```
PP{

  { turns, magnitude:single, ordering:strict} ;

  % turns simply alternate between pro and opp


  { roles, speaker, listener, winner } ;

  % Prakken also defines a winner


  { players, min:2, max:2 } ;

  { player, id:pro } ;

  { player, id:opp } ;


  { store, id:CS, owner:pro, structure:set, visibility:public } ;

  { store, id:CS, owner:opp, structure:set, visibility:public } ;


  { backtrack, on }; % backtracking is allowed


  { rule, StartingRule, scope:initial,

    { assign(pro, speaker) & move(add, next, claim, {p}, pro) & move(add, next, argue, {<{Q},{p}>,
DefaultInference}, pro) }} ; % pro starts (Def 6 p 8) and must always claim or argue (R6 p 11)


 { rule, Termination, scope:movewise,

    { if {size(LegalMoves, empty)}

      then {status(terminate, PP)} } }; % Game terminates when no legalmove possible.


  { interaction, claim, {p}, asserting, {p}, "Claim ",

          { store(add, {p}, CS, speaker) & move(add, future, challenge, {p}, listener) & move(add,
```

future, concede, {p}, listener) } } ; % after a claim you can challenge or concede (table 1 p 10)

{ interaction, challenge, {p} challenging, {p}, "Why?",

{ move(add, future, argue, {<{Q},{p}>, DefaultInference}, listener) & move(add, future, retract, {p}, listener) } } ; % after a challenge you can argue or retract (table 1 p 10)

{ interaction, argue, {<{Q},{p}>, DefaultInference}, asserting, {Q}, asserting, {p}, arguing, {<{Q},{p}>, Inference_Scheme}, "Argue ",

if { responds(challenge, {p}) }

then { store(add, Q, CS, speaker) & store(add, {p}, CS, speaker) & move(add, future, argue, {<{S},{t}>, DefaultInference}, listener, {extCondition(Defeats, {{<{S},{t}>, DefaultInference},{<{Q},{p}>, DefaultInference}})}) & move(add, future, challenge, Q, listener) & move(add, future, concede, Q, listener) }

else { store(add, Q, CS, speaker) & store(add, {p}, CS, speaker) & move(add, future, argue, {<{S},{t}>, DefaultInference}, listener, {extCondition(Defeats, {{<{S},{t}>, DefaultInference},{<{Q},{p}>, DefaultInference}})}) & move(add, future, challenge, Q, listener) & move(add, future, concede, Q, listener) & move(add, future, concede, {p}, listener)}} ; %if the move responds to a challenge move, the listener cannot concede the conclusion (concede p is already in listener's movelist because at some point a challenge p was moved, R7 p 11)

{ interaction, concede, {p}, asserting, {p}, "Concede ",

{store(add, {p}, CS, speaker) & move(delete, future, challenge, {p}, speaker) & move(delete, future, argue, {<{S},{t}>, DefaultInference}, speaker, {extCondition(Negation, {{t},{p}})}) }} ; %once you concede p you cannot challenge it or rebut it with an argument in a later move (R5 p8 and sec 3.1)

{ interaction, retract, {p}, withdrawing, {p}, "Retract ",

{ store(remove, {p}, CS, speaker) & move(delete, future, argue, {<{Q},{p}>, DefaultInference}) } ; %once you retract p you cannot argue for it in a later move

}