# Specification and Implementation of Toulmin Dialogue Game

Trevor J.M. Bench-Capon

*LIAL - Legal Informatics at Liverpool, Department of Computer Science,*
*University of Liverpool, Liverpool, England*
*tbc@csc.liv.ac.uk*

## Abstract

In this paper I describe the specification and implementation of a dialogue game based on the argument schema of Toulmin. It is argued that this schema is particularly suited to legal dialogues, and the need for the additional expressiveness provided by a game based on it, rather than logic alone, is illustrated by a sample dialogue. A full specification is given and an implementation, allowing the game to be played by human players, is described. Issues relating to the design of a computer program to play the game are discussed, and some concluding remarks are made.

## 1 Introduction

In this paper I shall give a detailed account of the specification and implementation of a Dialogue Game, based on the schema for arguments of Toulmin (1958), modified in several ways. The game is a development of that first described in Bench-Capon et al. (1992). In section 2, I will describe the schema and its extensions. In section 3, I shall motivate the use of this schema as a basis for a dialogue game by use of an example dialogue. In section 4, I shall specify the game to be implemented, by giving a state transition diagram to show the moves that are legal at various points in the game, and a specification of the semantics of each move in the game in terms of preconditions for their use, postconditions which update the state of the game, and completion conditions which state when the move is terminated. In section 5, I shall describe a Prolog implementation of the game, intended to support menu driven interaction between two human players. In section 6, I shall I shall discuss the issues which surround implementing a computer player of this game. Finally, I shall offer some concluding remarks.

The game described (called Toulmin Dialogue Game or TDG) differs from many other games discussed in the literature in important ways.

First, these other games are typically heavily based on a formal logic whereby an argument is seen as a set of premises which entail a particular conclusion. These games have been used to enforce proper logical conduct (e.g. Mackenzie 1979), to explore various relations between premises and conclusions, and between arguments relating to the nonmonotonic nature of argument (e.g. Prakken and Sartor 1995) and to model a particular legal practice (Gordon 1994). In contrast the structure of arguments proposed by Toulmin differentiates a number of different roles for premises, which give arguments a richer structure, and one which corresponds more closely to the way in which arguments are presented. Thus the point of TDG is not so much to enforce logically correct behaviour as to ensure that the final argument which emerges from the dialogue has the structure prescribed by Toulmin. The result is a game with more varied moves which we believe, and hope to show, enables a more faithful modelling of the kinds of dialogues found in everyday discourse. Moreover, these additional elements are considered important for representing legal arguments.

Second, the game is intended to be co-operative in that the participants are not intending to "win", but rather to arrive at a position where there is a supported claim on which they agree, together with a fully formed supporting argument structure. This is particularly important when we come to discuss strategies for playing the game.

I shall now describe Toulmin's original schema, and the modifications which we have made to it.

## 2 The Argument Schema

The original argument schema proposed in Toulmin (1958) had the form shown in Figure 1.

