```
module tests_model1_actions

open model0

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

                                    /*************************
                                *  ACTIONS TESTING  *
                                     *************************/
                                    pred ACTIONS_TESTING{}



                                    /************
                                *  MOVE *
                                     ************/
                                    pred MOVE{}
                                     run MOVE

//----SHOULD-SUCCEED-----//

run move for 3 but
exactly 1 Business
expect 1


pred a_moved_RootMotion {
        some before, after : SessionState, root : RootMotion | move[before, after, root]
}
run a_moved_RootMotion for 3 but
         exactly 1 Business
expect 1

pred a_moved_SecondaryMotion {
        some before, after : SessionState, sec : SecondaryMotion|  move[before, after, sec]
}
run a_moved_SecondaryMotion for 10 but
         exactly 1 Business
```

```
expect 1

pred a_motion_moved_on_top_of_a_motion_under_pq_order {
	some disj before, after : SessionState, motion : Motion |
		move[before, after, motion] and
		some (motion.takesOver) & (before.(underPreviousQuestionOrder))
}
run a_motion_moved_on_top_of_a_motion_under_pq_order for 5 but 6 SessionState
expect 1

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

pred a_motion_moved_on_top_of_an_adpted_motion {
	some disj before, after : SessionState, motion : Motion |
		move[before, after, motion] and some (motion.takesOver) & (before.adopted)
}
run a_motion_moved_on_top_of_an_adpted_motion for 5 but 8 SessionState
expect 0

pred a_motion_moved_on_top_of_a_motion_under_lot_c_or_p_order {
	some disj before, after : SessionState, motion : Motion |
		move[before, after, motion] and some  (motion.takesOver) & (before.
(laidOnTheTable+underCommitOrPostponeIndefinitelyOrder))
}
run a_motion_moved_on_top_of_a_motion_under_lot_c_or_p_order for 5 but 8 SessionState
expect 0

		/*************
	*  ADOPT *
		 *************/
		pred ADOPT{}
		 run ADOPT

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

run adopt for 4 but
	 exactly 1 Business
expect 1


pred an_Adopted_RootMotion {
	some before, after : SessionState, root : RootMotion|  adopt[before, after, root]
}
run an_Adopted_RootMotion for 4 but
	 exactly 1 Business
expect 1
```

```
pred an_Adopted_SecondaryMotion {
        some before, after : SessionState, sec : SecondaryMotion|  adopt[before, after, sec]
}
run an_Adopted_SecondaryMotion for 10 but
         exactly 1 Business
expect 1

pred show_PostponeIndefinitely_adopted {
        some before, after : SessionState, pi : PostponeIndefinitely | adopt[before, after, pi]
}
run show_PostponeIndefinitely_adopted for 5 but 6 SessionState
expect 1

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

                        /*******************
                *  DISPOSE OF *
                         ******************/
                        pred DISPOSE_OF{}
                         run DISPOSE_OF

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

run disposeOf for 3 but
         exactly 1 Business
expect 1


pred a_disposedOf_RootMotion {
        some before, after : SessionState, root : RootMotion |  disposeOf[before, after, root]
}
run a_disposedOf_RootMotion for 5 but
         exactly 1 Business
expect 1

pred a_disposedOf_SecondaryMotion {
        some before, after : SessionState, sec : SecondaryMotion |  disposeOf[before, after, sec]
}
run a_disposedOf_SecondaryMotion for 5 but
         exactly 1 Business
expect 1

pred show_disposingOf_rootMotion_and_staying_on_same_business_then_moving {
        some disj before, current, after : SessionState, r1, r2 : RootMotion | disposeOf[before, current,
r1] and
                        before.currentBusiness = current.currentBusiness and
                        move[current, after, r2]
```

```
}
run show_disposingOf_rootMotion_and_staying_on_same_business_then_moving for 5 but 7 Int
expect 1

pred
show_disposingOf_rootMotion_and_staying_on_same_business_then_moving_and_reach_finalState {
	some disj before, current, after : SessionState, r1, r2 : RootMotion | disposeOf[before, current,
r1] and
			before.currentBusiness = current.currentBusiness and
			move[current, after, r2]
			whole_session
}
run
show_disposingOf_rootMotion_and_staying_on_same_business_then_moving_and_reach_finalState
for 5 but 7 Int
expect 1

pred show_swithch_to_other_main_motion_dealing_with_same_business {
	some disj before, after : SessionState, r : RootMotion | disposeOf[before, after, r]
	one Business
	not lone RootMotion
	whole_session
	no RootMotion&(lastSessionState.toComeToOrder)
}
run show_swithch_to_other_main_motion_dealing_with_same_business for 5 but 7 Int
expect 1

pred show_disposingOf_rootMotion_and_more_than_one_business {
	some disj before, after : SessionState, r : RootMotion | disposeOf[before, after, r] and
	not lone Business
}
run show_disposingOf_rootMotion_and_more_than_one_business for 5 but 5 SessionState
expect 1

pred show_disposingOf_rootMotion_and_changing_of_current_business {//TODO

	some disj before, after : SessionState, r : RootMotion | disposeOf[before, after, r]
			and after.currentBusiness = before.currentBusiness.(before.nextBusiness)
			and some after.currentBusiness
}
run show_disposingOf_rootMotion_and_changing_of_current_business for 5 but 5 SessionState
expect 1

pred show_disposingOf_rootMotion_and_changing_of_current_business_and_reach_finalState
{//TODO
	some disj before, after : SessionState, r : RootMotion | disposeOf[before, after, r] and
			not before.currentBusiness = after.currentBusiness
			and whole_session
}
```

```
run show_disposingOf_rootMotion_and_changing_of_current_business_and_reach_finalState for 5 but
7 Int
expect 1

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

                        /*********************
                *  LOSE (implicit) *
                        *********************/
                                pred LOSE{}
                                        run LOSE

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

pred a_lost_RootMotion {
        some disj before, after : SessionState, root : RootMotion|
        move[before.prev, before, root] and
        not adopt[before, after, root] and
        disposeOf[before, after, root]
