Journal of Applied Logic 10 (2012) 309–329

Contents lists available at SciVerse ScienceDirect

Journal of Applied Logic

www.elsevier.com/locate/jal

A domain specific language for describing diverse systems of dialogue

S. Wells [a,*], C.A. Reed [b]

[a] School of Computing Science, Meston Building, University of Aberdeen, Aberdeen, AB24 3UE, United Kingdom
[b] School of Computing, Queen Mother Building, University of Dundee, Dundee, DD1 4HN, United Kingdom

ARTICLE INFO

Article history:
Received 14 January 2009
Accepted 30 April 2009
Available online 11 September 2012

Keywords:
Argumentation
Dialectical games
Agent communication
Online argumentation
Dialogue

ABSTRACT

This paper introduces the Dialogue Game Description Language (DGDL), a domain specific language for describing dialectical games. Communication is an important topic within agent research and is a fundamental factor in the development of robust and efficient multiagent systems. Similarly, argumentation has been recognised as a key component of an agent's ability to make decisions using complex, dynamic, uncertain, and incomplete knowledge. Dialectical games, a type of multi-player argumentative dialogue game, provide a mechanism for communication which incorporates argumentative behaviours. However there are very few tools for working with these games and little agreement over how they should be described, shared, and reused. The DGDL provides a grammar for determining whether a game description is syntactically correct and thus provides a foundation for new tools to support the future development and wider exploitation of dialectical games.

1. Introduction

Dialogue games have been proposed as a tool for modelling the interactions between participants during argumentative dialogues. One branch of dialogue game research is into the formal dialectical systems due to Hamblin [10, pp. 253–282]. Hamblin defines such systems, or *dialectical games*, as simple, two-player, turn-taking games in which the moves available to the players represent the locutional acts and utterances available to the participants in the dialogue.

Hamblin's motivation was to explore the circumstances under which certain logical fallacies, such as *petitio principii*, occur during dialogue. In response, Mackenzie subsequently developed DC [21], and Woods and Walton explored extensions to the rules of Hamblin's game as a technique for investigating fallacies [58]. Many more dialectical games have been formulated and proposed for application in a variety of dialogical situations occurring in a range of problem domains apart from fallacy research. Walton and Krabbe [52], for example, introduce the games PPD_0 [52, pp. 149–152], RPD_0 [52, pp. 158–161], and CPD [52, pp. 163–164] during an investigation of the interactions between parties during persuasion dialogues. Girle introduces a number of games which are aimed at modelling belief revision in A.I. systems [7–9]. McBurney and Parsons specify a number of games for use in communication between agents in multiagent systems (MAS) of which [31] is representative. Bench-Capon introduces the *Toulmin Dialogue Game* and its associated computational implementation which is particularly suited toward modelling legal argument [3]. More recently, Reed and Wells introduce a simple dialectical game for mediating debate between humans and agents in a semantic web based system to support exploration of complex and contentious subjects [45].

The attention paid to dialectical games in these domains is valuable, however that attention is usually confined to a particular game and its features rather than looking at the wider range of games and features available. For example,

* Corresponding author.
E-mail addresses: simon.wells@abdn.ac.uk (S. Wells), chris@computing.dundee.ac.uk (C.A. Reed).
URLs: http://www.simonwells.org (S. Wells), http://www.computing.dundee.ac.uk/staff/chris (C.A. Reed).

1570-8683/$ – see front matter
http://dx.doi.org/10.1016/j.jal.2012.09.001

investigations of dialectical games applied to MAS usually confine their attentions to Mackenzie's DC before proceeding to formulate new games. However DC, whilst both popular and influential, does not contain all or even most of the available features found in other games within the literature. Perhaps this is because DC incorporates a good basic feature set which form a good foundation from which to explore specific questions about argument and dialogical interaction. The sheer variety of games and their features, across the many domains in which they are investigated, is rich and deserving of attention. One way to achieve this is to analyse distinct games from these domains in terms of their unique features, and to collate those analyses. The benefit of this approach is that the results of the game analysis can subsequently serve as a foundation for the construction of more generic frameworks for representing and implementing new and existing games, and as a starting position from which to explore the space of possible dialectical games. A recent trend in dialectical games, at least in the MAS domain, has been towards creating generic frameworks for representing dialectical games in general, within which particular games are instantiated as required. To this end there have been generic frameworks proposed by Maudet and Evrard [29], McBurney and Parsons [32,31], Bench-Capon et al. [4], and Wells and Reed [54]. Whilst a valuable foundation for the construction of agent communication systems which maximise reuse, a consistent deficiency of this approach has been the lack of a principled exploration of the features that such frameworks should incorporate or the range of rules that should be supported. Without determining the range of rules and features required, at the very least, by extant games, and examining how they can be combined to formulate new games, generic frameworks run the risk of being *too* generic because the amount of effort required to implement new games in the framework is not reduced over that required to implement new games from scratch. Furthermore, generic frameworks which do not support the description of existing games are less flexible and reduce the space of potential games that can be formulated to tackle domain specific problems.

Whilst much exploitation of dialectical games has been found in agent communication, these games are beginning to be deployed in online argumentation systems to specify the communicative interactions available for both computer and human players. MAgtALO [45] is an online argumentation system which uses agents to represent the positions of domain experts. Argumentation is used to control the interaction between MAgtALO agents and users. These interactions are structured using dialogue games which guide the resulting dialogues towards constructive ends such as the elicitation of new knowledge and public participation in complex real-world issues. MAgtALO features *mixed initiative argumentation*, a mixture of human users and intelligent software agents, which have competing requirements. From the software agent perspective the interaction protocols used by MAgtALO should be computationally tractable and supportive of many of the identified desiderata in agent communication protocols [33]. However the system must also support the human users with protocol attributes such as comprehensibility and simplicity [41].

Another online argumentation system, InterLoc [41], is an educational tool that uses simple dialectical games to support students in exploring a knowledge domain with their peers. One of the games used by InterLoc, the Creative Thinking Game (CTG), guides students in their selection of responses to earlier dialogue events by suggesting permissible responses. The CTG is designed so that the suggested responses are suited to the context of the dialogue, but does not *mandate* the set of legal responses as is found in many of the games applied to agent communication. There is no reason to suggest that MAgtALO agents could not interact with students through the InterLoc interface if the interaction protocols were shared. Indeed it is this kind of cross-over between individual systems that is the core of the nascent World Wide Argumentation Web (WWAW) [39] which envisages sharing of arguments, and interaction, between otherwise separate online argumentation systems. In the WWAW, tools such as MAgtALO and InterLoc provide a mechanism for direct online interaction between users and the system, enabling real-world arguments to be captured and shared.

