```
module tests_model1_options

/****************
 chose model
****************/
open model1
//open model0

                                /*****************************************************
*******
                                *       CAN BE ENABLED/DISABLED:
                                *

        ************************************************************/
                                fact CAN_BE_ENABLED_DISABLED {
                                // disable_auxiliary_atoms
                                // enable_movedOnce
                                // whole_session
                                 session_prefix
                                // session_suffix
                                // reach_all_motions
                                // dynamic_mode_only
                                        no ErrorFlag
                                }

                                /*****************************
                                *          OPTIONS TESTING     *
                                *****************************/

/*************************
 ENABLE MOVED ONCE
*************************/


pred show_enable_moved_once1 {
        no Amend
        some disj sm1, sm2 : SecondaryMotion |
                                haveSameType[sm1+sm2] and
                                sm1.isAppliedTo = sm2.isAppliedTo and
                                sm1.takesOver = sm2.takesOver and
                                (sm1).isAppliedTo in RootMotion and
                                (sm2).isAppliedTo in RootMotion and
                                some sm1.yieldsTo and some sm2.yieldsTo
}
run show_enable_moved_once1 for 5 but exactly 1 RootMotion, exactly 4 SecondaryMotion
expect 1

pred show_enable_moved_once2 {
        no Amend
```

```
        some sm1: SecondaryMotion |
                                sm1.isAppliedTo in RootMotion and
                                not lone sm1.yieldsTo
}
run show_enable_moved_once2 for 5 but exactly 1 RootMotion, exactly 3 SecondaryMotion
expect 1

pred show_particular_enable_moved_once1 {
        no Amend
        some disj sm1, sm2 : SecondaryMotion |
                                sm1+sm2 in PreviousQuestion  and
                                sm1.isAppliedTo = sm2.isAppliedTo and
                                sm1.takesOver = sm2.takesOver and
                                (sm1).isAppliedTo in RootMotion and
                                (sm2).isAppliedTo in RootMotion and
                                some sm1.yieldsTo and some sm2.yieldsTo and
                                sm1.yieldsTo in Recess and  sm2.yieldsTo in LayOnTheTable
}
run show_particular_enable_moved_once1  for 5 but exactly 1 RootMotion, exactly 4
SecondaryMotion
expect 1

pred show_particular_enable_moved_once2  {
        no Amend
        some sm1: SecondaryMotion |
                                sm1.isAppliedTo in RootMotion and
                                sm1 in PreviousQuestion  and
                                not lone sm1.yieldsTo and
                                some sm1.yieldsTo & Recess and some sm1.yieldsTo & LayOnTheTable
}
run show_particular_enable_moved_once2  for 5 but exactly 1 RootMotion, exactly 3
SecondaryMotion
expect 1

//TODO
pred show_enable_moved_once_with_branch_dependency {
        no Amend
        some disj sm1, sm2 : SecondaryMotion |
                                haveSameType[sm1+sm2] and
                                sm1.isAppliedTo = sm2.isAppliedTo and
                                sm1.takesOver = sm2.takesOver and
                                (sm1).isAppliedTo in RootMotion and
                                (sm2).isAppliedTo in RootMotion and
                                some sm1.yieldsTo and some sm2.yieldsTo and
                                some sm2.yieldsTo&IncidentalMotion and