        //and not doNothing[before, after]

}
run a_lost_RootMotion for 5 but
         exactly 1 Business
expect 1

pred a_lost_SecondaryMotion {
        some before, after : SessionState, root : SecondaryMotion|
        move[before.prev, before, root] and
        not adopt[before, after, root] and
        disposeOf[before, after, root]
        //and not doNothing[before, after]
}
run a_lost_SecondaryMotion for 10 but
          5 Int
expect 1

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

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

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

assert the_specialOrder_should_be_followed_whenever_the_currentBusiness_is_changed {//TODO
	all disj before, after : SessionState|
		(after = before.next and
		 some before.specialOrder and
		 not before.currentBusiness = after.currentBusiness
		 )
			=> after.currentBusiness = (before.currentBusiness).(before.specialOrder)
}
check the_specialOrder_should_be_followed_whenever_the_currentBusiness_is_changed for 5 but
6 SessionState
expect 0

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

				/**********************
			 *  DONOTHING     *
				**********************/
//					pred DONOTHING{}
//					run DONOTHING


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


//pred leaving_the_start_state_by_doing_nothing{
//	some disj before, after : SessionState | initialSession[before] and doNothing[before, after]
//}
//run leaving_the_start_state_by_doing_nothing for 10 but  exactly 1 Business,  5 Int

//pred reaching_the_end_satate_by_doing_nothing {
//	some disj before, after :  SessionState | finalSession[before] and doNothing[before, after]
//}
//run reaching_the_end_satate_by_doing_nothing for 10 but  exactly 1 Business,  5 Int

//pred an_instance_where_doing_nothing_is_allowed_only_at_the_end_of_a_session_trace {
//		all disj before : SessionState , after : before.next |
//		(doNothing[before, after]) =>
//			finalSession[after]
//
//		no disj before, after : SessionState |
//		after = before.next and
//		doNothing[before, after] and
//		some after.next and
//		not doNothing[before.prev, before] and
```

```
//                  not doNothing[after, after.next]
//}
//run an_instance_where_doing_nothing_is_allowed_only_at_the_end_of_a_session_trace for 10 but exactly 1 Business,  5 Int
//expect 1


//-----SHOULD FAIL-----//

//pred doing_noting_at_least_once_between_two_other_actions {
//
//        some disj before, after : SessionState |
//                  doNothing[before, after] and
//                  some after.next and some before.prev and
//                  not doNothing[before.prev, before] and
//                  not doNothing[after, after.next] //manque la conditions que before.next = after !!
//
//}
//run doing_noting_at_least_once_between_two_other_actions for 10 but  exactly 1 Business,  5 Int
//expect 0

//pred some_hole_trace_where_one_only_does_nothing {
//        no disj before, after : SessionState |  not doNothing[before, after]
//        some final : SessionState | no final.next and finalSession[final]
//}
//run some_hole_trace_where_one_only_does_nothing for 5 but  exactly 1 Business,  5 Int expect 0

module tests_model0_domain_elements

open model0

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


/************************************
 *       DOMAIN ELEMENTS TESTING  *
 ************************************/
```

```
/*********************************************
 ***** MOTIONS AND ACTIONS TESTS *****
 *********************************************/
pred MOTIONS_AND_ACTIONS_TESTS{}
run MOTIONS_AND_ACTIONS_TESTS{}

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

 /*************************
        SUBSIDIARY_MOTION
  *************************/
 pred SUBSIDIARY_MOTIONS {}

pred move_PostponeIndefinitely {
        some before, after : SessionState, motion : PostponeIndefinitely| move[before, after, motion]
}run move_PostponeIndefinitely for 5 expect 1

pred move_AmendDebatable {
        some before, after : SessionState, motion : AmendDebatable | move[before, after, motion]
}run move_AmendDebatable for 5 expect 1

pred move_AmendUndebatable{
        some before, after : SessionState, motion : AmendUndebatable | move[before, after, motion]
}run move_AmendUndebatable for 8 expect 1

pred move_Commit{
        some before, after : SessionState, motion : Commit| move[before, after, motion]
}run move_Commit for 5 expect 1


pred move_Postpone {
        some before, after : SessionState, motion : Postpone| move[before, after, motion]
}run move_Postpone for 5 expect 1


pred move_LimitExtendDebate{
        some before, after : SessionState, motion : LimitExtendDebate| move[before, after, motion]
}run move_LimitExtendDebate for 5 expect 1


pred move_PreviousQuestion {
        some before, after : SessionState, motion : PreviousQuestion | move[before, after, motion]
}run move_PreviousQuestion for 5 expect 1


pred move_LayOnTheTable{
```

```
	some before, after : SessionState, motion : LayOnTheTable| move[before, after, motion]
}run move_LayOnTheTable for 5 expect 1

pred move_LayOnTheTable_in_whole_session{
	some before, after : SessionState, motion : LayOnTheTable| move[before, after, motion]
	and whole_session
}run move_LayOnTheTable_in_whole_session for 5 expect 1

//***//

pred adopt_PostponeIndefinitely {
	some before, after : SessionState, motion : PostponeIndefinitely | adopt[before, after, motion]
}run adopt_PostponeIndefinitely for 7 expect 1

pred adopt_PostponeIndefinitely_and_go_to_next_business {
	some before, after : SessionState, motion : PostponeIndefinitely | adopt[before, after, motion]
	and after.currentBusiness = before.currentBusiness.(before.nextBusiness)
}run adopt_PostponeIndefinitely_and_go_to_next_business for 7 expect 1


pred adopt_AmendDebatable{
	some before, after : SessionState, motion : AmendDebatable| adopt[before, after, motion]
}run adopt_AmendDebatable for 7 expect 1


pred adopt_AmendUndebatable{
	some before, after : SessionState, motion : AmendUndebatable| adopt[before, after, motion]
}run adopt_AmendUndebatable for 8 expect 1


pred adopt_Commit{
	some before, after : SessionState, motion : Commit| adopt[before, after, motion]
}run adopt_Commit for 7 expect 1

pred adopt_Commit_and_go_to_next_business {
	some before, after : SessionState, motion : Commit | adopt[before, after, motion]