Other tools, such as ArgDF[1] [40] and Avicenna,[2] enable arguments to be created, manipulated and displayed online. These systems are all developed by separate groups and one aspect underpinning them is the use of the nascent Argument Interchange Format (AIF) [6] to enable structured arguments to be represented and shared between different tools. The AIF, however, only supports interchange of monologic argument, whereas many, of the processes that expose or create monologic arguments are inherently dialogical. This has been recognised in O'Keefe's distinction between Argument_1 and Argument_2 [36], Kamlah and Lorenzen's use of dialogical definitions of logical connectives and quantifiers [14], and most recently made explicit by Reed and Budzynska in [43] which explores how the interactions between dialogue moves affect the structure of the constructed argument. A method is therefore required for sharing both the results of argumentation processes, in the form of structured dialogues, and the protocols by which the dialogues were constructed. This would directly support interaction between user-facing tools, such as the MAgtALO and InterLoc interactions, and between user-facing tools and the WWAW infrastructural tools, as well as enabling existing non-WWAW agent systems to interact with the WWAW. The proposed AIF+ [46] is envisaged as an enhanced AIF which includes support for both dialogue and protocol specification thus underpinning the dialogical aspects of the WWAW, but development of this interchange format is in its infancy and requires input from an array of stakeholders involved in the development and exploitation of argumentation software.

It is proposed that a unified framework for representing, implementing and deploying both existing and new dialectical games is needed and that a strong foundation for such a framework can be constructed based upon an analysis of a representative range of extant dialectical games across the range of problem domains. The remainder of this paper proceeds

[1] http://www.argdf.org/.
[2] http://www.research.buid.ac.ae:8080/Avicenna/.

as follows: a representative range of dialectical games from a variety of problem domains are introduced, analysed, and the results collated; a domain specific language (DSL) and formal grammar for describing the features of these dialectical games in a cohesive way is then presented and explored and finally, the extant games are reformulated and described using the DSL.

2. Features of extant dialectical games

An analysis of a range of extant dialectical games was performed with the aim of identifying those common features that are shared between disparate games. The underlying notion is that disparate representational styles can be reconciled if the common features are identified. As a result a single representational schema can then be used to describe a wide range of games and to facilitate interchange of games between various systems. The games that were surveyed include those from the philosophical and agent literature including, Hamblin's H [10], Mackenzie's DC [21], Walton's CB [51], Walton and Krabbe's PPD_0 [52], Bench-Capon's Toulmin Dialogue Game [4], as well as those used in online argumentation systems such as MAgtALO Knowledge Elicitation Game [45] and the Creative Thinking Game of InterLoc [41]. Many dialectical games have been developed in recent years and an understanding of the breadth of study of these games can be gained from the family tree shown in Fig. 1. As well as the games explored in this paper, there are a range of games due to Carlson [5], Girle [7–9], Lodder and Herczog [18], and Lodder [17] amongst many others. The overall aim is to account for the features of all available dialectical games but in the current research a representative sample of games have been investigated selected because they incorporate features that were novel when they were introduced and could not be found in contemporary games at the time. These features and components have to be recognised and explored so that they can inform the development of an interchange format for dialectical games. The remainder of this section introduces these games before presenting a summary of features and components that should underpin an interchange format, and which will inform the discussion regarding the capabilities of the AIF+.

2.1. Hamblin's 'H'

Having established the need for formal dialectic games as a tool for determining how various phenomena occur in dialogue [10, pp. 253–255], Hamblin proposes a general framework for formal dialectical games and a specific game that we shall refer to as 'H' [10, pp. 255–282]. Hamblin describes a number of abstract components and attributes which can be used to describe dialectical games. These components and attributes include commitment stores, rules and languages. H is a system of simple rules framed in terms of various components such as commitment stores, rules and language. To specify the gross structure of a dialectic system, Hamblin defines three sets of rules; locutional rules, syntactical rules, and commitment-store operations. Locutional rules specify the range of locutions that are permitted by the system. Syntactical rules are those rules which set out structural aspects of the system and which don't depend on commitment-store operations. The rules in the set of commitment-store operations define the operations that must be performed on the players commitment stores as a result of the utterance of particular locutions. It is the commitment of participants to various positions during a dialogue that was novel to H and which has subsequently been adopted by the majority of dialectical game developers. Hamblin [11] explains that speakers become committed to a statement when they either personally utter the statement or agree to the statement when it is uttered by another dialogue participant. Commitments are distinguished from beliefs because they are a function of the locutional events that have occurred during the dialogue although Hamblin allows that the resultant collection of commitments may be considered to represent a *persona* of beliefs. A speaker does not always believe everything they say but their saying it still commits them whether they believe it or not. Retraction of commitments is also distinguished from denying the statement associated with the commitment because retraction removes the commitment from the retracting player's store whilst a denial entails incurring a contradictory commitment. Commitment stores are introduced as a means for the players to keep a tally of any commitments that they have incurred during a dialogue. The players each maintain their own commitment store to which commitments are added or removed dependent upon the events that occur in the dialogue. Commitment stores are also publicly inspectable so that all players can inspect each others stores at any point. The commitment store is a flexible mechanism for specifying certain requirements over the gross structure of the system. For example, if it is required that a player must be consistent in their commitments then the rules of the system can be formulated to ensure that each new commitment incurred can be added without inconsistency to that player's commitment store. The commitment store is the main distinguishing feature of the Hamblin-type formal dialectic system and is the most common factor used to determine whether a given locutional act is permissible at any given point in the dialogue. A complete précis of H, along with all the other games presented here, is available in the extended technical report which accompanies this paper [56].

2.2. Mackenzie's 'DC'

Mackenzie's system DC [21] responds to the Woods and Walton exploration of *petitio principii* [58] in which cumulativity is identified as the defining factor of the *petitio* fallacy in a dialectical context. This conversation was to continue through Woods and Walton's response to DC [59] and Mackenzie's counter [24]. DC contributes a number of new component states, requirements and effects that can be used to specify a dialectical game. The commitment effect of some moves such as the