                                no sm1.yieldsTo&IncidentalMotion
                                and sm1 in IncidentalMotion.isAppliedTo

}
```

```
run show_enable_moved_once_with_branch_dependency for 5 but exactly 1 RootMotion, exactly 5 SecondaryMotion
expect 1

/*********************************
 DISABLE AUXILIARY ATOMS
*********************************/

pred a_transparent_amend {
not disable_auxiliary_atoms
        some amend : Amend | some getTargets[amend]
}
run a_transparent_amend for 5

module tests_model1_precedence

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

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

/*********************************************
                PRECEDENCE STRUCTURE TESTING
*********************************************/
pred PRECEDENCE_STRUCTURE_TESTING {}


//----SHOULD SUCCEED---//
pred SHOULD_SUCCEED{}

assert there_is_always_at_most_one_immediatePendingQuestion {
        all ss : SessionState | lone ss.immediatePendingQuestion
}
```

```
assert the_precedence_structure_is_a_tree {
	all m : Motion | lone m.takesOver
}
check the_precedence_structure_is_a_tree for 10 but exactly 1 Business, exactly 1 SessionState expect
0

assert no_session_with_no_business{
	no businessless : SessionState | no businessless.orderOfBusiness
}
check no_session_with_no_business  for 5 but  exactly 1 Business
expect 0

pred a_session_with_no_motion_toComeToOrder{
	no SessionState.toComeToOrder
}
run a_session_with_no_motion_toComeToOrder for 5 but  exactly 1 Business
expect 1

pred some_more_than_binary_tree_of_motions {
	some motion : Motion | not lone (motion.yieldsTo)
}
run some_more_than_binary_tree_of_motions for 13 but  exactly 1 Business, exactly 4 Motion expect
1

assert no_pending_secondaries_takingPrecedenceOver_a_non_pending_motion {
	no s: SessionState, sm: s.pendingQuestions, m: Motion |
		m in sm.takesOver and m not in s.pendingQuestions
}
check no_pending_secondaries_takingPrecedenceOver_a_non_pending_motion for 4 but 1
SessionState expect 0

assert no_pending_rootMotion_not_dealingWith_a_Business {

	no m: RootMotion & (SessionState.pendingQuestions) | no m.dealsWith
}
check no_pending_rootMotion_not_dealingWith_a_Business for 7 but 1 SessionState, exactly 1
Business
expect 0

assert pending_root_motions_allways_dealWith_current_businesses {

	all s: SessionState, rm : RootMotion&s.pendingQuestions | rm.dealsWith = s.currentBusiness
}
check pending_root_motions_allways_dealWith_current_businesses for 7 but 1 SessionState, exactly 1
Business
expect 0

pred all_subsidiary_and_privileged_on_one_branch {
	no RaiseQuestionOfPrivilege.questionOfPrivilege.yieldsTo
```

```
        all sm : SubsidiaryMotion | some (sm.isAppliedTo & RootMotion)
        some
      pi : PostponeIndefinitely
                , a : Amend
                , c : Commit
                , p : Postpone
                //, led : LimitExtendDebate /**BINGO1?**/ impossible to have all subsidiaries stacked
unless led and pq are removed
                //, pq : PreviousQuestion
                , lat : LayOnTheTable
           , cad : CallForOrdersOfDay
                , rqp : RaiseQuestionOfPrivilege
                , r : Recess
                , ad : Adjourn
                , fta  : FixTimeAdjourn |
                        (pi + a + c + p+ lat + cad+ rqp+r+ad ) in fta.^takesOver
}
run all_subsidiary_and_privileged_on_one_branch for 13 but 1 SessionState
expect 1

pred all_privileged_on_one_branch {
   all m : Motion | lone m.yieldsTo
        no RaiseQuestionOfPrivilege.questionOfPrivilege.yieldsTo
}
run all_privileged_on_one_branch for 10 but
exactly 1 Business

,exactly 2 RootMotion
,exactly 1 CallForOrdersOfDay
,exactly 1 RaiseQuestionOfPrivilege
,exactly 1 Recess
,exactly 1 Adjourn
,exactly 1 FixTimeAdjourn

expect 1

/**
 * BINGO1?