	and after.currentBusiness = before.currentBusiness.(before.nextBusiness)
}run adopt_Commit_and_go_to_next_business for 7 expect 1


pred adopt_Postpone {
	some before, after : SessionState, motion : Postpone| adopt[before, after, motion]
}run adopt_Postpone for 5 expect 1


pred adopt_LimitExtendDebate{
	some before, after : SessionState, motion : LimitExtendDebate | adopt[before, after, motion]
}run adopt_LimitExtendDebate for 7 expect 1
```

```
pred adopt_PreviousQuestion {
	some before, after : SessionState, motion : PreviousQuestion | adopt[before, after, motion]
}run adopt_PreviousQuestion for 7 expect 1


pred adopt_LayOnTheTable{
	some before, after : SessionState, motion : LayOnTheTable | adopt[before, after, motion]
}run adopt_LayOnTheTable for 6 but 7 SessionState, 2 Business expect 1

pred adopt_LayOnTheTable_in_whole_session{
	some before, after : SessionState, motion : LayOnTheTable | adopt[before, after, motion]
	and whole_session
}run adopt_LayOnTheTable_in_whole_session for 7 but exactly 2 Business expect 1
//NOTE : for 7 => big solving time when no instance found

//***//

pred disposeOf_PostponeIndefinitely {
	some before, after : SessionState, motion : PostponeIndefinitely | disposeOf[before, after,
motion]
}run disposeOf_PostponeIndefinitely for 5 expect 1


pred disposeOf_AmendDebatable {
	some before, after : SessionState, motion : AmendDebatable | disposeOf[before, after, motion]
}run disposeOf_AmendDebatable for 5 expect 1


pred disposeOf_AmendUndebatable{
	some before, after : SessionState, motion : AmendUndebatable | disposeOf[before, after,
motion]
}run disposeOf_AmendUndebatable for 8 expect 1


pred disposeOf_Commit{
	some before, after : SessionState, motion : Commit | disposeOf[before, after, motion]
}run disposeOf_Commit for 5 expect 1


pred disposeOf_Postpone {
	some before, after : SessionState, motion : Postpone| disposeOf[before, after, motion]
}run disposeOf_Postpone for 5 expect 1


pred disposeOf_LimitExtendDebate{
	some before, after : SessionState, motion : LimitExtendDebate | disposeOf[before, after,
motion]
}run disposeOf_LimitExtendDebate for 5 expect 1
```

```
pred disposeOf_PreviousQuestion {
	some before, after : SessionState, motion : PreviousQuestion | disposeOf[before, after, motion]
}run disposeOf_PreviousQuestion for 5 expect 1


pred disposeOf_LayOnTheTable{
	some before, after : SessionState, motion : LayOnTheTable | disposeOf[before, after, motion]
}run disposeOf_LayOnTheTable for 5 expect 1

//pred disposeOf_LaidOnTheTable_secondaryMotion{//TODO
//	some before, after : SessionState, motion : SecondaryMotion |
//		disposeOf[before, after, motion] and motion in before.laidOnTheTable
//}run disposeOf_LaidOnTheTable_secondaryMotion for 7 expect 1

//pred disposeOf_LaidOnTheTable_rootMotion{//TODO
//	some before, after : SessionState, motion : RootMotion |
//		disposeOf[before, after, motion] and motion in before.laidOnTheTable
//}run disposeOf_LaidOnTheTable_rootMotion for 6 expect 1


//***//

pred lose_PostponeIndefinitely {
	some before, after : SessionState, motion : PostponeIndefinitely | not adopt[before, after,
motion] and disposeOf[before, after, motion]
}run lose_PostponeIndefinitely for 5 expect 1


pred lose_AmendDebatable {
	some before, after : SessionState, motion : AmendDebatable | not adopt[before, after, motion]
and disposeOf[before, after, motion]
}run lose_AmendDebatable for 5 expect 1


pred lose_AmendUndebatable{
	some before, after : SessionState, motion : AmendUndebatable | not adopt[before, after, motion]
and disposeOf[before, after, motion]
}run lose_AmendUndebatable for 8 expect 1


pred lose_Commit{
	some before, after : SessionState, motion : Commit | not adopt[before, after, motion] and
disposeOf[before, after, motion]
}run lose_Commit for 5 expect 1


pred lose_Postpone {
```

```
	some before, after : SessionState, motion : Postpone | not adopt[before, after, motion] and disposeOf[before, after, motion]
}run lose_Postpone for 5 expect 1


pred lose_LimitExtendDebate{
	some before, after : SessionState, motion : LimitExtendDebate | not adopt[before, after, motion] and disposeOf[before, after, motion]
}run lose_LimitExtendDebate for 5 expect 1


pred lose_PreviousQuestion {
	some before, after : SessionState, motion : PreviousQuestion | not adopt[before, after, motion] and disposeOf[before, after, motion]
}run lose_PreviousQuestion for 5 expect 1


pred lose_LayOnTheTable{
	some before, after : SessionState, motion : LayOnTheTable | not adopt[before, after, motion] and disposeOf[before, after, motion]
}run lose_LayOnTheTable for 5 expect 1

 /*************************
	PRIVILEGED-MOTIONS
  *************************/
pred PRIVILEGED_MOTIONS {}

pred move_CallForOrdersOfDay {
	some before, after : SessionState, motion : CallForOrdersOfDay | move[before, after, motion]
}run move_CallForOrdersOfDay for 5 expect 1

pred move_RaiseQuestionOfPrivilege{
	some before, after : SessionState, motion : RaiseQuestionOfPrivilege | move[before, after, motion]
}run move_RaiseQuestionOfPrivilege for 5 expect 1


pred move_Recess{
	some before, after : SessionState, motion : Recess | move[before, after, motion]
}run move_Recess for 5 expect 1


pred move_Adjourn {
	some before, after : SessionState, motion : Adjourn | move[before, after, motion]
}run move_Adjourn for 5 expect 1


pred move_FixTimeAdjourn{
	some before, after : SessionState, motion : FixTimeAdjourn| move[before, after, motion]
```

```
}run move_FixTimeAdjourn for 5 expect 1

//***//


pred adopt_CallForOrdersOfDay {
	some before, after : SessionState, motion : CallForOrdersOfDay| adopt[before, after, motion]
}run adopt_CallForOrdersOfDay for 3 but /*exactly 2 RootMotion,*/ 5 SessionState expect 1

pred adopt_CallForOrdersOfDay_and_move_upon_the_new_buisness {
	some before, current, after : SessionState, cod : CallForOrdersOfDay, motion : Motion|
		adopt[before, current, cod] and move[current, after, motion]
}run adopt_CallForOrdersOfDay_and_move_upon_the_new_buisness for 3 but exactly 2 RootMotion,
5 SessionState expect 1

pred adopt_CallForOrdersOfDay_move_and_come_back {
	some before, cod_adoted, root_moved, back : SessionState, cod : CallForOrdersOfDay, motion :
Motion|
		adopt[before, cod_adoted, cod] and
		move[cod_adoted, root_moved, motion] and
		disposeOf[root_moved, back, motion]
}run adopt_CallForOrdersOfDay_move_and_come_back for 3 but exactly 2 RootMotion, 7