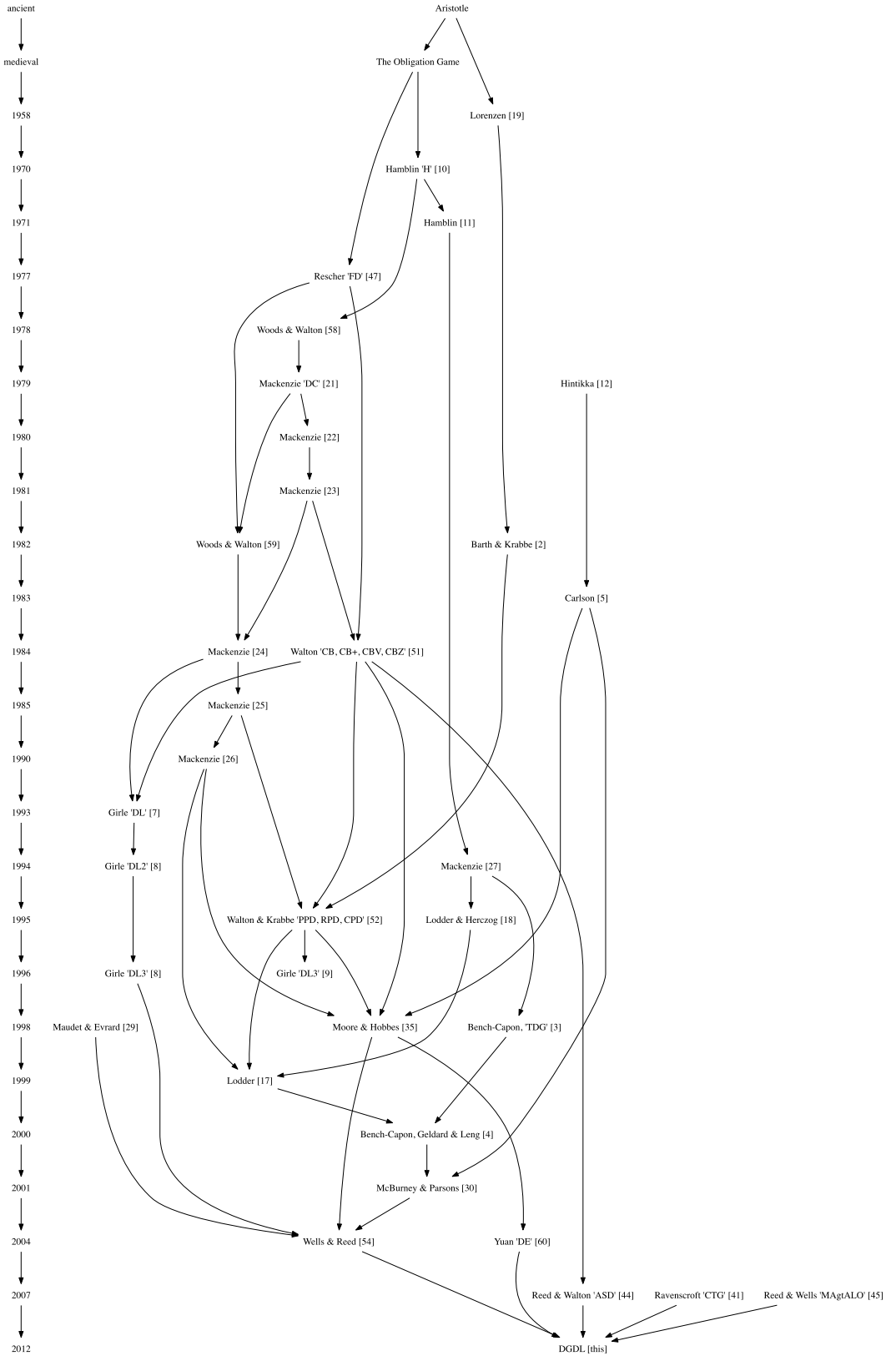

Fig. 1. A family tree of dialectical game research.

"Statements" move is expressed in terms of earlier moves that have taken place in the immediate previous turn. Specifically, the commitment effect of the "Statements" move is dependent upon the move played in the last turn. DC introduces commitment with respect to a particular move rather than commitment just to the propositional content of the move. Thus the range of conditions used in DC to decide whether a move is legal is greater than when compared with H. The legality of a move can depend upon the logical relationships between relevant commitments in the players commitment stores. For example, one of the requirements of the "Resolve" move is that the content of the move is a conjunction of inconsistent statements from the hearer's commitment store. DC has inspired a number of dialectical games, either through amendments to the original DC rules or through adoption of the DC locutions. For example, Mackenzie explores a variety of games related to DC in [22–25], and [26]. Other researchers, including Moore and Hobbes [35], and Yuan, who developed DE [60], have researched games based on amended versions of DC. Yuan proposes DE formulated for use in human–computer debate. Two types of commitment are identified, the assertion and the concession, which differ based upon whether the speaker has explicitly incurred the commitment through asserting some position, or whether the speaker has merely accepted something said by their opponent "for the sake of argument". A commitment is classified depending upon the way in which the commitment is incurred when a move is played. The conditions under which a move is made set out which type of commitment is incurred as an effect of the move. Other than the incorporation of different commitment-store models, DE differs little from DC in terms of the range of components and artifacts which the game's rules manipulate.

2.3. Walton's 'CB'

Walton introduces a number of games in [51] beginning with the game CB which is a simple sub-game of both H and DC that incorporates win/loss notions inspired by Rescher's game of disputation [47]. Walton proposes CB as a minimal system for studying strategic play in dialectical games. This is examined through a win-strategy, a dialectical path though the dialogue terminating at a player's thesis. CB is the initial game in a group of games constructed from CB through inclusion of a number of additional rules. These systems are focused on the process of articulating and clarifying the players commitments through the question–answer structures found in argumentative dialogue. Walton arranges the systems in terms of the strength or strictness with which commitments are handled, consequently CB is identified as the weakest system in the CB family of games whilst CBZ is identified as the strongest. The system CB+ [51] incorporates the rules of CB with the addition of the $(+)$ rule. This rule is formulated to enable the system to regulate inconsistency in the players commitments. The type of consistency that the $(+)$ rule regulates is that which Walton identifies as *ambivalence* and represents "a kind of insincerity or irregularity in playing the game". It occurs when a player incurs commitment to a statement but later retracts that commitment when it becomes apparent that maintaining the commitment may prove to be a liability. What is interesting about this system is that instead of prohibiting undesirable behaviour outright, the rules specify a sanction that is imposed if the player exhibits the undesirable behaviour. In this case the undesirable behaviour is that of players indicating that they are not committed to a statement which follows from their existing commitments. Instead of being prohibited, the player is allowed to engage in this behaviour but at the risk of losing their current score of points which might result in losing the dialogue outright. The CBV system [51] introduces the *Veiled* or *dark-side commitment*. Each player possesses a dark-side commitment store which is private. The contents of the dark-side commitment store are not on public view unlike the conventional commitment store proposed by Hamblin. The rules of CBV set out the circumstances under which a commitment is moved from the dark-side to the light-side so that the contents of the dark-side store become known to the participants.

2.4. Walton and Krabbe's permissive persuasion

Walton and Krabbe introduce a framework for specifying persuasion dialogues alongside a number of specific dialectical games. These games capture notions of permissive persuasion and rigorous persuasion and the embedding of one in another in a process referred to as "tightening up". In [52] the game PPD_0 is introduced to model permissive persuasion type dialogues. Walton and Krabbe propose PPD_0 as just one example of a dialectic system to govern permissive persuasion dialogue and set out eighteen general attributes which can be used to characterise permissive persuasion dialogues. Their approach is to begin with a simple system to model such dialogues which can be extended through the inclusion, removal and replacement of rules to enable particular problems to be studied. In [52] Walton and Krabbe introduce a number of new features to formal dialectic research whilst simultaneously producing mature treatments of some existing concepts. Dialogue stability, embeddings and transitions are introduced as new feature of dialectical systems. Dark-side commitments, introduced in [51], and their associated stores are given a central role in the model of permissive persuasion captured in PPD_0. Commitment in general is analysed in depth and the ways of both incurring and retracting commitments are enumerated.