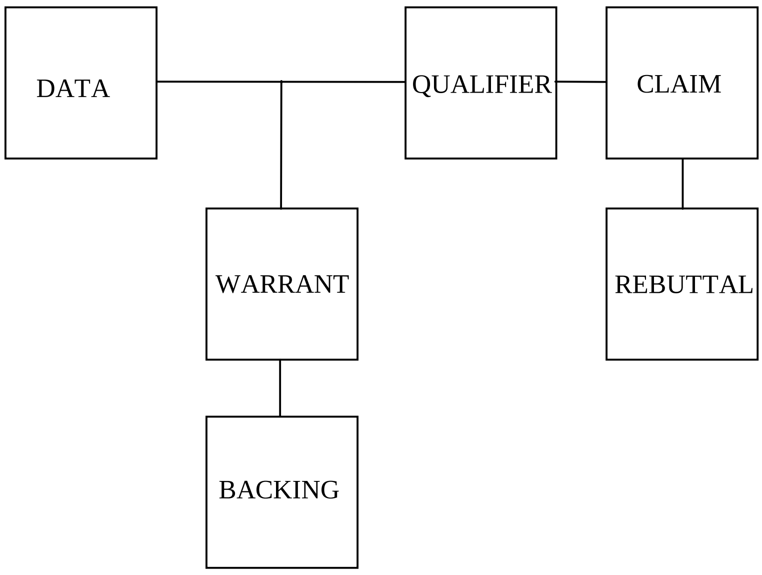

Figure 1. Toulmin's Argument Schema

In this schema we have a *claim* which is the conclusion of the argument. The *data* and the *warrant* are like traditional premises, in that the *warrant* is an implication of the form *data -> claim*. The *rebuttal*, is a proposition which while it does not need to be proven false, would if true, refute the claim. This corresponds to the idea of *non-refutandum* in Sartor (1995). The *backing* represents the authority for the warrant, and the *qualifier* the strength of the argument, whether it is intended to establish the claim necessarily, on the balance of probabilities, or whatever. Thus the structure proposed by Toulmin incorporates three elements important in legal arguments: their defeasibility, via the rebuttal, their need to appeal to extra logical justifications, through the backing, and the degree of proof required, through the qualifier.

For our purposes we have adapted Toulmin's schema to give the revised schema shown in Figure 2.

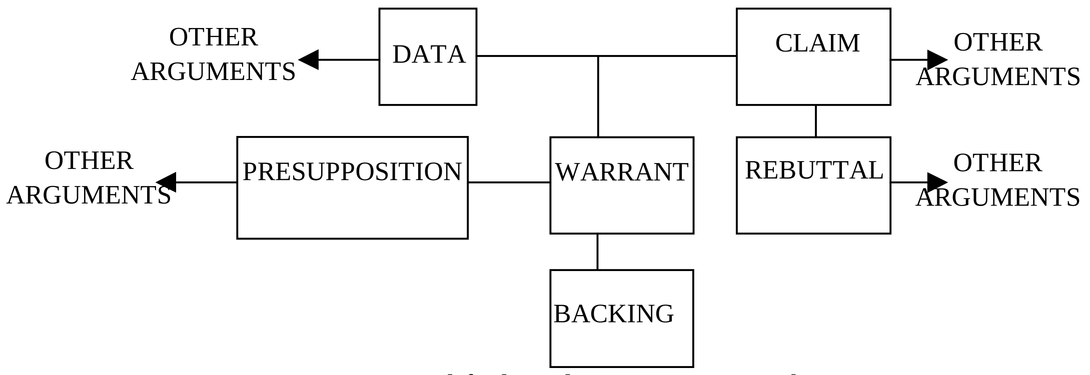

Figure 2. Modified Toulmin Argument Schema

Essentially we have omitted the qualifier, since we take the onus of proof to be agreed at the outset, allowed for chaining arguments together so that some data can be the claims of other arguments, and that claims can serve as the data for succeeding arguments, and introduced the notion of *presup-position*, which is supposed to represent propositions assumed to be true in the context, and so which do not *need* to be discussed but which can be made explicit if required. For example, we may assume in a discussion of benefit entitlement that a person is resident in the UK, but we might wish to make this explicit.

With this schema we have some flexibility in assigning particular roles to premises in an argument. In the next section I shall go through a dialogue concerning an argument about entitlement to a social security benefit to illustrate the importance of having this extra expressiveness.

## 3 A Motivating Dialogue

The particular problem being addressed here is whether someone observed gardening should be considered unfit for work, and so entitled to sickness benefit. While *prima facie* this appears so, in fact in this particular case it is not so, because gardening is not considered suitable employment for the person concerned. Annotations are supplied in italics; the propositions advanced are given identifying letters in brackets.

A1: John should not get sickness benefit. (-B)

*This is the initial claim.*

B1: Why?

*The first question simply seeks the grounds for the claim, the data in our schema.*

A2: He is a lazy person. (L)

*Some grounds are advanced.*

B2: OK, but so what?

*Here two moves are made. The truth of the data is accepted, but its relevance to the argument is questioned.*

A3: Lazy people should not get sickness benefit. (L -> -B)

*A rule which would licence the conclusion is advanced.*

B3: Is there a law to that effect?

*Here a justification for the rule is sought. Note that here we do not seek a justification in terms of a logical argument for the rule, but rather in terms the authority from which the rule derives – making use of the backing in our schema. Moreover we seek a legal, not a moral authority.*

A4: Not as such.

*No authority is available, and so the rule is withdrawn.*

B4: So why do you think he should not get the benefit? (-B)

*Now there is no reason to think that L is a reason for -B, a different reason is sought.*

A5: He is fit for work. (F)

*Another reason is supplied.*

B5: Suppose he is. Why should that matter?