SessionState expect 1

pred adopt_RaiseQuestionOfPrivilege{
	some before, after : SessionState, motion : RaiseQuestionOfPrivilege | adopt[before, after,
motion]
}run adopt_RaiseQuestionOfPrivilege for 7 expect 1

pred adopt_QuestionOfPrivilege{
	some before, after : SessionState, motion : RaiseQuestionOfPrivilege.questionOfPrivilege |
		adopt[before, after, motion]
}run adopt_QuestionOfPrivilege for 7 expect 1

pred adopt_Recess{
	some before, after : SessionState, motion : Recess | adopt[before, after, motion]
}run adopt_Recess for 7 expect 1


pred adopt_Adjourn {
	some before, after : SessionState, motion : Adjourn| adopt[before, after, motion]
}run adopt_Adjourn for 7 expect 1


pred adopt_FixTimeAdjourn{
	some before, after : SessionState, motion : FixTimeAdjourn | adopt[before, after, motion]
}run adopt_FixTimeAdjourn for 7 expect 1

//***//
```

```
pred disposeOf_CallForOrdersOfDay {
	some before, after : SessionState, motion : CallForOrdersOfDay | disposeOf[before, after, motion]
}run disposeOf_CallForOrdersOfDay for 5 expect 1


pred disposeOf_RaiseQuestionOfPrivilege{
	some before, after : SessionState, motion : RaiseQuestionOfPrivilege | disposeOf[before, after, motion]
}run disposeOf_RaiseQuestionOfPrivilege for 5 expect 1

pred disposeOf_QuestionOfPrivilege{
	some before, after : SessionState, motion : RaiseQuestionOfPrivilege.questionOfPrivilege | disposeOf[before, after, motion]
}run disposeOf_QuestionOfPrivilege for 5 expect 1


pred disposeOf_Recess{
	some before, after : SessionState, motion : Recess | disposeOf[before, after, motion]
}run disposeOf_Recess for 5 expect 1


pred disposeOf_Adjourn {
	some before, after : SessionState, motion : Adjourn | disposeOf[before, after, motion]
}run disposeOf_Adjourn for 5 expect 1


pred disposeOf_FixTimeAdjourn {
	some before, after : SessionState, motion : FixTimeAdjourn | disposeOf[before, after, motion]
}run disposeOf_FixTimeAdjourn for 5 expect 1

//***//

pred lose_CallForOrdersOfDay {
	some before, after : SessionState, motion : CallForOrdersOfDay | not adopt[before, after, motion] and disposeOf[before, after, motion]
}run lose_CallForOrdersOfDay for 5 expect 1


pred lose_RaiseQuestionOfPrivilege {
	some before, after : SessionState, motion : RaiseQuestionOfPrivilege | not adopt[before, after, motion] and disposeOf[before, after, motion]
}run lose_RaiseQuestionOfPrivilege for 5 expect 1


pred lose_Recess{
	some before, after : SessionState, motion : Recess| not adopt[before, after, motion] and disposeOf[before, after, motion]
```

```
}run lose_Recess for 5 expect 1


pred lose_Adjourn {
	some before, after : SessionState, motion : Adjourn| not adopt[before, after, motion] and
disposeOf[before, after, motion]
}run lose_Adjourn for 5 expect 1


pred lose_FixTimeAdjourn{
	some before, after : SessionState, motion : FixTimeAdjourn| not adopt[before, after, motion]
and disposeOf[before, after, motion]
}run lose_FixTimeAdjourn for 5 expect 1


pred move_Incidental {
	some before, after : SessionState, motion : IncidentalMotion | move[before, after, motion]
}run move_Incidental for 5 expect 1


 /**************************
	INCIDENTAL-MOTIONS
  **************************/
pred INCIDENTAL_MOTIONS{}

pred adopt_Incidental {
	some before, after : SessionState, motion : IncidentalMotion | adopt[before, after, motion]
}run adopt_Incidental for 8 expect 1


pred disposeOf_Incidental {
	some before, after : SessionState, motion : IncidentalMotion | disposeOf[before, after, motion]
}run disposeOf_Incidental for 5 expect 1


pred lose_Incidental {
	some before, after : SessionState, motion : IncidentalMotion | not adopt[before, after, motion]
and disposeOf[before, after, motion]
}run lose_Incidental for 5 expect 1

//***//
                                                                    /*****************
                                                                   SHOULD_FAIL

*****************/
                                                                          pred SHOULD_SUCCEED
{}
```

```
 /*************************
        PRIVILEGED-MOTIONS
  *************************/
pred PRIVILEGED_MOTIONS {}

pred move_CallForOrdersOfDay_when_out_of_order{
        some before, after : SessionState, motion : CallForOrdersOfDay | move[before, after, motion]
                                        and no (((Business-before.currentBusiness).isDealtWithBy &
before.pendingQuestions)+
                                                                                ((Business-
before.currentBusiness).isDealtWithBy & before.toComeToOrder) )
}run move_CallForOrdersOfDay_when_out_of_order for 5
expect 0

/************************************************
 ***** MISCELLANEOUS MOTION TESTS *****
************************************************/
pred MISCELLANEOUS_MOTION_TESTS{}
run MISCELLANEOUS_MOTION_TESTS{}


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

        SHOULD_SUCCEED

**********************/
                                                                        pred
SHOULD_SUCCEED{}

 /*************************
        SUBSIDIARY_MOTIONS
  *************************/
 pred SUBSIDIARY_MOTIONS {}

pred whole_session_with_non_empty_Table {
        whole_session// and
        some adopted.LayOnTheTable
}
run whole_session_with_non_empty_Table for 5
expect 1

pred show_the_PreviousQuestion {
        one pq : Motion| pq in PreviousQuestion
}
run show_the_PreviousQuestion for 5
expect 1

pred some_Amend_yielding_to_another_subsidiary_motion {
        some a : Amend | some (a.yieldsTo & SubsidiaryMotion)
```

```
}
run some_Amend_yielding_to_another_subsidiary_motion for 5 but exactly 1 Business
expect 1

pred moving_layOnTheTable_on_top_of_the_previousQuestion_and_at_least_2_secondaries {
	some lat : LayOnTheTable, pq : PreviousQuestion |
		pq in lat.takesOver and not lone (pq.isAppliedTo&SecondaryMotion)
}
run moving_layOnTheTable_on_top_of_the_previousQuestion_and_at_least_2_secondaries for 5 but 7
SessionState
expect 1

 /*************************
	PRIVILEGED-MOTIONS
  *************************/
pred PRIVILEGED_MOTIONS {}

pred show_CallForOrdersOfDay {
	some CallForOrdersOfDay
}
run  show_CallForOrdersOfDay for 5
expect 1

pred call_for_a_postponed_order_of_the_day {
	some s1, s2, s3, s4, s5 : SessionState, p : Postpone, cod : CallForOrdersOfDay, root :
RootMotion |
		adopt[s1, s2, p] and move[s2, s3, root] and move[s3, s4, cod] and adopt[s4, s5, cod]