 */
pred all_subsidiaries_on_one_branch {
                some
      pi : PostponeIndefinitely
                , a : Amend
                , c : Commit
                , p : Postpone
                //, led : LimitExtendDebate /**BINGO1?**/ impossible to have all subsidiaries stacked
unless led and pq are removed
                //, pq : PreviousQuestion
                , lat : LayOnTheTable |
```

```
                        (pi + a + c + p ) in lat.^takesOver
                all sm : SecondaryMotion | some (sm.isAppliedTo & RootMotion)
}
run all_subsidiaries_on_one_branch for 7 but 7 SessionState
expect 1

pred some_business_dealtWith_multiple_rootMotions {
        some b : Business | not lone b.isDealtWithBy
}
run some_business_dealtWith_multiple_rootMotions for 5 but 4 Int
expect 1

//-----SHOULD FAIL-----//
pred SHOULD_FAIL{}

pred some_LayOnTheTable_with_bad_precedence_rank {

        some lott : LayOnTheTable | some (lott.^yieldsTo & (SubsidiaryMotion-Amend))

}
run some_LayOnTheTable_with_bad_precedence_rank for 5 but  exactly 1 SessionState, exactly 1
Business
expect 0

//TODO : not tight enough
pred motion_potentially_movable_twice_and_not_Amend_or_a_MainMotion {
        enable_movedOnce and
        some victim : Motion+Business |
(not lone (victim.yieldsTo & PostponeIndefinitely) and (some m, n : victim.yieldsTo&
PostponeIndefinitely| m.isAppliedTo = n.isAppliedTo) ) or
(not lone (victim.yieldsTo & Commit ) and (some m, n : victim.yieldsTo & Commit| m.isAppliedTo =
n.isAppliedTo) ) or
(not lone (victim.yieldsTo & Postpone) and (some m, n : victim.yieldsTo & Postpone| m.isAppliedTo =
n.isAppliedTo) ) or
(not lone (victim.yieldsTo &  LimitExtendDebate) and (some m, n : victim.yieldsTo &
LimitExtendDebate| m.isAppliedTo = n.isAppliedTo) )or
(not lone (victim.yieldsTo & PreviousQuestion) and (some m, n : victim.yieldsTo & PreviousQuestion|
m.isAppliedTo = n.isAppliedTo) )or
(not lone (victim.yieldsTo & LayOnTheTable)  and (some m, n : victim.yieldsTo & LayOnTheTable|
m.isAppliedTo = n.isAppliedTo))or
(not lone (victim.yieldsTo & CallForOrdersOfDay) and (some m, n : victim.yieldsTo &
CallForOrdersOfDay| m.isAppliedTo = n.isAppliedTo) )or
(not lone (victim.yieldsTo & RaiseQuestionOfPrivilege) and (some m, n : victim.yieldsTo &
RaiseQuestionOfPrivilege| m.isAppliedTo = n.isAppliedTo) )or
(not lone (victim.yieldsTo & Recess ) and (some m, n : victim.yieldsTo & Recess | m.isAppliedTo =
n.isAppliedTo) )or
(not lone (victim.yieldsTo & Adjourn ) and (some m, n : victim.yieldsTo & Adjourn| m.isAppliedTo =
n.isAppliedTo) )or
(not lone (victim.yieldsTo & FixTimeAdjourn ) and (some m, n : victim.yieldsTo & FixTimeAdjourn|
```

```
m.isAppliedTo = n.isAppliedTo) )
}
run motion_potentially_movable_twice_and_not_Amend_or_a_MainMotion for 5 but  1 Business
expect 0


/**
 * BINGO1? the previous question cannot apply to the lower ranked
 * and to the root at the same time.