*Here player B does not immediately accept the truth of the data, but wishes to defer consideration of it, while exploring the connection between F and –B.*

A6: If someone is fit for work, they should not get sickness benefit.

*The relevant rule is supplied.*

B6: Any work at all?

*Player B suspects that the rule is not always applicable and seeks the elements of the context which make it applicable to this case.*

A7: Work that someone would pay to have done. (P)

*The background assumption is given.*

B7: True, but what's your authority?

*Player B accepts that P is true, but now wishes to know the authority for the rule.*

A8: Social Security Act 1984, section 32(1).

*This time player A has an authority, in the form of a statutory provision.*

B8: Why do you say John is fit for work? (F)

*Player B accepts the rule but questions the premise, asking for grounds that support it.*

A9: I saw him gardening. (G)

*A supplies the grounds.*

B9: I can accept that. But gardening is unsuitable work for John. (G, U -> B)

*B accepts the truth of the grounds, and that there is a rule entailing F. On the face of it this would establish -B. But Player B is aware of an exceptional circumstance which can defeat the argument.*

A10: What's that got to do with it?

*Player A now wishes to know the authority for the rule implicit in B's rebuttal. Note that it is now B who is the proponent of the claim.*

B10: Fit for work means fit for suitable work. Too good, Commisssioner, RS3/58.

*B supplies an authority from case law.*

A11: Why do you say gardening is not suitable work?
*Player A is forced to accept the rule, but can question the premise.*

B11: John is a University Lecturer. (J)

*B supplies his data for U.*

A12: OK, OK. John should get the benefit. (B)

*Player accepts the truth of J, and that J implies U, and so is constrained to accept that B.*

The intention here is to present a natural dialogue, but one which shows that there are different ways of probing for information and challenging statements which require the distinctions amongst premises that we find in the argument schema. Contrast this with a less rich dialogue, based only on logical consequence, such as is found in Lodder (1998). In such a dialogue no distinction can be made between justification by argument and from authority, between presupposed context and substantive premises, or between a claim and a claim intended as a rebuttal of a prior claim. Nor is it possible to defer consideration of a claim while some previous claim is considered fully. The result is a dialogue which, while it may exhibit impeccable logic, is rather stilted, and does not show the desired structure. Similar criticism can be levelled at other logic based games such as that of Mackenzie (1979).

In the next section I will describe the flow of dialogue in TDG, and specify the various moves it requires.

## 4 Specification of TDG

In order to show what moves, and sequences of moves, are possible in TDG we use a State Transition Diagram, shown in Figure 3.

Some comments on the diagram are needed. First we should notice that there are three roles undertaken by the participants: proponent of the claim, opponent of the claim, and referee. The proponent and the opponent can change roles during the game, if a rebuttal is issued. This change is indicated by the circled node. The other, boxed, nodes represent who has control of the dialogue (is the next to play) at a given point. Each role has several nodes because the options available change with the context. The arcs entering a node indicate the moves that can bring about that state, and those leaving a state the moves that can be made in that state.

We can now specify the semantics for each move in the game. We do this by giving the preconditions for making a move, the postconditions which achieve the effects of a given move, and the completion conditions which say when a move is fully complete.