		and s5.immediatePendingQuestion = p.isAppliedTo
}
run call_for_a_postponed_order_of_the_day for 4 but 7 SessionState
expect 1



pred coming_back_to_a_postponed_order_of_the_day_after_a_normal_disposeOf {
		some s1, s2, s3, s4 : SessionState, p : Postpone, root : RootMotion |
			adopt[s1, s2, p] and move[s2, s3, root] and disposeOf[s3, s4, root]
			and s4.immediatePendingQuestion = p.isAppliedTo and some
(s4.currentBusiness & p.^takesOver.dealsWith)
}
run coming_back_to_a_postponed_order_of_the_day_after_a_normal_disposeOf for 4 but 7
SessionState expect 1

pred
coming_back_to_a_postponed_order_of_the_day_after_a_normal_disposeOf_and_returning_to_norma
l {
		some s1, s2, s3, s4, s5: SessionState, p : Postpone, root1, root2 : RootMotion |
			adopt[s1, s2, p] and move[s2, s3, root1] and disposeOf[s3, s4, root1]
```

```
                and s4.immediatePendingQuestion = p.isAppliedTo and some
(s4.currentBusiness & p.^takesOver.dealsWith)
                and disposeOf[s4, s5, root2] and s5.currentBusiness = (s3.currentBusiness).
(s3.nextBusiness)
}
run
coming_back_to_a_postponed_order_of_the_day_after_a_normal_disposeOf_and_returning_to_norma
l for 4 but 8 SessionState
expect 1

pred call_for_a_postponed_order_of_the_day_in_a_wholeSession {
        whole_session
        some s1, s2, s3, s4, s5 : SessionState, p : Postpone, cod : CallForOrdersOfDay, root :
RootMotion |
                adopt[s1, s2, p] and move[s2, s3, root] and move[s3, s4, cod] and adopt[s4, s5, cod]
                and s5.immediatePendingQuestion = p.isAppliedTo
}
run call_for_a_postponed_order_of_the_day_in_a_wholeSession  for 4 but 9 SessionState expect 1

pred
coming_back_to_a_postponed_order_of_the_day_after_a_normal_disposeOf_and_returning_to_norma
l_in_a_wholeSession {
                whole_session
                some s1, s2, s3, s4, s5 : SessionState, p : Postpone, root1, root2 : RootMotion |
                        adopt[s1, s2, p] and move[s2, s3, root1] and disposeOf[s3, s4, root1]
                        and s4.immediatePendingQuestion = p.isAppliedTo and some
(s4.currentBusiness & p.^takesOver.dealsWith)
                        and disposeOf[s4, s5, root2] and s5.currentBusiness =  (s3.currentBusiness).
(s3.nextBusiness)
}
run
coming_back_to_a_postponed_order_of_the_day_after_a_normal_disposeOf_and_returning_to_norma
l_in_a_wholeSession
for 4 but 9 SessionState
expect 1


                                                                        /*****************
                                                                SHOULD_FAIL

*****************/
                                                                pred SHOULD_FAIL{}

 /**************************
        SUBSIDIARY_MOTIONS
  **************************/
pred some_Commit_Applied_to_non_consecutive_secondary_parent {
        some commit : Commit |
                no (commit.isAppliedTo & commit.takesOver) and
```

```
		no (commit.isAppliedTo & RootMotion)
}
run some_Commit_Applied_to_non_consecutive_secondary_parent for 7 but exactly 1 SessionState,
exactly 1 Business
expect 0

pred some_PreviousQuestion_Applied_to_non_consecutive_secondary_parent {
	some previousQuestion : PreviousQuestion |
		no (previousQuestion.isAppliedTo & previousQuestion.takesOver) and
		no (previousQuestion.isAppliedTo & RootMotion)
}
run some_PreviousQuestion_Applied_to_non_consecutive_secondary_parent for 7 but exactly 1
SessionState, exactly 1 Business
expect 0

 /*************************
	PRIVILEGED-MOTIONS
  *************************/

pred moving_CallForOrdersOfDay_when_one_Business {
	some disj before, after : SessionState, motion :  CallForOrdersOfDay | move[before, after,
motion]
	and one Business
}
run moving_CallForOrdersOfDay_when_one_Business for 5
expect 0


/*************************
 ***** BUSINESSES *****
*************************/
pred BUSINESSES {}
run BUSINESSES {}


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

	SHOULD_SUCCEED

**********************/
                                                                pred
SHOULD_SUCCEED{}


                                                                  /*****************
                                                                 SHOULD_FAIL

*****************/
                                                                pred SHOULD_FAIL{}
```

```
pred a_sessionstate_with_an_initial_discontinuous_order_of_business {
	dynamic_mode_only and not (firstSessionState.orderOfBusiness =
firstSessionState.firstBusiness.*(firstSessionState.(nextBusiness)))
}
run a_sessionstate_with_an_initial_discontinuous_order_of_business for 5 but 1 SessionState
expect 0

//TODO : idscontinutous non initial

/**********************
 * SESSION STATES *
**********************/
pred SESSION_STATES{}
run SESSION_STATES{}


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

	SHOULD_SUCCEED

**********************/
                                                                pred
SHOULD_SUCCEED{}

assert all_sessionStates_have_a_next_but_the_last {
		some SessionState => (one ss : SessionState | no ss.next and
			all ss : SessionState - ss | one ss.next )
}
check all_sessionStates_have_a_next_but_the_last for 5
expect 0

assert all_sessionStates_have_a_previous_but_the_first {
		some SessionState => (one ss : SessionState | no ss.prev and
			all ss : SessionState - ss | one ss.prev )
}
check all_sessionStates_have_a_previous_but_the_first for 5
expect 0

assert
all_sessionStates_are_next_to_one_previous_sessionStates_and_previous_to_one_next_sessionStates {
	all ss1 : SessionState-firstSessionState | one next.ss1
	all ss2 : SessionState-lastSessionState | one prev.ss2
}
check
all_sessionStates_are_next_to_one_previous_sessionStates_and_previous_to_one_next_sessionStates
for 5
expect 0
```

```
                                                                /*****************
                                                               SHOULD_FAIL

*****************/
                                                                        pred SHOULD_FAIL{}

pred a_session_with_no_order_of_business {
        some ss : SessionState | no ss.orderOfBusiness
}
run a_session_with_no_order_of_business for 5  expect 0

module tests_model0_state

open model0

                                        /*********************************
                                        *CAN BE ENABLED/DISABLED *