Walton and Krabbe state that how commitment should be incurred or retracted is dependent upon a number of general factors which include the kind of move made, the type of dialogue, the goal of the dialogue, the speaker's role, and the rules appropriate to the dialogue. They identify a partial taxonomy of dialogue types which include persuasion, negotiation, enquiry, deliberation, information-seeking, eristic and mixed. Each type of dialogue may have further subtypes and can be classified based upon the initial situation, main goal and the participants' individual goals. Furthermore, Walton and Krabbe identify that certain fallacies may arise as the result of improper shifts between sub-dialogues and suggest that dialectical

games support such shifts. Given two games which support two types of dialogue, permissive persuasion and rigorous persuasion, the complex dialogue game, CPD [52, pp. 163–164], is then introduced. In CPD players begin the complex dialogue in a permissive persuasion sub-dialogue, governed by the rules of PPD, which can shift to an embedded instance of an RPD sub-dialogue. Walton and Krabbe's games introduce an important new idea to the dialectical game approach, that of having multiple rule-sets governing different contexts of interaction and that as a dialogue develops it can move through various sub-dialogues.

2.5. The Toulmin Dialogue Game

The Toulmin Dialogue Game (TDG) [3] can be described as Rescherian in style [47] for two reasons, firstly because it has three participants: two players and a referee which is in contrast to the games in the Hamblin tradition. Secondly, the style of turn-taking in TDG is based upon a player making moves until a state of the game is reached at which the speaker role can be assigned to the other player, a feature that is in common with Rescher's approach. The participants work cooperatively towards a position in which a claim is agreed and the players' interactions facilitate the construction of a supporting argument structure for that claim. TDG is specified using a state transition diagram to illustrate the procession of moves and player roles, and a schema for each illocutionary act which defines the pre-, post- and completion-conditions for each move. TDG makes a number of novel contributions including a wider range of illocutionary acts and a new type of data store. The Toulmin argument schema [50] suggests a structure for arguments in which the component propositions of the argument can occupy various roles. Bench-Capon proposes a modified argument schema in which there is no qualifier, a presupposition role is added, and in which individual instances of the schema can be chained together such that the claim for one argument can fulfil the data role of another argument. This model of argument structure is explicitly supported by TDG through its wide range of illocutionary acts. TDG incorporates a claim stack, a shared data structure to which is added each new claim that the players make and whose contents are altered by the moves the players make during a dialogue. The presence and position of an individual claim within the claim stack is used to extend legality conditions for the moves of the game.

TDG makes a number of novel contributions to dialectical games. A tripartite condition structure, incorporating pre-conditions, post-conditions and completion-conditions, is used to specify individual TDG moves, similar to the way that the semantics of KQML performatives are specified [16]. This breaks away from the traditional specification model involving locutional, commitment, and syntactical rules. In TDG the rules are grouped according to the move to which the rule applies. This approach gives TDG a clean and well structured presentation in which the legality requirements of each move and the effects of playing the move can be easily comprehended and understood. Unfortunately the simultaneous use of a state transition diagram, specifying the progression of moves, and the move list, setting out legality conditions, means that their are two sources of information about available moves, both of which must be consulted to determine the set of legal responses. The diagram specifies the progression of possible responses moves but not the subset of legal responses which is determined through inspection of the pre-conditions for each move. Rather than building on a simple alternating turn structure, in TDG a player can make a move if they have *control* of the dialogue. Control of the dialogue, and thus permission to play a move, is passed to a given participant dependent upon the move that has just occurred and the roles that the players occupy. Some players will thus have the opportunity to play many more moves than others, and the referee, for example, will often play relatively few moves.

2.6. Summary of features

As a result of analysing the games a range of attributes were identified that a common game description format must support. These are summarised in the table shown in Fig. 2. It is the particular combination of these attributes which yield a distinct dialectical game. It is important therefore that any language for describing these games must be sufficiently flexible to allow these attributes to be expressed.

3. The Architecture for Argumentation (A4A)

The Architecture for Argumentation (A4A) is a theoretical and applied framework, for rapidly specifying, implementing, and exploring computational dialectical games [53]. The A4A Dialogue Game Description Language (DGDL) was developed to balance between the needs of developers, who are seeking to build tools to work with dialectical games, and the needs of researchers, who have in depth knowledge of the domain of dialectical games but not necessarily programming experience and the needs of end-users, who must understand the resultant games.

DGDL is a Domain Specific Language (DSL), a small language developed to be both concise and powerful. This is achieved by designing such a language to fit a particular knowledge domain. This enables the developer to focus on the key abstractions of the problem space. As a result DSLs are becoming an increasingly popular way to balance the needs of programmers against those of domain experts as evidenced by popular examples such as the dot DSL for specifying graphs,[3] the CSound

[3] http://www.graphviz.org/.

Moves per turns	Either a single, or multiple, or a defined maximum number of moves per turn
Turn Organisation	Strict progression of alternating moves or liberal progression
Dialogue Magnitude	The maximum number of turns that the dialogue allows
Move Types	Range of available locutions or performative types
Move Content	The statements, propositions, variables, tokens, or collections thereof that are moved
Openers	A description of the locutional form of the move, e.g. "Is it the case that…?"
Stores	Collections of in-game artifacts, organised as sets, stacks, or queues
Store Contents	In addition to move content, stores can contain locutions or arguments
Store Visibility	Whether a given store is public or private
Move Legality	The formulation of conditions that must be satisfied for the move to be legal: Inspection of previous moves; store contents; role occupation; store magnitude; store comparisons; the length of the dialogue; correspondence to a given scheme; relation between content elements; form in which the content is presented
Move Effects	The effect of a successfully played move upon the dialogue & its components: Prescription of mandated responses; operations on stores; update to status of a game or system; assignment of role
Participants	The number & identity of players that the game supports
Roles	The roles that players can occupy at various stages of a game
Rules	Non-move specific rules that can alter the game as a function of game state rather than as a result of the particular move that has been played

Fig. 2. Attributes of extant games.

language for creating audio files,[4] and the increasing visibility of languages such as Ruby, where the support for DSL development is integral to exploitation of the language [49].

A number of goals can be identified for DGDL. Firstly, to enable existing dialogue games to be represented in a unified, machine-readable format so that common dialogue game run-times can be developed which support multiple games. Secondly, to simplify the development of new dialogue games, either as wholly new specifications of rules or through recombination of well understood and verified rules and elements from existing games. Thirdly, to support the sharing and interchange of dialogue and interaction protocols between a diversity of argumentation software platforms. Fourthly, as the number of available game protocols increases, it is important to support effective comparison and evaluation of games to enable users to make an informed decision about which protocol to adopt in their problem domain.

3.1. A4A Dialogue Game Description Language

DGDL is underpinned by an Extended Backus–Naur Form (EBNF) [57] grammar, presented in Fig. 3, which enables syntactically correct and verifiable dialectical game descriptions to be formulated. For non-terminal symbols x and y, $[x]?$ denotes optionality so that x can appear zero or one times; $[x]^*$ denotes that x may appear zero or more times; $[x]+$ denotes that x may appear one or more times; $x \mid y$ denotes either x or y. DGDL uses || as an in-fix terminal symbol to indicate disjunction whilst '!' is used as a pre-fix terminal symbol to indicate negation of the symbol with which it is paired.