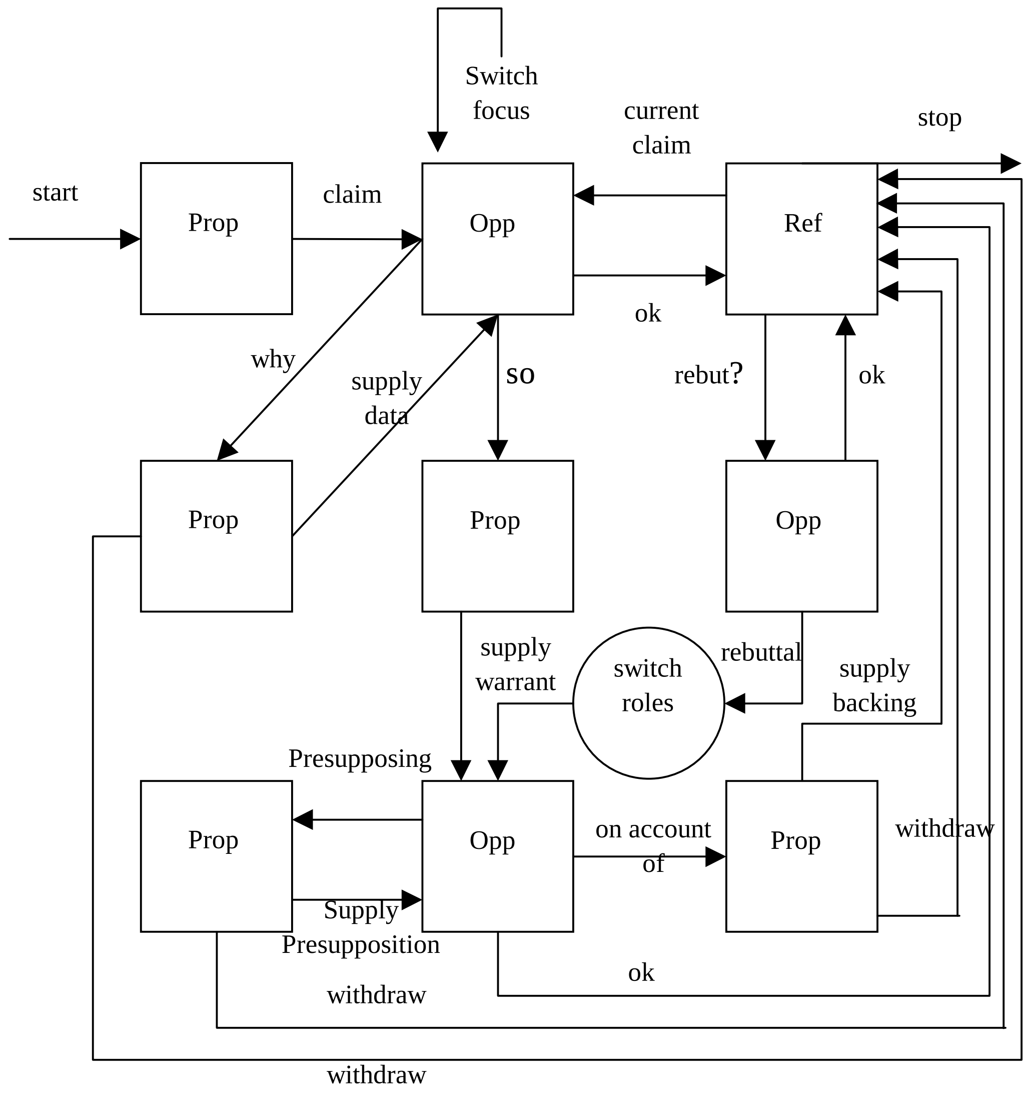

Figure 3. State Transition Diagram for TDG

The conditions are expressed in terms of events brought about by moves in the game: thus there is no appeal to intentional attitudes on the part of the players, and hence the players need make no assumptions about the beliefs and reasoning apparatus of the other players. We use two notions to describe these aspects of the game:

- *The Claim Stack*: This is a stack (in the standard computer data structure sense) of claims that have been made, either explicitly as claims, or implicitly when data or warrants are supplied.
- *The commitment stores:* Each player will become committed to the truth of certain propositions as the result of moves. The commitment stores, maintained for each player other than the referee, record these commitments. This is a standard notion found also in Mackenzie (1979) and Lodder (1998).

We can now specify each of the moves. In our specification P and O represent the roles currently occupied by the two players, proponent and opponent respectively; C, D and S are propositional variables.

## 4.1 Semantics of Moves in TDG

*claim (C)*
Description: P asserts that C
Preconditions: P has control of the dialogue
Postconditions: O has control of the dialogue
C is pushed onto the claim stack
P is committed to C
Completion Conditions: C is popped from the claim stack

*why (C)*
Description: O seeks data supporting C
Preconditions: O has control of the dialogue
C is top of claim stack
Postconditions: P has control of the dialogue
Completion Conditions: C is not top of claim stack

*OK (C)*
Description: O accepts C
Preconditions: O has control of the dialogue
C is top of claim stack
Postconditions: C is popped from the claim stack
O is committed to C
O is not committed to not C
If not C is on claim stack, it is removed
Referee has control of the dialogue
Completion Conditions: None

*So (C)*
Description: O requests the warrant for C
Preconditions: O has control of the dialogue
O is not committed to if D then C, for any D
for which he is not committed to -D
C is top of claim stack
Postconditions: P has control of the dialogue
Completion Conditions: C is not top of the claim stack

*Presupposing (C)*
Description: O requests the presupposition of C
Preconditions: O has control of the dialogue
If D then C is top of claim stack
Postconditions: P has control of the dialogue
Completion Conditions: If D then C is popped from the claim stack

*On Account Of (C)*
Description: O requests the backing for the warrant of C
Precondition: O has control of the dialogue
If D then C is top of claim stack
P has issued a supply warrant (C)
Postconditions: P has control of the dialogue
Completion Conditions: If D then C is popped from the claim stack.

