(
                                                    //limited effect of adopting
lay on the table+ never taken back again
                                                    motion in LayOnTheTable
                                                    => (

        sAfter.currentBusiness = none
                                                    and no
sAfter.immediatePendingQuestion
                                                            and
sAfter.toComeToOrder = sBefore.toComeToOrder
                                                            and
sAfter.pendingQuestions = sBefore.pendingQuestions - motion

                - motion.isAppliedTo //all parents if applied to the root

                - (((RootMotion & motion.isAppliedTo).^yieldsTo) & motion.^takesOver)
                                    )
                                                    else(
                                                        //limited effect of
adopting Postpone
                                                        motion in Postpone
                                                        =>(

        sAfter.toComeToOrder = sBefore.toComeToOrder and

        sAfter.currentBusiness = sBefore.currentBusiness
                                                                and
sAfter.pendingQuestions = sBefore.pendingQuestions-motion
                                                                    )
                                                        else (

        (motion in (Commit+PostponeIndefinitely)

        => (
```

```
                    sAfter.toComeToOrder = sBefore.toComeToOrder and
                    sAfter.currentBusiness = none and
                    sAfter.pendingQuestions = sBefore.pendingQuestions - motion -
motion.isAppliedTo
                                                                    - (((RootMotion &
motion.isAppliedTo).^yieldsTo) & motion.^takesOver)
                      and no sAfter.immediatePendingQuestion
                )
        else( sAfter.toComeToOrder = sBefore.toComeToOrder and
                    sAfter.immediatePendingQuestion = sBefore.immediatePendingQuestion
and
                sAfter.currentBusiness = sBefore.currentBusiness and
                    sAfter.pendingQuestions = sBefore.pendingQuestions
                )
        )
                                                                        )
                                                                    )
                                                                )
           )
          )
         )
            //everything else stays the same
            sAfter.orderOfBusiness = sBefore.orderOfBusiness
            sAfter.adopted = sBefore.adopted + motion
        }

        pred disposeOf_preconditions [sBefore: SessionState, motion : Motion] {
            /**initial situation **/
            //disposeOf is true only if the motion is immediately pending //TODO ref
            motion = sBefore.immediatePendingQuestion
                //disposeOf is false when there is some separated motion still toComeToOrder
among the
                //children of the to be disposedOf motion  §27 Division of the Question, Form
and example
            no (sBefore.immediatePendingQuestion.yieldsTo)
                & (DivisionOfQuestion.separatedQuestions)
                & (sBefore.toComeToOrder)
```

```
	}

	/**
	 * A motion must be disposed of before
	 * the next business or motion can be
	 * dealtWith.
	 * note : this action lets the definition
	 * of the ipq determine what will be the
	 * new ipq. The new ipq is thus not described
	 * explicitly here.
	 */
	pred disposeOf[sBefore, sAfter :SessionState, motion : Motion]{
		/** initial situation **/
		disposeOf_preconditions[sBefore, motion]
		/** situation after the action **/
		motion not in sAfter.pendingQuestions //TODO redundant ?
			//whatever we are taking out of the pendingQuestions set,
			//if it was under some order then it shouldn't be anymore //TODO : this is never used for c, pi and lat since order took effect immediately
			//§16 Previous Question
		motion not in (sAfter.underCommitOrPostponeIndefinitelyOrder
					+sAfter.underPreviousQuestionOrder
					+sAfter.laidOnTheTable)
			//if an adopted division of the question is disposed of
			//all motions between it and its divided questions should be lost//TODO : ref
		(motion in DivisionOfQuestion and motion in sBefore.adopted and not motion.isAppliedTo = motion.takesOver )
				=> (sAfter.pendingQuestions = sBefore.pendingQuestions

				-motion

				-(motion.takesOver)

				-(motion.^takesOver) + ((motion.isAppliedTo).*takesOver)) and
					sAfter.immediatePendingQuestion = motion.isAppliedTo
				else (
		motion in (RaiseQuestionOfPrivilege.questionOfPrivilege)
				=> (sAfter.pendingQuestions = sBefore.pendingQuestions-(motion+questionOfPrivilege.motion)and
					sAfter.immediatePendingQuestion = questionOfPrivilege.motion.takesOver)
			else (sAfter.pendingQuestions = sBefore.pendingQuestions-motion and
					 sAfter.immediatePendingQuestion = sBefore.immediatePendingQuestion.takesOver)
					)
		//transition from one business to another NOTE: since the last has no next currentBusiness will be empty
		motion in RootMotion =>(
			sAfter.currentBusiness = sBefore.currentBusiness
```

```
                                                                                                                    )
                else (    sAfter.currentBusiness = sBefore.currentBusiness)
                //everything else stays the same
                sAfter.orderOfBusiness = sBefore.orderOfBusiness
                sAfter.adopted = sBefore.adopted
                sAfter.toComeToOrder = sBefore.toComeToOrder
                sAfter.underCommitOrPostponeIndefinitelyOrder =
sBefore.underCommitOrPostponeIndefinitelyOrder-motion
                sAfter.underPreviousQuestionOrder = sBefore.underPreviousQuestionOrder-motion
                sAfter.laidOnTheTable = sBefore.laidOnTheTable-motion
        }


        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]
                                                                                        )
        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 othe ratributes that are not specified by the final state
        sAfter.immediatePendingQuestion = sBefore.immediatePendingQuestion
        sAfter.orderOfBusiness = sBefore.orderOfBusiness
        sAfter.adopted = sBefore.adopted
        sAfter.toComeToOrder = sBefore.toComeToOrder
        sAfter.underCommitOrPostponeIndefinitelyOrder =
sBefore.underCommitOrPostponeIndefinitelyOrder
        sAfter.underPreviousQuestionOrder = sBefore.underPreviousQuestionOrder
        sAfter.laidOnTheTable = sBefore.laidOnTheTable
        }