 */
pred the_previous_question_applied_to_the_lower_ranked{
        some pq : PreviousQuestion, pi : PostponeIndefinitely, a : Amend ,
                c : Commit/*, p : Postpone, led : LimitExtendDebate*/ , r : RootMotion |
                        (r+pi+a+c) in pq.isAppliedTo
}
run the_previous_question_applied_to_the_lower_ranked for 5 but 7 SessionState
expect 0

module tests_model1_state

open model1

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


/*****************************************************************
 *                                                STATE TESTING
                                                *
 * (MOTION STATES SETS AND SESSION STATES TESTING)  *
 *****************************************************************/
                                                pred STATES_TESTING{}


/*****************************
 *       ORDER OF BUSINESS  *
 *****************************/
pred ORDER_OF_BUSINESS{}
```

```
run ORDER_OF_BUSINESS

//_____SHOULD_SUCCEED____//
  pred SHOULD_SUCCEED {}

assert businesses_not_part_of_some_orderOfBusiness_are_auxiliary{
        all b : Business | (b not in SessionState.orderOfBusiness) <=> b in nonSessionStateElements
}
check businesses_not_part_of_some_orderOfBusiness_are_auxiliary for 2
expect 0

assert the_ipq_s_parents_are_all_pending {
        all ss : SessionState |
                ss.immediatePendingQuestion.*takesOver in ss.pendingQuestions
}
check the_ipq_s_parents_are_all_pending for 5 but 1 Business
expect 0

pred some_sessionState_with_no_currentBusiness {
        some ses : SessionState | no ses.currentBusiness
}
run some_sessionState_with_no_currentBusiness for 10 but  exactly 1 Business
expect 1

pred show_a_session_with_a_non_null_orderofBusiness {
        some ss : SessionState |
                some ss.orderOfBusiness
}
run show_a_session_with_a_non_null_orderofBusiness for 5
expect 1

//_____SHOULD_FAIL_____//
pred SHOULD_FAIL {}

/**
 *!should fail only if there exist no actions capable of
 * modifying the orderOfBuisness of a sessionState
 */
pred some_session_with_oscillating_orderOfBusiness_set {
        some ss1, ss2 : SessionState | not ss1.orderOfBusiness=ss2.orderOfBusiness
}
run some_session_with_oscillating_orderOfBusiness_set for 2 but 5 Business
expect 0

/**
 *!should fail only if there exist no actions capable of
 *modifying the orderOfBuisness of a sessionState
 */
/*pred some_session_with_oscillating_orderOfBusiness_first_or_last_item {
```

```
	some ss1, ss2 : SessionState |
		(not ss1.firstBusiness=ss2.firstBusiness) or (not ss1.lastBusiness= ss2.lastBusiness)
}
run some_session_with_oscillating_orderOfBusiness_first_or_last_item for 2 but  5 Business
expect 0
*/
pred some_non_final_sessionState_with_no_currentBusiness_when_wholeSessions {
	whole_session and
	some ses : SessionState | not finalSession[ses] and no ses.currentBusiness
}
run some_non_final_sessionState_with_no_currentBusiness_when_wholeSessions for 7
expect 0


/*****************************
 *	PENDING QUESTIONS  *
 *****************************/
pred PENDING_QUESTIONS{}


//_____SHOULD_SUCCEED____//
  pred SHOULD_SUCCEED {}


assert the_immediate_pending_question_is_unique {
	all ss : SessionState | lone ss.immediatePendingQuestion
}
check the_immediate_pending_question_is_unique for 5
expect 0

pred some_SessionState_with_no_pending_motion {
	some ses : SessionState| no ses.pendingQuestions
}
run some_SessionState_with_no_pending_motion for 5 but exactly 1 Business expect 1

pred some_main_never_pending {
	some m :MainMotion | m not in SessionState.pendingQuestions
}
run some_main_never_pending for 5 but exactly 1 Business expect 1

assert no_pending_secondaries_with_no_main {
	no s: SessionState| some(s.pendingQuestions & SecondaryMotion) and no (s.pendingQuestions