*Supply Data (C)*
Description: P asserts that D and that D supports C
Preconditions: P has control of the dialogue
  O has issued a Why (C)
  C is top of the claim stack
Postconditions: P is committed to D
  D is pushed on the claim stack
  O has control of the dialogue
Completion Conditions: D is popped from the claim stack

*Supply Warrant (C)*
Description: P asserts that If D then C
Preconditions: P has control of the dialogue
  O has issued a So (C)
  C is top of the claim stack
Postconditions: P is committed to If D then C
  If D then C is pushed on to the claim stack
  O has control of the dialogue
Completion Conditions: If D then C is popped from the claim stack

*Supply Presupposition (C)*
Description: P asserts that S
Preconditions: P has control of the dialogue
  O has issued a presupposing (C)
  If D then C is top of the claim stack
Postconditions: P is committed to S
  P is committed to If not S then not C
  S is pushed on to the claim stack
  O has control of the dialogue
Completion Conditions: S is popped from the claim stack

*Supply backing (C)*
Description: P says that B is the authority for his argument for C
Preconditions: P has control of the dialogue
  O has issued an on account of (C)
  P has issued a supply warrant (C)
  If D then C is top of the claim stack
Postconditions: R has control of the dialogue
  O is committed to If D then C
  If D then C is popped from the claim stack
Completion Conditions: None

*Withdraw (C)*
Description: P withdraws his commitment to C
Preconditions: P has control of the dialogue
  C is top of the claim stack
Postconditions: C is popped from the claim stack
  P is not committed to C
  R has control of the dialogue
Completion Conditions: None

*Switch Focus (C)*
Description: O wishes to consider a claim not currently top of the claim stack
Preconditions: C is not top of claim stack
C is on the claim stack
O has control of the dialogue
O is not committed to C
Postconditions: C is moved to top of claim stack
O has control of the dialogue
Completion Conditions: None

*Current Claim (C)*
Description: The referee passes control to the opponent of the current claim
Preconditions: C is top of the claim stack
Ref has control of the dialogue
Player A is committed to C
Postconditions: Player B has control of the dialogue
Completion Conditions: None

*End*
Description: The referee terminates the dialogue
Preconditions: Ref has control of the dialogue
The claim stack is empty
Postconditions: The dialogue terminates
Completion Conditions: None

*Rebut? (C)*
Description: Player is invited to rebut an implicit commitment
Preconditions: Referee has control of the dialogue
C is top of claim stack
Player is not committed to C
Player is committed to if D then C for some D
Player is committed to D
Postconditions: Player has control of the dialogue
Player is opponent, other player is proponent
Completion Conditions: C is not top of claim stack

*Rebuttal (C)*
Rebuttal is the most complicated of the moves, and is perhaps best seen as a sequence of other moves:
a. Issuer becomes proponent, other player become opponent
b. Issuer claims -C
c. Other Player issues why(-C)
d. Issuer supplies data, D
e. Other player issues a switch focus (-C)
f. Other player issues so (C)
g. Issuer supplies warrant if D then -C

All the commitment stores are updated to reflect these moves, and the claim stack is also modified as if these moves were made explicitly.

This gives the semantics of Rebuttal as:

*Rebuttal (C)*

| | |
|---|---|
| Description: | Player provides a rebuttal, D, of C |
| Preconditions: | Player has control of the dialogue |

Player is not committed to C
Other player is committed to C

| | |
|---|---|
| Postconditions: | D is pushed on to the claim stack |

Player is committed to -C
Player is committed to D
C is pushed on to claim stack
Player is committed to if D then -C
If D then -C is pushed onto claim stack
Player is proponent, other player is opponent
Opponent has control of the dialogue

| | |
|---|---|
| Completion Conditions: | -C is no longer on the claim stack |

Given this specification, we can now proceed to implementation.