DGDL supports the description of either a single individual dialectical game, or a collection of dialectical games which form a single system, with appropriate rules to facilitate shifts between them. Each game is formed from a mandatory description of its composition, which describes attributes of the game such as turns, players, and stores. The composition is then followed by a mandatory set of interactions, and an optional set of rules.

Turns are used to control which player or players can make a move at any given point. Only players who occupy the speaker role may take a turn and the manner in which the speaker role is assigned depends upon both the permissible magnitude of each turn and the turn ordering. Turns may consist of a single move, a defined set of moves, or an arbitrary set of moves in which the player makes as many moves as they decide to make in that turn. Once one player has played her turn, then another player can take his. The order in which turns progress can be either strict of liberal. A strict ordering means that after each turn is played the speaker role is assigned to the next player in the player list and if the end of the player list has been reached then it becomes the turn of the first player once more, and so on. A liberal ordering means that after each turn is played, the next player to move is determined based upon the particular moves that have been played, for example, a player might be able to continue playing moves until they play a particular move that causes the speaker role to be reassigned.

Players are participants of the game. The minimum and maximum numbers of participants must be identified, and for each of these an identifier and initial role list must be specified. There are a number of roles that can be identified in extant games, including the speaker and listener, initiator and respondent, winner and loser, and proponent and opponent roles. Roles are used to refer to players in different ways through the rules of the game. For example, given two players, one identified as black in the proponent role, and the other identified as white in the opponent role, referring to the proponent enables a rule to be applied with respect to the black player. If the players subsequently swap roles, so that the proponent becomes the opponent, and vice versa, then the same rule formulation can be applied but now affects the white player.

Stores are collections of game artifacts that are used to keep track of the game. Each store is identified by a name and ownership. A single store can be owned by any individual player, a group of players, or shared by all players. Furthermore,

[4] http://www.csounds.com/.

```
System          ::= SystemID { [Game]+ } | Game
SystemID        ::= Identifier
Game            ::= GameID { Composition [Rule]* [Interaction]+ }
GameID          ::= Identifier
Composition     ::= Turns [RoleList]? Participants, [Player]+ [Store]*
Turns           ::= {turns,  TurnSize, Ordering, [MaxTurns]? };
TurnSize        ::= magnitude: Number | single | multiple
Ordering        ::= ordering: strict | liberal
MaxTurns        ::= max: Number | RunTimeVar
RunTimeVar      ::= $Identifier$
RoleList        ::= {roles, { Role [, Role]+ } };
Role            ::= speaker | listener | Identifier
Participants    ::= {players, min: Number, max: Number| undefined };
Player          ::= {player, id: PlayerID|RunTimeVar[, roles:{ Roles }]? };
PlayerID        ::= Identifier
Store           ::= {store, id: StoreName, owner=StoreOwner, StoreStructure, Visibility};
StoreName       ::= Identifier
StoreOwner      ::= PlayerID | { PlayerID[, PlayerID] } | shared
StoreStructure::= structure:set|queue|stack
Visibility      ::= visibility:public|private
Rule            ::= {RuleID, scope:initial|turnwise|movewise, RuleBody};
RuleID          ::= Identifier
RuleBody        ::= Effects | Conditional [& Conditional]*
Effects         ::= { Effect [& Effect]* } | { Effect [|| Effect]* } | { Effects [&||| Effects]* }
Effect          ::= EffectID( Parameter [, Parameter]* )
Parameter       ::= Identifier|Number|Commitment|SystemID|GameID|PlayerID|MaxTurns
                    |StoreName|StoreOwner|Requirements|Role|RunTimeVar|Condition|Effect
Interaction     ::= { MoveID, Content [,Opener]?, RuleBody };
MoveID          ::= Identifier
Content         ::= { ContentSet|ContentVar[,ContentSet|ContentVar]*}
ContentSet      ::= UpperChar
ContentVar      ::= LowerChar | !LowerChar
Opener          ::= String
Conditional     ::= {if Requirements then Effects [elseif Requirements then Effects]* [else Effects]?}
Requirements    ::= {Condition [& Condition]*} | {Requirements[ || Requirements]*}
Condition       ::= ConditionID( Parameter [, Parameter]* )
ConditionID     ::= Identifier
Commitment      ::= Content | Locution | Argument
Locution        ::= < MoveID, Content>
Argument        ::= < Conclusion, Premises >
Premises        ::= {ContentVar[, ContentVar]*}
Conclusion      ::= ContentVar
SchemeID        ::= Identifier
Identifier      ::= UpperChar [ UpperChar | LowerChar | Number ]+
String          ::= "[UpperChar|LowerChar|Number|Symbol]+"
Number          ::= [0-9]+
UpperChar       ::= [A-Z]+
LowerChar       ::= [a-z]+
Symbol          ::= ` '|`?'|`,'|`.'
```

Fig. 3. EBNF grammar for the A4A DGDL.

the way that artifacts are arranged within a store can differ, depending upon whether a set, a stack, or a queue is required. This enables an arbitrary collection of artifacts to be stored, such as the commitments incurred during a dialogue, or a stack of artifacts, such as the claims raised, or a queue of artifacts, such as the issues that the dialogue must address. Finally, the visibility of artifacts within a store can be specified as a public or private.

Interactions are the moves that players can make and take the form of locutions that the players utter during a dialogue. The interactions cause the game components to be altered, such as commitment stores to be updated or restrictions on the set of legal responses to be imposed, and represent the only method within the rules of a game that a player can directly interact with the game components. Interactions are formed from a move ID, specification of content and opener, and a body.

Rules are optional and set out effects triggered by reaching a particular game state, and are different to interactions because their requirements are checked at regular discrete time-points, such as after the completion of each turn (*turnwise*) or after every single move (*movewise*) where multiple moves are made per turn, rather than directly as a result of a specific move being played. The advantage of this latter approach is that the rule-set can be made more compact so that conditions which must be checked after every single move, and would therefore require to be added to the formulation of each move and thus increase repetition, can be grouped into a single place. Similar to interactions, rules are formed from a rule ID and specification for the scope with which the rule should be checked, and a body.