& RootMotion)
}
check no_pending_secondaries_with_no_main for 5 but exactly 1 Business expect 0

//_____SHOULD_FAIL_____//
pred SHOULD_FAIL {}
```

```
/***************
 *        ADOPTED  *
 ***************/
pred ADOPTED{}
run ADOPTED

//_____SHOULD_SUCCEED____//
  pred SHOULD_SUCCEED {}

pred some__motions_can_be_adopted_and_some_not {
        some ses : SessionState | some (Motion & ses.adopted) and not (Motion in ses.adopted)
}
run some__motions_can_be_adopted_and_some_not  for 10 but exactly 1 Business expect 1

//_____SHOULD_FAIL_____//
pred SHOULD_FAIL {}


/********************
 *        UNDER ORDER *
 ********************/
pred UNDER_ORDER{}
run UNDER_ORDER

//_____SHOULD_SUCCEED____//
  pred SHOULD_SUCCEED {}

//_____SHOULD_FAIL_____//
pred SHOULD_FAIL {}


pred some_initial_sessionState_with_motions_under_some_order {
        some ss : SessionState | ss = firstSessionState and some
(ss.underCommitOrPostponeIndefinitelyOrder + ss.underPreviousQuestionOrder)
}
run some_initial_sessionState_with_motions_under_some_order  for 5
expect 0


/************************
 *        MISCELLANEOUS  *
 ************************/
pred MISCELLANEOUS{}
run MISCELLANEOUS

//_____SHOULD_SUCCEED____//
  pred SHOULD_SUCCEED {}
```

```
assert no_RootMotion_not_pending_nor_toComeToOrder_nor_to_come_to_existance {
	no rm : RootMotion |
				rm not in (SessionState.(pendingQuestions+toComeToOrder)
							+ DivisionOfQuestion.separatedQuestions
							+
RaiseQuestionOfPrivilege.questionOfPrivilege )
}
check no_RootMotion_not_pending_nor_toComeToOrder_nor_to_come_to_existance for 5 but exactly
1 Business
expect 0

pred some_motion_unreached {
	some ses : SessionState | finalSession[ses] and one ses.toComeToOrder
}
run some_motion_unreached for 10 but exactly 1 Business
expect 1

pred some_motion_unreached_because_others_are_under_some_order {
	//there is one motion that does not put others under some order
	//there is one motion to Postpone
	//the rootMotion yields to two siblings
	//one motion yields precedence to something that will never be pending
	all orderer : SecondaryMotion | orderer in (LayOnTheTable + Commit + Postpone +
PostponeIndefinitely)
	some ses : SessionState | finalSession[ses] and some ses.adopted and one ses.toComeToOrder
}
run some_motion_unreached_because_others_are_under_some_order for 10 but exactly 1 Business
expect 1

//_____SHOULD_FAIL_____//
pred SHOULD_FAIL {}

			/*****************************
			 * STATE OSCILLATION   *
			 *****************************/

pred STATE_OSCILLATION{}
run STATE_OSCILLATION

//-----auxiliary-predicates-and-functions-----//
pred entered[before, after : SessionState, elem : Motion+Business, aSet : (SessionState -> Motion) +
(SessionState -> Business)]{
	after = before.next and
		elem not in before.aSet and
		elem in after.aSet
}

pred exited[before, after : SessionState, elem : Motion+Business, aSet : (SessionState -> Motion)+
```

```
(SessionState -> Business)]{
	after = before.next and
		elem in before.aSet and
		elem not in after.aSet
}

//-----Should-fail-----//
pred SHOULD_FAIL{}

pred a_motion_illegally_inside_the_toComeToOrder{
	some motion : Motion, before, after : SessionState |
		entered[before, after, motion, toComeToOrder] and
		motion not in firstSessionState.toComeToOrder and
		(no dq : DivisionOfQuestion| adopt[before, after, dq]
			and motion in dq.separatedQuestions.*yieldsTo) and
		(no rqp : RaiseQuestionOfPrivilege | adopt[before, after, rqp]
			and motion in rqp.questionOfPrivilege .^yieldsTo)
}
run a_motion_illegally_inside_the_toComeToOrder for 7
expect 0

pred a_motion_illegally_outside_the_toComeToOrder{

	some motion : Motion, before, after : SessionState |
		after = before.next and
		motion not in after.toComeToOrder and
		motion in before.toComeToOrder and