## 5 Implementation of TDG

In this section we will describe the implementation of the core program, game moves and interface. The system has been implemented in Prolog.

### 5.1 Core program

At the heart of the program is a procedure which gives effect to the various moves. This procedure, called **effect** takes three arguments, the move being made, the player making the move, and the number of the move in the game:

```
effect(end,_,_).
effect(R, A, N):-move(R, A, Pre, Post, Comp, H, O),
                                check(Pre),
                                check(Post),
                                check(Comp),
                                M is N + 1,
                                report,
                                go(H ,M).
    effect(R, A, N):-write([invalid, move]), nl, go (A, N).
```

If the move is **end**, the program terminates. Otherwise it finds the appropriate move, checks that the preconditions are satisfied, applies the postconditions, increments the move number, reports the current state of the claim stack and commitment stores, and seeks the next move. Should an invalid move have been entered the second clause will fail and the third clause will report that the move was invalid and request re-input.

The moves themselves are defined with seven arguments: the name of the move and its parameters; the player making the move, a list of preconditions, a list of postconditions, a list of moves completed by the move, the player to whom control is transferred and the role of the player. As an example consider the definition of **why** (C):

```
Move(why(P), S, [opp(S), claim_stack([P|_])],
                [swap(S,H),
                   retract(open(O)),asserta(open([why(P)|O]))],
                [],
                H, opp).
```

The preconditions are as in the specification and represented using declarative statements about the role of the player and the claim stack. The postconditions are represented using procedural Prolog, and here have the effect of transferring control (the specified postcondition) and adding the move to the list of uncompleted moves. The completions list is empty since the move does not cause any others to be completed.

The lists of pre, post and completion conditions are applied using the procedure **check**:

```
check([]).
check([H|T]):-call(H),check(T)
```

which simply calls each term in the list recursively.

A more complicated move is **supply_warrant**:

```
move(supply_warrant(P,D), S,
             [prop(S),open(O), member(so(P),O),
               claim_stack([P|_])],
             [retract(com(S,Scom)),
               asserta(com(S,[[D,mi,P]|Scom])),
               retract(claim_stack(Cms)),
               asserta(claim_stack([[D,mi,P]|Cms])),
               retract(open(O)),
               asserta(open([supply_warrant(P,D)|O])),
               swap(S,H)],
             [retract(open(Q)),remove(so(P),Q,Z),
               asserta(open(Z))],
               H, prop).
```

**[D,mi,P]** here means "**D -> P**". Here we have the three preconditions as specified, but this time need to examine the list of open moves as well as the claim stack. The postconditions modify the player's commitment store and the claim stack as well as transferring control, but do not add the move to the list of uncompleted moves, since it completes itself. Additionally, however, it completes the previous so, which must therefore be removed from the list of open moves.

Similar definitions are given for each of the moves in the game.

## *5.2 Interface*

The program has been supplied with a simple menu driven interface. This uses the preconditions to find what moves are available and forms them into a list. This list is then used to generate a menu from which the user can select the desired move.

```
getMove(Player,R):-
findall(X,(move(X,_,Pre,_,_,_,Player),
                  check(Pre)),L),
                  display_menu(L,R).

display_menu(L,R):-printmenu(L,1),read(N),getnth(N,L,R),handle(R).
```

The predicate handle is used to solicit any extra information that the move requires: for example a rebuttal requires the player to enter a rebutting proposition:

```
handle(rebuttal(R,S)):-
write([what,is,your,rebuttal]),nl,read(S).
```

The move is then passed to effect, as described above.

### 5.3 Making a Machine Player

The program described above mediates between human players. In this section I shall consider what would be involved in implementing a program which could act as a participant in dialogues managed by this program. There are essentially three matters to discuss: the the knowledge required to participate properly and its representation; the principles which will enable the choice between the options presented at any given point; and the ability to parse the content of the performatives.

#### 5.3.1 Knowledge Required

To contribute fully to the argument, a program must have a knowledge base containing three elements.