The bodies of both interactions and rules are formed in a similar fashion, and consist of either a plain set of effects, or a conditional statement setting out the conditions under which various sets of effects can occur. Conditions and effects are both represented in the grammar as predicate statements having either a condition ID or effect ID followed by a comma-separated list of elements. The advantages of this approach are that the specific range of conditions and effects

```
Event             ::= event( last|!last|past|!past, MoveID [,Content]? [, PlayerID|Role]?
                      [, Requirements]? )
StoreInspection   ::= inspect( in|!in|on|!on|top|!top, Commitment, StoreName, [PlayerID|Role]?
                      [, initial|past|current]? )
RoleInspection    ::= inrole( PlayerID, Role )
Magnitude         ::= size( StoreName|LegalMoves, PlayerID, empty|!empty|Number )
StoreComparison   ::= magnitude( StoreName, PlayerID|Role, greater|smaller|equal|!equal,
                      StoreName, PlayerID|Role, )
DialogueSize      ::= numturns( SystemName, Number)
Correspondence    ::= corresponds( Argument, SchemeID)
Relation          ::= relation(Content|Argument, backing|warrant, Content|Argument )
CurrentPlayer     ::= player(PlayerID|Role)
ExternalCondition::= extCondition(Identifier [, Identifier]*)
```

Fig. 4. Condition predicates used in DGDL.

```
Move           ::= move(permit|mandate,next|!next|future|
                   !future, MoveID, [, Content]? [, PlayerID|Role]? )
StoreOp        ::= store( add|remove, Commitment, StoreName, PlayerID|Role )
StatusUpdate   ::= status( active|inactive|complete|incomplete| initiate|terminate, SystemID|GameID )
RoleAssignment::= assign( PlayerID|Role, Role )
ExternalEffect::= extEffect(Identifier [, Identifier]*)
```

Fig. 5. Effects predicates used in DGDL.

are formulated externally and are therefore decoupled from the basic grammar. To facilitate this requires only that implementations of DGDL specify the range of predicates that they will support and that range of predicates conform to either the Condition or Effect symbols of DGDL grammar. Support for any specific range of conditions and effects is thus moved from the grammar into any subsequent implementation of the grammar. This means that as new conditions and effects are required for new games they can be incorporated into DGDL without alteration to the grammar.

3.2. Enumeration of conditions and effects

The initial sets of condition and effect predicates are set out in Figs. 4 and 5, respectively. The conditions support checking earlier moves, inspecting the contents of specified stores, inspecting which players are currently in which roles, checking the size of a given store, comparing stores, checking the number of turns that have occurred since the dialogue commenced, checking whether an argument corresponds to a given scheme, checking whether an item of content or an argument occupies a specific relationship to another item of content or argument, and whether a player occupies a given role. The effects that can be applied to a game include the specification of mandatory or permissible responses, updates to artifact stores, updates to the status of a specific system or game, assignment of a role to a player, and the swapping of roles between players. The ExternalCondition and ExternalEffect predicates are designed to enable DGDL to make "calls" into external run-times or to evaluate functions outside of DGDL. This enables DGDL to be focused exclusively on dialectical game specification without requiring the DGDL to account for aspects of argumentative communication that are more peripheral to the interaction protocol aspects of DGDL.

4. Application of DGDL

DGDL was developed as a description language for dialectical games and has been applied in two modes thus far. Firstly, a range of extant dialectical games have been reformulated into DGDL from their original natural language specifications. These include those games whose analysis underpinned the development of DGDL, such as H, DC, and PPD_0, as well as other games that were not analysed when DGDL was developed, such as ASD [44], CTG [41] and the MAgtALO game [45]. Secondly, a new game has been developed which is expressed entirely in DGDL and is used to explore the dialectical shift from persuasion to negotiation during the fallacy of bargaining [55]. These modes of application demonstrate that DGDL is sufficiently expressive to describe many existing games, including those on which it is based, but importantly, also games that did not contribute to its development. Additionally, the development of a new game within DGDL demonstrates that DGDL is not limited merely to describing games that have already been developed. It should be noted that only one reformulation is presented here for each game "family" because many games are related and share a similar core of rules. For example, the CB, CBV, and CBZ games are very similar and would consume space for little gain if presented here. To remedy this, a game library has been created and stored online[5] which contains a wider selection of game descriptions than those presented here.

[5] https://github.com/siwells/DGDL.

4.1. Reformulation of Hamblin's game "H"

The reformulation of H into H_{DGDL} includes a specification for the structure of the game which describes the way that turns are structured, the players, denoted black and white, and the commitment stores used in the game. Following the specification of the game's structure there is the specification of a set of seven moves which enable the players to interact during the game. It is in the reformulation of the moves that H_{DGDL} differs from H which allows only five moves. The extra two moves in H_{DGDL} are the Statements move, which enters a commitment to an argument into the player's commitment stores, whereas the basic Statement move enters only a commitment to the statement into the player's commitment stores. In H both types of commitment update are handled by a single Statement move which serves to complicate the formulation of the Statement move and makes it less clear what effect playing the move will have on the dialogue as a result. The other deviation of H_{DGDL} from H is in the NoCommitmentPlusChallenge move which merely concatenates the rule bodies from each of the individual NoCommitment and Challenge moves to formulate a new move. This is done to accommodate the single exception to the one move per turn structure that H includes which allows a NoCommitment move to accompany a Challenge move during a single turn. In the original formulation it is not clear that this partnering of moves refers to the same locutional content in each move rather than two separate and different instances of the NoCommitment and Challenge moves. In formulating H, Hamblin's intention was to enable a player to state that they are not committed to a given position and then in the same turn to seek the grounds for their opponents commitment with respect to the same position. Interpretations of the rules of H can easily be described using DGDL in way that makes the differences crystal clear.

```
H{
  {turns, magnitude:single, ordering:strict };
  {players, min:2, max:2 };
  {player, id:black, role:speaker };
  {player, id:white, role:listener };
  {store, id:CS, owner:black, structure:set, visibility:public } ;
  {store, id:CS, owner:white, structure:set, visibility:public } ;

  {Statement, {p},
    { store(add, {p}, CS, speaker) & store(add, {p}, CS, listener) } };
  {Statements, {p,q},
    { if { event, last, Challenge, {p}) } then { store(add, {q}, CS, speaker)
    & store(add, <p,{q}>, CS, speaker) & store(add, {q}, CS, listener)
    & store(add, <p,{q}>, CS, listener) } } };
  {NoCommitment, {p},
    { store(remove, {p}, CS, speaker) } };
  {Question, {p},
    { store(add, {p}, CS, speaker) & { { move(mandate, next, statement, {p}) }
    || { move(mandate, next, statement, {!p}) } } } };
  {Challenge, {p},
    { store(add, {p}, CS, listener) & { { move(mandate, next, statement, {!p}) }
    || { move(mandate, next, NoCommitment, {p}) } || { move(mandate, next, statements, {p,q}) } } } };
  {Resolve, {p},
    { { move(mandate, next, statement, {!p}) } || { move(mandate, next, NoCommitment, {p}) } } };
  {NoCommitmentPlusChallenge, {p},
    { store(remove, {p}, CS, speaker) & store(add, {p}, CS, listener)
    & { { move(mandate, next, statement, {p}) } || { move(mandate, next, NoCommitment, {!p}) }
    || { move(mandate, next, statements, {p,q}) } } } };
}
```

Compared to the original rule description, H_{DGDL} is concise and easily comprehensible. Notice that for each move available to the players, it is clear both when that move can be played and what the effect of playing that move is.