                                        *********************************/
                                        fact CAN_BE_ENABLED_DISABLED {
                                        // enable_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 AND CO  *
 ************************************/
pred ORDER_OF_BUSINESS{}
run ORDER_OF_BUSINESS

//_____SHOULD_SUCCEED____//
  pred SHOULD_SUCCEED {}

pred businesses_not_part_of_some_orderOfBusiness {
//        some b : Business | b not in SessionState.orderOfBusiness
}
run businesses_not_part_of_some_orderOfBusiness for 2  expect 1
```

```
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

//assert the_pendingQuestions_are_on_one_precedence_branche { //TODO : how to update?
//	all ss : SessionState |
//	(ss.pendingQuestions-ss.immediatePendingQuestion) in
ss.immediatePendingQuestion.^takesOver
//}
//check the_pendingQuestions_are_on_one_precedence_branche 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 {//TODO
	some ss : SessionState |
		not lone ss.orderOfBusiness
		and not ss.firstBusiness = ss.lastBusiness
		and whole_session and reach_all_motions
}
run show_a_session_with_a_non_null_orderofBusiness for 5
expect 1


//_____SHOULD_FAIL_____//
pred SHOULD_FAIL {}

pred some_non_final_sessionState_with_no_currentBusiness {
	some ses : SessionState | not finalSession[ses] and no ses.currentBusiness
}
run some_non_final_sessionState_with_no_currentBusiness for 5
expect 0

/**
 *!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 5
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 {}

//pred an_unstable_pendingQuestions_set {
	//#1 not equal when adopt and motion not LayOnTheTable
	//#2 not decrease of one when dispose
	//#3 not increase of one when move
//}

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

pred a_disposedOf_moton_underPreviousQuestion_order {
	some b, a : SessionState , m : Motion | m in b.underPreviousQuestionOrder and disposeOf[b, a,
m]
	reach_all_motions
}
run a_disposedOf_moton_underPreviousQuestion_order for 5 but 2 Motion, 7 SessionState
expect 1

pred commit_a_motion_underPreviousquestionOrder{
some b, a : SessionState, c : Commit, m : Motion |
				adopt[b,a,c]
				and m in c.isAppliedTo
				and m in b.underPreviousQuestionOrder
				SecondaryMotion.isAppliedTo in RootMotion
				reach_all_motions
}
run commit_a_motion_underPreviousquestionOrder for 5 but 7 SessionState
expect 0
/**********************
 *	MISCELLANEOUS  *
 **********************/
pred MISCELLANEOUS{}
run MISCELLANEOUS

//_____SHOULD_SUCCEED____//
  pred SHOULD_SUCCEED {}


assert
no_RootMotion_not_pending_nor_toComeToOrder_nor_to_come_to_existance_when_only_dynamic_
mode {
	dynamic_mode_only => (no rm : RootMotion |
				rm not in (SessionState.(pendingQuestions+toComeToOrder)
+
RaiseQuestionOfPrivilege.questionOfPrivilege ) )
}
check
no_RootMotion_not_pending_nor_toComeToOrder_nor_to_come_to_existance_when_only_dynamic_
mode 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 SET OSCILLATION  *
                         *********************************/
                        pred STATE_SET_OSCILLATION  {}
                                run STATE_SET_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 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 |
		after = before.next and
		motion in after.pendingQuestions and
		motion not in before.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 |
		after = before.next and
		motion not in after.pendingQuestions and
		motion in before.pendingQuestions and
		not move[before, after, motion] and
		not adopt[before, after, motion]  and
		not disposeOf[before, after, motion] and
		(no pi : PostponeIndefinitely | adopt[before, after, pi ] ) and
		(no c : Commit | adopt[before, after, c]) and
		(no lot : LayOnTheTable | adopt[before, after, lot]) and
		not disposeOf[before, after, motion.questionOfPrivilege]
}
run a_motion_illegally_outside_the_pendingQuestions for 5 but 6 SessionState
expect 0

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 cod : CallForOrdersOfDay | adopt[before, after, cod] and
                                getBusiness[cod] = before.currentBusiness and
                                getBusiness[motion] = after.currentBusiness ) and
            (no p : Postpone | p in before.adopted and p in motion.yieldsTo and
                                not getBusiness[motion] = before.currentBusiness and
                                 getBusiness[motion] = after.currentBusiness ) and
            (no cod : CallForOrdersOfDay | cod in before.adopted and cod in motion.yieldsTo and
                                not getBusiness[motion] = before.currentBusiness and
                                 getBusiness[motion] = after.currentBusiness )
}
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{//TODO :definitely deal with the crossed
questions of privilege
    (all rqp : RaiseQuestionOfPrivilege |
```

```
		rqp not in rqp.questionOfPrivilege.^yieldsTo.questionOfPrivilege.^yieldsTo) and
		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{
	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( not
					( some root: before.currentBusiness.isDealtWithBy|
disposeOf[before, after, root])
					and (not after.currentBusiness =  before.currentBusiness.
(before.nextBusiness) )
						)
				)
}
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_relation_illegally_inside_the_nexBusiness{
	some rel : SessionState.nextBusiness, before, after : SessionState |
		after = before.next and
		rel not in before.nextBusiness and
```

```
                rel in after.nextBusiness
}
run a_relation_illegally_inside_the_nexBusiness for 5 but 7 SessionState
expect 0

pred a_motion_illegally_outside_the_nextBusiness {
        some rel : SessionState.nextBusiness, before, after :  SessionState |
                after = before.next and
                rel in before.nextBusiness and
                rel not in after.nextBusiness and
                before.currentBusiness.rel = after.currentBusiness
                and (no r : before.currentBusiness.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_nextBusiness for 5
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]
}

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{
```

```
		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{
	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 {
```