- *Facts*: A set of propositions taken as true by the program.
- *Rules*: These are standard looking rules of the form “If Antecedent then Consequent”, where consequent is an atomic proposition. Horn clauses and production rules supply rules of this form. If, however, correct use is to be made of the rule, the propositions in the antecedent must be distinguished into three classes: (a) the crucial condition, which will be offered as data; (b) the background assumptions which will be offered as presuppositions; and (c) propositions whose truth need not be established, but whose falsity would serve to defeat the rule, which will be the source of potential rebuttals. Since there is nothing intrinsic to differentiate the propositions in the antecedent in this way, these distinctions must be supplied by the author of the knowledge base. Similar distinctions can be found in some default logics, and in the annotated logic programs of Bench-Capon et al. (1991). It is not necessary that every rule have all three types of proposition in its antecedent: it does not matter if there are no defeating conditions, or if that list is incomplete.
- *Rule Sources*: If the rules are to stand up in an argument they must also have a record of their source, which can be used to supply backings.

From this we see that the required knowledge base is in the usual form of a rule based knowledge base, but that the need to act within the required structure requires some extra-logical information, distinguishing the roles of the constituent propositions in the antecedents, and supplying a justification for the rules. It is contended that this is a relatively small overhead for the knowledge base author.

### *5.3.2 Choosing the Move*

A major benefit of the state transition diagram is that it clearly identifies the choice points that arise in the course of the dialogue, and the options that may be available at each of those choice points. If we examine the state transition diagram we can see that the potential for a choice of moves arises at eight of the nine states possible in the dialogue.

For the proponent the choices are relatively easy; in one state the only possible move is to make a claim, and in the other four the choice is between supplying the relevant data, warrant, presupposition or backing, respectively, or withdrawing the claim. In each case we will assume that the program will supply the relevant information if it is able to do so, and otherwise will choose to withdraw. The ability to supply the information depends on the contents of the knowledge base as follows:

- *Supply Data*: The knowledge base contains at least rule one with claim as consequent. If there is more than one such rule, if a warrant has already been supplied the data of the rule supplied as warrant should be given; otherwise the choice is arbitrary.
- *Supply Warrant*: The knowledge base contains at least one rule with claim as consequent. If there is more than one such rule, if a data has already been supplied the rule with that data in its antecedent should be given; otherwise the choice is arbitrary.
- *Supply Presupposition*: To be put in this state the program will have a given rule as a warrant; it should supply the propositions in the antecedent marked as presuppositions.
- *Supply backing*: To be put in this state the program will have a given rule as a warrant; it can supply a backing if it has the source for that rule.

The limited nature of the choices of the system was exploited in Bench-Capon et al. (1991) in the explanation of conclusions from a logic program: there the system only adopted the role of proponent, and was, since it was simply repeating a proof, always able to supply the desired information.

The choice of the referee is likewise easy; if the preconditions for issuing a *rebut?* are satisfied, this should be done, so as to give the opponent of the claim a chance to defeat the currently successful argument. Otherwise the referee should issue a *current claim*, if there is still a claim on the claim stack, and otherwise an *end*.

This leaves the three states in which the opponent of a claim is in control. Here we will order our preferences on the basis that the game is co-operative, and so will not require the system to force elaboration of an argument which it already accepts. A suggested ordering is as follows:

- After a *rebut*? the system should issue a rebuttal if one is available. One is available if the knowledge base contains a rule which is the warrant of the claim it wishes to rebut, and this rule contains some defeating condition, which it can prove from its knowledge base. If this is so, that condition can be issued as a rebuttal, otherwise it must issue an *ok*.
- After a *supply warrant* there are three options: if there is a rule in the knowledge base with claim as consequent, and data as antecedent, but with additional antecedents representing presuppositions which are not provable from the knowledge base (augmented by current commitments), it should issue a *presupposing*; otherwise if the warrant appears as part

of a rule in the knowledge base for which the knowledge base contains a source it should be accepted; otherwise it should issue an *on account of*.
- After a *supply presupposition* if the warrant appears as a part of a rule in the knowledge base for which the knowledge base a contains a source it should be accepted; otherwise it should issue an *on account of*.
- All other moves lead to a single state in which there are a possible four options. If the claim can be proved from the knowledge base (augmented by any current commitments), *ok* should be issued. Otherwise, if the knowledge base contains a rule with claim as consequent, or if the proponent is committed to such a rule, a *why* should be issued. Otherwise, if no such rule exists either in the knowledge base or in the proponents commitments, a *so* should be issued to obtain such a rule. This covers every case, and so *switch focus* will not be used by the program.Since the role of *switch focus* is only to allow for flexibility in ordering the questions to be addressed, its use is a matter of personal taste, and not essential: thus it does not matter that the program never chooses to use it.