4.2. Reformulation of Mackenzie's "DC"

No examination of dialectical games would be complete without inclusion of Mackenzie's DC which has proven to be the most popular dialectical game with many implementations of the original game [21] and a number of variations on the original rule-set [60] and [35]. DC, like H, conflates in natural language moves that are semantically distinct. The rule that was applied in these cases when reformulating games was to look at the legality conditions and effects of the move and split the move into two or more differently named moves if there were two sets of legality conditions, each of which leads to a different effect. This disambiguation of more complex move formulations serves to simplify and clarify the overall game description.

```
DC{
  {turns, magnitude:single, ordering:strict };
  {players, min:2, max:2 };
  {player, id:init };
  {player, id:resp };
  {store, id:CS, owner:init, structure:set, visibility:public };
  {store, id:CS, owner:resp, structure:set, visibility:public };

  {Initial, scope:initial, { size(CS, init, empty) & size(CS, resp, empty) & assign(init, speaker)
    & assign(resp, listener) } };

  {Statement, {p},
    { if { inspect(!in, {p}, CS, speaker) & inspect(in, {p}, CS, listener) }
    then { store(add, {p}, CS, speaker) & store(add, {p}, CS, listener) } } };
  {Defence, {p},
```

```
    { if { event(last, Challenge, {q}) & inspect(!in, <Challenge,{p}>, CS, Listener) }
    then { store(add, {p}, CS, speaker) & store(add, <p,{q}>, CS, speaker) &
    store(add, {p}, CS, listener) & store(add, <p,{q}>, CS, listener) } } };
  {Withdraw, {p},
    { store(remove, {p}, CS, speaker) } };
  {Resolve, {p},
    { { move(mandate, next, Statement, {p}) } || { move(mandate, next, Statement, {!p}) }
    || { move(mandate, next, NoCommitment, {p}) } } };
  {Challenge, {p},
    { store(remove, {p}, CS, speaker) & store(add, {p}, CS, listener)
    & store(add, <Challenge,{p}>, CS, speaker)& { { move(mandate, next, NoCommitment, {p}) }
    || { move(mandate, next, Resolve, {p}) } || { move(mandate, next, Defence, {q}) } } } };
  {Resolve, {p},
    { if { event(last, Challenge, {p}) } then { { move(mandate, next, NoCommitment, {p}) }
    || { move(mandate, next, Statement, {p}) } } } };
}
```

One notable aspect of both “H” and “DC” is the lack of formulation for conditions under which a dialogue should terminate. This is an artifact of the reasons for which the games were originally developed, to illuminate particular sequences of dialogue that lead to the occurrence of particular logical fallacies. It is reasonable to assume that neither game was developed to be played as a game with the aim of one or another player winning, hence the lack of either victory or termination conditions. For this reason, within certain contexts of use, such as formal disputations in pedagogic settings or computational applications as communication protocols between intelligent software agents, neither game should be considered for actual play unless rules are added to enable, firstly, for the dialogue to come to an end, and secondly, for the determination of a victor to be made. In less formal contexts, such as modelling real-world dialogues, dialogue games should be considered within Wittgenstein’s concept of a game which need have no winner. It should be noted that neither termination nor victory conditions are necessary aspects for a game to be considered fully formed, playable, or even enjoyable, but the specification of such conditions can make the game very useful in computational contexts. The specification of such rules is a matter for the game designer to decide upon taking into consideration how they conceive that the game will be used.

4.3. Reformulation of Walton’s CB

CB is an interesting game for two reasons: the upper bound (n) on the maximum number of turns allowed during a dialogue, and the use of points to determine a winner. The upper bound on the number of turns means that there is a hard limit to how long a dialogue can run on, an attractive feature for a game to possess if it were to be deployed as an interaction protocol for autonomous agents. However this is at the expense of realism in the game from the perspective of real-world argumentative dialogue. This is because, although time-bounded, real-world dialogues usually run until many, if not all, of the arguments have been examined and either the participants have agreed to disagree or one of them has capitulated with respect to the position of the other. Real-world dialogues, excepting stylised debates, are not commonly bounded by the number of turns that the locutors might take, regardless of the time it takes to do so. As a group of games, the CB*n* family of games incorporate the most highly developed formulations of victory conditions. This is because the CB*n* games were developed to investigate strategy in dialogue games. Investigation of strategic play presupposes a way to determine the effectiveness of a given strategy and one way to determine the effectiveness of a dialogue game strategy is in terms of who won and who lost.

```
CB{
  {turns, magnitude:single, ordering:strict, max:$UserDefined$ };
  {roles, {winner} };
  {players, min:2, max:2 };
  {player, id:black };
  {player, id:white };
  {store, id:CS, owner:black, structure:set, visibility:public } ;
  {store, id:CS, owner:white, structure:set, visibility:public } ;

  {SpeakerWins1, scope:turnwise,
    if { numturns(CB, max) & magnitude(CS, speaker, greater, CS, listener) }
    then { status(terminate, CB) & assign(speaker, winner) } };
  {ListenerWins1, scope:turnwise,
    if { numturns(CB, max) & magnitude(CS, listener, greater, CS, speaker) }
    then { status(terminate, CB) & assign(listener, winner) } };
  {SpeakerWins2, scope:turnwise,
    if { inspect(in, {p}, CS, Listener, initial) & inspect(!in, {p}, CS, Listener, current) }
    then { status(terminate, CB) & assign(speaker, winner) } };
  {ListenerWins2, scope:turnwise,
    if { inspect(in, {p}, CS, Speaker, initial) & inspect(!in, {p}, CS, Speaker, current) }
    then { status(terminate, CB) & assign(listener, winner) } };

  {statement, {p},
    { if { event(last, challenge, q) & corresponds(<q,P>, ImmediateConsequence) }
    then { store(add, {p}, CS, listener) & store(add, {p}, CS, speaker) }
    else { store(add, {p}, CS, speaker) } } };
  {withdraw, {p},
    { store(remove, {p}, CS, speaker) } };
  {question, {p},
    { { move(mandate, next, statement, {p}) } || { move(mandate, next, statement, {!p}) }
    || { move(mandate, next, withdraw, {p}) } } };
  {challenge, {p},
    { store(add, {p}, CS, listener) & { { move(mandate, next, withdraw, {p}) }
    || { move(mandate, next, statement, {q}) & corresponds(<p,Q>, Consequence) } } } };
  {WithdrawPlusChallenge, {p},
```

```
    { store(remove, {p}, CS, speaker) & store(add, {p}, CS, listener)
    & { { move(mandate, next, withdraw, {p}) }
    || { move(mandate, next, statement, {q}) & corresponds(<p,Q>, Consequence) } } } };
}
```

There are also a number of aspects of CB that are unclear in the original formulation of rules: the use of immediate consequence schemes; and how a player's thesis is established. Walton's CB*n* family of games make use of Mackenzie's *immediate consequence* which provides a schema for recognising and classifying certain inferences if they occur during a game, and thus enabling different paths through the dialogue to occur as a result. In [20], Mackenzie suggests a range of immediate consequence schemes but leaves the decision of which schemes to apply during any given instance of a game up to the players to determine. Walton does not elaborate on the range of immediate consequence schemes available to the player, nor are they specified as a part of the rules of CB. Hence a faithful reformulation of either CB or DC does not include specification of the range of schemes available to the game, nor is any suggestion made of how conformance to a given immediate consequence scheme can be determined by a game engine that processes the game. Instead, the rules merely state where it is important to verify whether an immediate consequence scheme has been fulfilled. CB does not include rules which specify how a player's thesis is established. In this case the reformulation deems the initial commitment of each player to be their thesis and a victory condition is formulated as a result which specifies that if a players initial commitment is removed from their commitment store then that player loses because they have retracted their thesis.

