```
CB{  % The file starts with Composition

  turns, magnitude:single, ordering:strict, max:$UserDefined$ ;
  % Turns: single move per turn with strict turntaking (a-b-a-b-etc), Dialogue Rule R1. % Max
turns is user defined (strategic rule i)

   roles speaker listener winner  ;
  % Rolelist: there are 2 roles speaker & listener

   players min:2 max:2  ; % Participants
   player id:black  ; % Player
   player id:white  ; % Player

   store id:CS owner:black structure:set visibility:public  ;
   store id:CS owner:white structure:set visibility:public  ;
  % commitment stores for both players

   transforce {<challenge, p>} {<statement, q>} arguing {<q, p>, Inference_Scheme }
   transforce {question, p>} {<statement, !p>} contradicting {<!p, p>, Conflict_Scheme }

   backtrack off

  % In the next part the DGDL Rules are given, these are mostly CB's strategic rules
   rule StartingRule scope:initial
         assign(black, speaker) & move(add, next, statement, p) & move(add, next, withdraw, p) &
move(add, next, question, p) & move(add,
         next, challenge, p) & move(add, next, withdrawPlusChallenge, p   ;

   rule SpeakerWins1 scope:turnwise
            if  numturns(CB,max) & magnitude(CS, speaker, greater, CS, listener)
            then  status(terminate,CB) & assign(speaker, winner)   ;

   rule ListenerWins1 scope:turnwise
            if  numturns(CB,max) & magnitude(CS, listener, greater, CS, speaker)
            then  status(terminate,CB) & assign(listener, winner)   ;
  % the above rules encode strategic rules ii and iv

   rule SpeakerWins2 scope:turnwise
        if  inspect(in, p, CS, speaker, initial) & inspect(in, q, CS, listener,   current) &
extCondition(ImmediateConsequence(q,p))
           then  status(terminate,CB) & assign(speaker, winner)    ;

   rule ListenerWins2 scope:turnwise
```

```
      if  inspect(in, p, CS, listener, initial) & inspect(in, q, CS, speaker, current) &
extCondition(ImmediateConsequence(q,p))
          then  status(terminate,CB) & assign(speaker, listener)    ;
  % the above rules encode strategic rule iii. extCondition calls an external function. In Simon's
DGDL this does not have parameters but we need
    them. ImmediateConsequence(q, p) calls some theorem-prover that determines whether q is
an immediate consequence of p (i.e., whether, q can be
    derived from p in one step)

  % And the Interactions (locution rules and dialogue rules)

   interaction statement asserting p "State"
           if extCondition(ImmediateConsequence, {q, p}) & inspect(in, q, CS, listener)
         then  store(add, p, CS, listener) & store(add, p, CS, speaker) & move(add, next,
statement, r) & move(add, next, withdraw, r, {extCondition(ImmediateConsequence, {q,r}) &
inspect(!in, q, CS, listener) & inspect(in, r, listener)}) &
               move(add, next, question, r) & move(add, next, challenge, r) & move(add, next,
withdrawPlusChallenge, r, {extCondition(ImmediateConsequence, {q,r}) & inspect(!in, q, CS,
listener) & inspect(in, r, listener)})
         else  store(add, p, CS, speaker) & move(add, next, statement, r) & move(add, next,
withdraw, r, {extCondition(ImmediateConsequence, {q,r}}) & inspect(!in, q, CS, listener) &
inspect(in, r, listener)) & move(add, next, question, r) &
               move(add, next, challenge, r) & move(add, next, withdrawPlusChallenge, r,
{extCondition(ImmediateConsequence, {q,r}) & inspect(!in, q, CS, listener) & inspect(in, r,
listener)})    ;

  % This defines a statement (assertion) speech act. Basically, when a statement is made, it is
checked whether the statement is the immediate
    consequence of something in the listener's commitment store. If this is the case, the
statement is added to both speaker's and listener's
    commitment stores. Otherwise, only the speaker becomes committed.

   interaction, withdraw, withdrawing, p}, "No commitment",
           store(remove, p}, CS, speaker) & move(add, next, statement, r}) & move(add, next,
withdraw, r}, {extCondition(ImmediateConsequence, {{q},{r}}) & inspect(!in, {q}, CS, listener) &
inspect(in, {r}, listener)}) & move(add, next, question, {r}) & move(add, next, challenge, {r}) &
move(add, next, withdrawPlusChallenge, {r}, {extCondition(ImmediateConsequence, {{q},{r}}) &
inspect(!in, {q}, CS, listener) & inspect(in, {r}, listener)}) } } ;
          % A withdraw removes something from the CS of the speaker

  { interaction, question, questioning, {p}, "?",
      { move(add, next, statement, {p}) & move(add, next, Statement, {q}, {extCondition(Negation,
{{p},{q}})}) & move(add, next, withdraw, {p}, {extCondition(ImmediateConsequence, {{q},{p}}) &
```

```
inspect(!in, {q}, CS, listener) & inspect(in, {p}, listener)}) } };
          % A question demands an answer ("yes, p", "no, !p" or "I'm not committed to p")

  { interaction, challenge, challenging, {p}, "Why?",
       { store(add, {p}, CS, listener) & move(add, next, withdraw, {p},
{extCondition(ImmediateConsequence, {{q},{p}}) & inspect(!in, {q}, CS, listener) & inspect(in, {p},
listener)}) &  move(add, next, withdrawpluschallenge, {p},
{extCondition(ImmediateConsequence, {{q},{p}}) & inspect(!in, {q}, CS, listener) & inspect(in, {p},
listener)}) & move(add, next, statement, {q}, extCondition(Consequence, {{q},{p}}}) } } };
       % A challenge adds something to the listener's CS and mandates that he reply by either
withdrawing the challenged proposition or state
       something that provides a reason for p. Consequence(p, q) calls some theorem prover that
determines whether p is a consequence of q (i.e.,
       whether, p can be derived from q in a finite number of steps)

  { interaction, WithdrawPlusChallenge, withdrawing, {p}, challenging, {p}, "No commitment,
why?",
       { store(remove, {p} , CS, speaker) & store(add, {p} , CS, listener) & move(add, next,
withdraw, {p}, {extCondition(ImmediateConsequence, {{q},{p}}) & inspect(!in, {q}, CS, listener) &
inspect(in, {p}, listener)} ) & move(add, next, withdrawpluschallenge, {p},
{extCondition(ImmediateConsequence, {{q},{p}}) & inspect(!in, {q}, CS, listener) & inspect(in, {p},
listener)}) & move(add, next, statement, {q}, extCondition(Consequence, {{q},{p}}})} } };
```