The strategy given above is sufficient to determine the choices made at any point in the dialogue. It is co-operative, in the sense that it will not ask for argument for anything it accepts itself, but rigorous in the sense that it will highlight potential rebuttals and seek justification for rules for which it has no justification, even if it believes them itself.

### *5.3.3 Parsing the Contents of the Moves*

For the knowledge base to be effective, it must be possible to match propositions input by the user with propositions in the knowledge base. Since there is considerable scope for stylistic variation, this requires that we impose some discipline on the user.

One way to achieve this is to offer the user a menu of propositions recognised in the knowledge base. This will ensure that the phrasing is correct. Of course the users will be free to add in propositions of their own as well; these will not be recognised by the knowledge base, but this does not matter as it can be presumed that they represent information new to the system.

A second option is to provide the user with a restricted syntax, such as Prolog, together with information as to the predicates and terms recognised by the knowledge base. This is possibly more flexible, but less user friendly.

A final possibility is to attempt to provide the program with some natural language capability. This is not currently a feasible option.

### *5.3.4 At the End of the Dialogue*

At the conclusion of a dialogue the program will have a store of commitments, not all of which will be present in it knowledge base. If we wish, we could update the knowledge base to include these commitments. Whether it is desirable to do so, depends on the context of use and the authority of the user, but it could provide an interesting means of knowledge acquisition.

## 6 Conclusion

In this paper I have presented a complete specification, and discussed the implementation, of a dialogue game based on the argument schema of Toulmin. This schema has been found effective for the presentation of legal argument (see Bench-Capon (1997) for some examples of researchers who have found it so). The present dialogue game, however, goes beyond presentation, since it allows for the players to adopt the roles both of proponent and opponent of claim. New too, is the strategy which can be used to make decisions as to the move to play, which makes it possible to have a program to play the game.

I believe that I have achieved three things:

- Presented a dialogue game which allows for the modelling of arguments in a fashion more realistic than other, logic bound, dialogue games found in the literature;
- Used a method which is intended to be a generic way of specifying dialogue games, and which makes the implementation of the games relatively straightforward;
- Described a tool which could be used to explore the possibilities of the dialogue game, and which could be modified and customised to try variations of the game.

## References

Bench-Capon *et al.* (1991)
Bench-Capon, T.J.M., Lowes, D., and McEnery, A.M., Using Toulmin's Argument Schema to Explain Logic Programs, *Knowledge Based Systems*, Vol. 4, No. 3, September 1991, pp.177-83.

Bench-Capon *et al.* (1992)
Bench-Capon, T.J.M., Dunne, P.E. and Leng, P.H., *A Dialogue Game for Dialectical Interaction with Expert Systems*, 12th Annual Conference on Expert Systems and Their Applications, Avignon 1992, EC2.

Bench-Capon (1997)
Bench-Capon, T.J.M., Argument in Artificial Intelligence and Law, *Artificial Intelligence and Law*, Vol 5, No 4, 1997, pp 249-61.

Gordon (1994)
Gordon, T.F., The Pleadings Game: An Exercise in Computational Dialectics, *Artificial Intelligence and Law*, Vol 2, No 4, 1994.

Lodder ( 1998)
Lodder, A.R., *DiaLaw: On legal justification and Dialogue games*, dissertation, University of Maastricht 1998.

MacKenzie (1979)
MacKenzie, J.D., Question-Begging in Non-Cumulative Systems, *Journal of Philosophical Logic*, vol 8, 1979, pp 159-177.

Prakken and Sartor (1995)
Prakken H., and Sartor, G., On the Relation Between legal Language and Legal Argument, in: *Proceedings of the Fifth International Conference on AI and Law*, University of Maryland: ACM Press 1995, pp 1-10.

Sartor (1995)
Sartor, G., Defeasibility in Legal Reasoning, in: Bankowski, Z., White, I., and Hahn, U. (eds), *Informatics and the Foundations of Legal Reasoning*, Dordrecht: Kluwer Academic 1995, pp119-158.

Toulmin (1958)
Toulmin, S., *The Uses of Argument*, Cambridge: Cambridge University Press 1958.