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

        module tests_model1_actions

open model1

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

                                    /*************************
                                *  ACTIONS TESTING  *
                                     *************************/
                                    pred ACTIONS_TESTING{}



                                    /************
                                *  MOVE *
                                     ************/
                                    pred MOVE{}
                                     run MOVE

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

run move for 3 but
exactly 1 Business
expect 1


pred a_moved_RootMotion {
        some before, after : SessionState, root : RootMotion | move[before, after, root]
}
run a_moved_RootMotion for 3 but
         exactly 1 Business
```

```
expect 1

pred a_moved_SecondaryMotion {
	some before, after : SessionState, sec : SecondaryMotion|  move[before, after, sec]
}
run a_moved_SecondaryMotion for 10 but
	 exactly 1 Business
expect 1

pred a_motion_moved_on_top_of_a_motion_under_pq_order {
	some disj before, after : SessionState, motion : Motion |
		move[before, after, motion] and some (motion.takesOver) & (before.
(underPreviousQuestionOrder))
}
run a_motion_moved_on_top_of_a_motion_under_pq_order for 5 but 8 SessionState
expect 1 //expecting motion  = doq

pred another_than_divisionOfQuestion_moved_on_top_of_a_motion_under_pq_order {
	some disj before, after : SessionState, motion : Motion |
		move[before, after, motion] and
		some (motion.takesOver) & (before.(underPreviousQuestionOrder)) and
		motion not in DivisionOfQuestion
}
run another_than_divisionOfQuestion_moved_on_top_of_a_motion_under_pq_order for 5 but 6
SessionState
expect 1

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

pred a_motion_moved_on_top_of_an_adpted_motion {
	some disj before, after : SessionState, motion : Motion |
		move[before, after, motion] and some (motion.takesOver) & (before.adopted)
}
run a_motion_moved_on_top_of_an_adpted_motion for 5 but 8 SessionState
expect 0

pred a_motion_moved_on_top_of_a_motion_under_lot_c_or_p_order {
	some disj before, after : SessionState, motion : Motion |
		move[before, after, motion]
		and some  (motion.takesOver) & (before.
(laidOnTheTable+underCommitOrPostponeIndefinitelyOrder))
}
run a_motion_moved_on_top_of_a_motion_under_lot_c_or_p_order for 5 but 8 SessionState
expect 0


		/************
	*  ADOPT *
```

```
                                **********/
                               pred ADOPT{}
                                run ADOPT

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

run adopt for 4 but
        exactly 1 Business
expect 1


pred an_Adopted_RootMotion {
        some before, after : SessionState, root : RootMotion|  adopt[before, after, root]
}
run an_Adopted_RootMotion for 4 but
        exactly 1 Business
expect 1

pred an_Adopted_SecondaryMotion {
        some before, after : SessionState, sec : SecondaryMotion|  adopt[before, after, sec]
}
run an_Adopted_SecondaryMotion for 10 but
        exactly 1 Business
expect 1

pred show_PostponeIndefinitely_adopted {
        some before, after : SessionState, pi : PostponeIndefinitely | adopt[before, after, pi]
}
run show_PostponeIndefinitely_adopted for 5 but 6 SessionState
expect 1

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

pred
an_adopted_doq_applyingTo_a_motion_under_pq_order_when_all_separated_not_under_same_order
{
        some before, after : SessionState, doq : DivisionOfQuestion | adopt [before, after, doq] and
                        doq in after.adopted and
                        some (doq.isAppliedTo & before.underPreviousQuestionOrder) and
                        doq.separatedQuestions not in after.underPreviousQuestionOrder
}
run
an_adopted_doq_applyingTo_a_motion_under_pq_order_when_all_separated_not_under_same_order
for 5 but 8 SessionState
expect 0