		not move[before, after, motion]

}
run a_motion_illegally_outside_the_toComeToOrder for 7
expect 0

pred a_motion_illegally_inside_the_pendingQuestions{
	some motion : Motion, before, after : SessionState |
		entered[before, after, motion , pendingQuestions] and
		not move[before, after, motion] and
		no rqp : RaiseQuestionOfPrivilege |
			motion = rqp.questionOfPrivilege and
			adopt[before, after, rqp]
}
run a_motion_illegally_inside_the_pendingQuestions for 7
expect 0

pred a_motion_illegally_outside_the_pendingQuestions {

	some motion : Motion, before, after : SessionState |
		exited[before, after, motion, pendingQuestions] and
		not move[before, after, motion] and
```

```
		not adopt[before, after, motion]  and
		not disposeOf[before, after, motion] and
		(no pi : PostponeIndefinitely | adopt[before, after, pi ] ) and
		(no c : Commit | adopt[before, after, c]) and
		(no lot : LayOnTheTable | adopt[before, after, lot]) and
		(no dq : DivisionOfQuestion | dq in before.adopted
						and disposeOf[before, after, dq] and some (motion &
dq.^takesOver & dq.isAppliedTo.^yieldsTo)) and
		not disposeOf[before, after, motion.questionOfPrivilege]
}
run a_motion_illegally_outside_the_pendingQuestions for 7
expect 0

pred a_motion_illegally_inside_the_ipq{
	some motion : Motion, before, after : SessionState |
		entered[before, after, motion, immediatePendingQuestion ]and
		not move[before, after, motion] and
		(no child : motion.yieldsTo| disposeOf[before, after, child]) and
		(no rqp : RaiseQuestionOfPrivilege |
			motion = rqp.questionOfPrivilege and
			adopt[before, after, rqp]) and
		(no rqp2 : RaiseQuestionOfPrivilege |
			motion = rqp2.takesOver and
			disposeOf[before, after, rqp2.questionOfPrivilege]) and
		(no p : Postpone | adopt[before, after, p] and p in motion.yieldsTo) and
		(no cod : CallForOrdersOfDay | adopt[before, after, cod]and cod in motion.yieldsTo)
}
run a_motion_illegally_inside_the_ipq for 5
expect 0

pred a_motion_illegally_outside_the_ipq{
	some motion : Motion, before, after : SessionState |
		exited[before, after, motion, immediatePendingQuestion ]and
		not disposeOf[before, after, motion] and
		not (motion in PostponeIndefinitely and adopt[before, after, motion]) and
		not (motion in Commit and adopt[before, after, motion]) and
		not (motion in RaiseQuestionOfPrivilege and adopt[before, after, motion]) and
		not (motion in LayOnTheTable and adopt[before, after, motion]) and
		not (motion in Postpone and adopt[before, after, motion]) and
		not (motion in CallForOrdersOfDay and adopt[before, after, motion]) and
		(no child : motion.yieldsTo | move[before, after, child ])

}
run a_motion_illegally_outside_the_ipq for 5
expect 0

pred a_business_illegally_inside_the_currentBusiness{
	some business : Business, before, after : SessionState |
		entered[before, after, business, currentBusiness]
```

```
		and (no r : RootMotion| no(r.dealsWith&after.currentBusiness) and disposeOf[before,
after, r])
		and (no motion :
(Commit+PostponeIndefinitely+Postpone+LayOnTheTable+CallForOrdersOfDay)| adopt[before, after,
motion])
}
run a_business_illegally_inside_the_currentBusiness for 5
expect 0

pred a_motion_illegally_outside_the_currentBusiness{
	some business : Business, before, after : SessionState |
		exited[before, after, business, currentBusiness]
		and (no r : business.isDealtWithBy | disposeOf[before, after, r])
		and (no motion :
(Commit+PostponeIndefinitely+Postpone+LayOnTheTable+CallForOrdersOfDay)|
				some(before.currentBusiness & motion.getBusiness) and  adopt[before,
after, motion])
}
run a_motion_illegally_outside_the_currentBusiness for 5
expect 0

//--//

/*pred a_relation_illegally_inside_the_specialOrder{//TODO
	some rel : SessionState.specialOrder, before, after : SessionState |
		after = before.next and
		rel not in before.specialOrder and