/************
 * SHOW   *
 ************/
pred SHOW {}

        pred show {
                //not disable_auxiliary_atoms
                //disable_auxiliary_atoms
                //whole_session
                session_prefix
                //session_suffix
                //enable_movedOnce
                //reach_all_motions
                //no ErrorFlag
```

```
        }

run show for 10 but 5 Int

run show for 1 but   8 SessionState, 5 Int
run show for 2 but   8 SessionState,5 Int
run show for 3 but   8 SessionState,5 Int
run show for 4 but   8 SessionState,5 Int
run show for 5 but   8 SessionState,5 Int
run show for 6 but   8 SessionState,5 Int
run show for 7 but   8 SessionState,5 Int
run show for 8 but   8 SessionState,5 Int
run show for 9 but   8 SessionState,5 Int
run show for 10 but   8 SessionState,5 Int

//---------------DRAFT-TESTS--------------//

module tests_model0_properties

/***************
Chose model
***************/
open model0
--open model1

/*********************************
 *         PROPERTY VERIFICATION *
 *********************************/

                                        /*********************************
                                        *CAN BE ENABLED/DISABLED *
                                        *********************************/
                                        fact CAN_BE_ENABLED_DISABLED {
                                        //disable_auxiliary_atoms
                                        // movedOnce
                                        // whole_session
                                        // session_prefix
                                        // session_suffix
                                        // reach_all_motions
                                        dynamic_mode_only
                                        // no ErrorFlag
                                        }

/******************
 *         No dedlock *
 ******************/

/** should succeed **/
assert no_error_flag_in_whole_sessions {
```

```
	whole_session => no ErrorFlag
}
check no_error_flag_in_whole_sessions for 5 but 7 SessionState
expect 0

/** should succeed **/
assert no_error_flag_in_suffixe_sessions {
	session_suffix => no ErrorFlag
}
check no_error_flag_in_suffixe_sessions for 5 but 7 SessionState
expect 0

/*********************************
 *	No unfinished business *
 *********************************/

/** should succeed **/
assert permissive_no_pending_remain_when_session_is_over{
	(encourage_move and no ErrorFlag and whole_session) => no
lastSessionState.pendingQuestions
}
check permissive_no_pending_remain_when_session_is_over for 7

/** should succeed **/
assert excessive_no_pending_remain_when_session_is_over{
	((no SessionState.adopted&Postpone) and
	encourage_move and no ErrorFlag and whole_session) => no
lastSessionState.pendingQuestions
}
check excessive_no_pending_remain_when_session_is_over for 7

/*****************************************
 *	No impeached legitimate action *
 *****************************************/

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
/***/

/** should both succeed **/
pred some_motion_is_adoptable_for_a_given_session_configuration_1 {
	some s1, s2 : SessionState , motion : /*Chose motion here*/ Amend |
		configuration[s1] and adopt[s1, s2, motion]
}
run some_motion_is_adoptable_for_a_given_session_configuration_1 for 7 but exactly 7 SessionState

pred some_motion_is_adoptable_for_a_given_session_configuration_2 {
	some s1, s2 : SessionState , motion : /*Chose motion here*/ Amend |
		configuration[s1] and not adopt[s1, s2, motion]
}
run some_motion_is_adoptable_for_a_given_session_configuration_2 for 7 but exactly 7 SessionState
/***/

/** should both succeed **/
pred some_motion_is_disposable_for_a_given_session_configuration_1 {
	some s1, s2 : SessionState , motion : /*Chose motion here*/ Amend |
		configuration[s1] and disposeOf[s1, s2, motion]
}
run some_motion_is_disposable_for_a_given_session_configuration_1 for 7 but exactly 7 SessionState

pred some_motion_is_disposable_for_a_given_session_configuration_2 {
	some s1, s2 : SessionState , motion : /*Chose motion here*/ Amend |
		configuration[s1] and not disposeOf[s1, s2, motion]
}
run some_motion_is_disposable_for_a_given_session_configuration_2 for 7 but exactly 7 SessionState
/***/

/**
* should succeed
* WARNING ! But there shouldn't be globally one complete session among all the complete sessions
* where all the amendments can be made,
* there should be one complete session per group of similar sessions !
*/
pred a_session_where_all_roots_can_be_amended {
	//if we want to be able to amend all the root motion  and end the session and reach all the
amendments
	(all root : RootMotion&AmendableMotion | some isAppliedTo.root&Amend)
	and whole_session
	and Amend in SessionState.pendingQuestions
}
```

```
run a_session_where_all_roots_can_be_amended for 5 but 8 SessionState


/*****************************************************
 *         No static-dynamic description incoherence *
 *****************************************************/
assert permissive_all_motions_are_reached{
          (no ErrorFlag and
          encourage_move and
          whole_session) => reach_all_motions
}
check permissive_all_motions_are_reached for 7


assert all_motions_are_reached{
         ((one Business => no CallForOrdersOfDay) and
          no ErrorFlag and
          encourage_move and
          whole_session) => reach_all_motions
}
check all_motions_are_reached for 7

/****************************************/

pred aberr {
         (some disj rqp1,rqp2 : RaiseQuestionOfPrivilege, disj qp1, qp2:
(rqp1+rqp2).questionOfPrivilege, b: Business|
   rqp1.questionOfPrivilege = qp2 and
   rqp2.questionOfPrivilege = qp1 and
   (qp1+qp2).dealsWith = b)
}
run aberr for 5 but 7 SessionState
expect 1
```