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

```
                        *  DISPOSE OF *
                              ******************/
                             pred DISPOSE_OF{}
                              run DISPOSE_OF

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

run disposeOf for 3 but
        exactly 1 Business
expect 1


pred a_disposedOf_RootMotion {
        some before, after : SessionState, root : RootMotion |  disposeOf[before, after, root]
}
run a_disposedOf_RootMotion for 5 but
        exactly 1 Business
expect 1

pred a_disposedOf_SecondaryMotion {
        some before, after : SessionState, sec : SecondaryMotion |  disposeOf[before, after, sec]
}
run a_disposedOf_SecondaryMotion for 5 but
        exactly 1 Business
expect 1

pred show_dispoal_of_adopted_divisionOfQuestion_takingOver_PostPoneIndefinitely {
        some before, after :SessionState, doq : DivisionOfQuestion, pi : PostponeIndefinitely |
                disposeOf[before, after, doq] and doq in before.adopted and doq.takesOver = pi
}
run show_dispoal_of_adopted_divisionOfQuestion_takingOver_PostPoneIndefinitely for 5 but 8
SessionState
expect 1

pred show_disposingOf_rootMotion_and_staying_on_same_business_then_moving {
        some disj before, current, after : SessionState, r1, r2 : RootMotion | disposeOf[before, current,
r1] and
                        before.currentBusiness = current.currentBusiness and
                        move[current, after, r2]
}
run show_disposingOf_rootMotion_and_staying_on_same_business_then_moving for 5
expect 1

pred
show_disposingOf_rootMotion_and_staying_on_same_business_then_moving_and_reach_finalState {
        some disj before, current, after : SessionState, r1, r2 : RootMotion | disposeOf[before, current,
r1] and
                        before.currentBusiness = current.currentBusiness and
```

```
                move[current, after, r2]
                whole_session
}
run
show_disposingOf_rootMotion_and_staying_on_same_business_then_moving_and_reach_finalState
for 5 but 8 SessionState
expect 1

pred show_swithch_to_other_main_motion_dealing_with_same_business {
        some disj before, after : SessionState, r : RootMotion | disposeOf[before, after, r]
        one Business
        not lone RootMotion
        whole_session
        no RootMotion&(lastSessionState.toComeToOrder)
}
run show_swithch_to_other_main_motion_dealing_with_same_business for 5 but 8 SessionState
expect 1

/*pred show_disposingOf_rootMotion_and_changing_of_current_business {

        some disj before, after : SessionState, r : RootMotion | disposeOf[before, after, r]
                and after.currentBusiness = before.currentBusiness.(before.nextBusiness)
}
run show_disposingOf_rootMotion_and_changing_of_current_business for 5 but 5 SessionState
expect 1*/

/*pred show_disposingOf_rootMotion_and_changing_of_current_business_and_reach_finalState {
        some disj before, after : SessionState, r : RootMotion | disposeOf[before, after, r] and
                not before.currentBusiness = after.currentBusiness
                and whole_session
}
run show_disposingOf_rootMotion_and_changing_of_current_business_and_reach_finalState for 5
expect 1*/

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

pred
dispoal_of_adopted_divisionOfQuestion_and_not_loosing_whatever_between_it_and_appliedTo_moti
on {
        some before, after :SessionState, doq : DivisionOfQuestion |
                disposeOf[before, after, doq] and doq in before.adopted
                and not after.immediatePendingQuestion = doq.isAppliedTo
}
run
dispoal_of_adopted_divisionOfQuestion_and_not_loosing_whatever_between_it_and_appliedTo_moti
on for 5 but 8 SessionState
expect 0
```

```
                    /**********************
                *  LOSE (implicit) *
                    **********************/
                        pred LOSE{}
                            run LOSE

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

pred a_lost_RootMotion {
        some disj before, after : SessionState, root : RootMotion|
        move[before.prev, before, root] and
        not adopt[before, after, root] and
        disposeOf[before, after, root]
        //and not doNothing[before, after]

}
run a_lost_RootMotion for 5 but
         exactly 1 Business
expect 1

pred a_lost_SecondaryMotion {
        some before, after : SessionState, root : SecondaryMotion|
        move[before.prev, before, root] and
        not adopt[before, after, root] and
        disposeOf[before, after, root]
        //and not doNothing[before, after]
}
run a_lost_SecondaryMotion for 10 but
          5 Int
expect 1

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

                    /**********************
                *  DONOTHING     *
                    **********************/
//                        pred DONOTHING{}
//                        run DONOTHING


//----SHOULD-SUCCEED-----//


//pred leaving_the_start_state_by_doing_nothing{
//        some disj before, after : SessionState | initialSession[before] and doNothing[before, after]
//}
//run leaving_the_start_state_by_doing_nothing for 10 but  exactly 1 Business
```

```
//pred reaching_the_end_satate_by_doing_nothing {
//	some disj before, after :  SessionState | finalSession[before] and doNothing[before, after]
//}
//run reaching_the_end_satate_by_doing_nothing for 10 but  exactly 1 Business

//pred an_instance_where_doing_nothing_is_allowed_only_at_the_end_of_a_session_trace {
//		all disj before : SessionState , after : before.next |
//		(doNothing[before, after]) =>
//			finalSession[after]
//
//		no disj before, after : SessionState |
//		after = before.next and
//		doNothing[before, after] and
//		some after.next and
//		not doNothing[before.prev, before] and
//		not doNothing[after, after.next]
//}
//run an_instance_where_doing_nothing_is_allowed_only_at_the_end_of_a_session_trace for 10 but
exactly 1 Business
//expect 1


//-----SHOULD FAIL-----//

//pred doing_noting_at_least_once_between_two_other_actions {
//
//	some disj before, after : SessionState |
//		doNothing[before, after] and
//		some after.next and some before.prev and
//		not doNothing[before.prev, before] and
//		not doNothing[after, after.next] //manque la conditions que before.next = after !!
//
//}
//run doing_noting_at_least_once_between_two_other_actions for 10 but  exactly 1 Business
//expect 0

//pred some_hole_trace_where_one_only_does_nothing {
//	no disj before, after : SessionState |  not doNothing[before, after]
//	some final : SessionState | no final.next and finalSession[final]
//}
//run some_hole_trace_where_one_only_does_nothing for 5 but  exactly 1 Business expect 0
```