		rel in after.specialOrder
		and (no cod : CallForOrdersOfDay | adopt[before, after, cod])
		and (after.currentBusiness = before.currentBusiness.(before.nextBusiness) and no
after.pendingQuestions )
}
run a_relation_illegally_inside_the_specialOrder for 5 but 7 SessionState
expect 0

pred a_motion_illegally_outside_the_specialOrder{//TODO
	some rel : SessionState.specialOrder, before, after : SessionState |
		after = before.next and
		rel in before.specialOrder and
		rel not in after.specialOrder
		and (no r : before.currentBusiness.isDealtWithBy | disposeOf[before, after, r])

}
run a_motion_illegally_outside_the_specialOrder for 5 but 6 SessionState//NOTE :+1 sessionstate
explodes solving time
expect 0
*/

//--//
```

```
pred a_motion_illegally_inside_the_adopted{
	some motion : Motion, before, after : SessionState |
		entered[before, after, motion, adopted]
		and not adopt[before, after, motion]
}
run a_motion_illegally_inside_the_adopted for 5
expect 0

pred a_motion_illegally_outside_the_adopted{
	some motion : Motion, before, after : SessionState |
		exited[before, after, motion, adopted]
}
run a_motion_illegally_outside_the_adopted for 5
expect 0


//--//

pred a_motion_illegally_inside_the_laidOnTheTable{
	some motion : Motion, before, after : SessionState |
		entered[before, after, motion, laidOnTheTable]
		and (no lot : LayOnTheTable | motion in
((lot.isAppliedTo.*yieldsTo&before.pendingQuestions)-lot) and adopt[before, after, lot])
}
run a_motion_illegally_inside_the_laidOnTheTable for 5 but 7 SessionState
expect 0

pred a_motion_illegally_outside_the_laidOnTheTable{//TODO
		some motion : Motion, before, after : SessionState |
		exited[before, after, motion, laidOnTheTable]
}
run a_motion_illegally_outside_the_laidOnTheTable for 5 but 7 SessionState
expect 0

//--//


pred a_motion_illegally_inside_the_underPreviousQuestionOrder{
	some motion : Motion, before, after : SessionState |
		entered[before, after, motion, underPreviousQuestionOrder]
		and (no pq : PreviousQuestion | motion in pq.isAppliedTo and adopt[before, after, pq])

}
run a_motion_illegally_inside_the_underPreviousQuestionOrder for 5 but 6 SessionState
expect 0

pred a_motion_illegally_outside_the_underPreviousQuestionOrder{
	some motion : Motion, before, after : SessionState |
```

```
                exited[before, after, motion, underPreviousQuestionOrder]
                and not disposeOf[before, after, motion]
        and (no pi : PostponeIndefinitely |
                                                (motion in (getAppliedTo[pi]) and
adopt[before, after, pi])   )
        and (no c : Commit |
                                                (motion in (getAppliedTo[c]) and
adopt[before, after, c])  )

}
run a_motion_illegally_outside_the_underPreviousQuestionOrder for 5 but 6 SessionState
expect 0

//--//

pred a_motion_illegally_inside_the_underCommitOrPostponeIndefinitelyOrder{//TODO
        some motion : Motion, before, after : SessionState |
                entered[before, after, motion, underCommitOrPostponeIndefinitelyOrder]
                and (no c : Commit |
                                                (motion in (getAppliedTo[c]) and
adopt[before, after, c])  )
                and (no pi : PostponeIndefinitely |
                                                (motion in (getAppliedTo[pi]) and
adopt[before, after, pi])   )
}
run a_motion_illegally_inside_the_underCommitOrPostponeIndefinitelyOrder for 5
expect 0

pred a_motion_illegally_outside_the_underCommitOrPostponeIndefinitelyOrder {//TODO
        some motion : Motion, before, after : SessionState |
                exited[before, after, motion, underCommitOrPostponeIndefinitelyOrder]
}
run a_motion_illegally_outside_the_underCommitOrPostponeIndefinitelyOrder for 5
expect 0
```