4.4. Reformulation of Walton and Krabbe's PPD$_0$

A novelty of PPD$_0$ is that it uses strict turn ordering but the players are allowed to make multiple moves per turn where the number of moves that can be made is decided by the player. To achieve this flexibility in DGDL reformulation of PPD$_0$ players indicate that their turn is complete by playing the EndTurn move, which in the PPD$_0$ DGDL reformulation is additional to the original rules, which has no pre-conditions and whose sole effect is to reassign the speaker and listener roles at the end of a turn. PPD$_0$ elaborates on the use of commitment stores in dialectical games by incorporating the dark-side commitments introduced by Walton in the CBV game [51] and introducing a mechanism within the light-side commitment store for dealing with the differences in strengths of commitment that can arise during a dialogue. Walton and Krabbe recognise that there is a difference in a player's positions as reflected by the state of their commitment store depending upon whether the player has asserted something to which they are committed, or whether they have merely conceded something which their opponent has asserted. This difference is made concrete in PPD$_0$ through two acts: firstly, by partitioning each players commitment store into two overlapping sets, the set of assertions and the set of concessions; and secondly, by formulating rules that manipulate the assertions and concessions separately.

```
PPD0{
  {turns, magnitude:multiple, ordering:strict };
  { roles, {Speaker, Listener} };
  {players, min:2, max:2 };
  {player, id:black, roles:{ Speaker } };
  {player, id:white, roles:{ Listener } };
  {store, id:Assertions, owner:black, structure:set, visibility:public };
  {store, id:Assertions, owner:white, structure:set, visibility:public };
  {store, id:Concessions, owner:black, structure:set, visibility:public };
  {store, id:Concessions, owner:white, structure:set, visibility:public };
  {store, id:Dark, owner:black, structure:set, visibility:private };
  {store, id:Dark, owner:white, structure:set, visibility:private };

  {Commencement, scope:initial,
    { move(mandate, next, Assertion, Speaker) } };
  {SpeakerWins, scope:turnwise,
    { if { inspect(in, {p}, Assertions, Listener, initial) & inspect(!in, {p}, Assertions, Speaker, current) }
    then { status(terminate, PPD0), assign(speaker, winner) } } } ;
  {ListenerWins, scope:turnwise,
    { if { inspect(in, {p}, Assertions, Speaker, initial) & inspect(!in, {p}, Assertions, Speaker, current) }
    then { status(terminate, PPD0) & assign(listener, winner) } } };

  {Assert, {p},
    { store(add, {p}, Assertions, Speaker) & store(add, {p}, Assertions, Listener) } };
  {Concede, {p},
    { if { { inspect(!in, {p}, Concessions, Speaker) & inspect(in, {p}, Assertions, Listener) }
    || { inspect(!in, {p}, Concessions, Speaker) & { { event(last, Request, {p}) }
    || { event(last, Serious, {p}) } } } } ||
    then {store(add, {p}, Concessions, Speaker) } } };
  {ElementaryArgument, {p, Q},
    { if { inspect(!in, {p}, Concessions, Listener) & event(past, Challenge, {p}, Listener) }
    then { store(add, {p}, Assertions, Speaker) & store(add, {p}, Concessions, Speaker)
    & store(add, {Q}, Assertions, Speaker) & store(add, {Q}, Concessions, Speaker)
    & store(add, <{p},Q>, Assertions, Speaker) & store(add, <{p},Q>, Concessions, Speaker) } } };
  {Request, {p},
    { if { inspect(!in, {p}, Concessions, Speaker) }
    then { { move(mandate, next, Concede, {p}) } || { move(mandate, next, nc, {p}) } } } };
  {Serious, {p},
    { if{ { inspect(in, {p}, Dark, Listener) } & { { event(last, nc, {p}) } || { event(last, Challenge, {p}) } } }
    then { move(mandate, next, Concede, {p}) }
    elseif{ { inspect(!in, {p}, Dark, Listener) } & { { event(last, nc, {p}) } || { event(last, Challenge, {p}) } } }
    then { { move(mandate, next, Concede, {p}) } || { move(mandate, next, nc, {p}) } } } };
  {Resolve, {p, q},
    { if { inspect(in, {p}, Concessions, Listener) & inspect(in, {q}, Concessions, Listener) }
    then { { move(mandate, next, nc, {p}) } || { move(mandate, next, nc, {q}) } } } };
  {Challenge, {p},
    { if { inspect(in, {p}, Assertions, Listener) & inspect(in, {p}, Concessions, Speaker)
```

```
    & event(!past, Challenge, {p}, Listener) } then { { move(mandate, next, ElementaryArgument, {p,Q}) }
    || { move(mandate, next, nc, {p}) } || { move(mandate, next, na, {p}) } } } };
  {nc, {p},
    { if { { { inspect(!in, {p}, Dark, Speaker) & event(!past, Serious, {p}, Listener) }
    & { { event(last, Request, {p}, Listener) } || { event(last, Serious, {p}, Listener) } } }
    || { inspect(!in, {p}, Dark, Speaker) & event(!past, Serious, {p}, Listener)
    & inspect(in, {p}, Concessions, Speaker) } }
    then { store(remove, {p}, Assertions, Speaker) & store(remove, {p}, Concessions, Speaker) } } };
  {na, {p},
    { store(remove, {p}, Assertions, Speaker) } };
  {EndTurn, {p},
    { assign(Speaker, Listener) & assign(Listener, Speaker) } } ;
}
```

In the reformulation of PPD_0 it becomes apparent that there are a number of ambiguities about how certain rules of PPD_0 should be interpreted and also that there is some under-specification that could lead to alternative correct interpretations of the rules. Whilst PPD_0 includes victory rules to determine which player wins there are no corresponding termination rules to determine when the dialogue is complete. In the reformulation, therefore, there is a turnwise rule which checks for and terminates the dialogue whenever a player retracts commitment to their initial thesis, defined in this case as the first commitment that the player makes. The winner and loser roles are then assigned as a result of this rule. A further point of under-specification in PPD_0 is found in the lack of rules for establishing the initial contents of the players dark-side commitment stores which are subsequently fixed throughout the dialogue. Whilst there are rules for checking whether a commitment is in a given dark-side commitment store there is no way within the rules of the game to establish those commitments to begin with.

4.5. Reformulation of Bench-Capon's TDG

TDG supports one of the most extensive range of locutions. Although TDG was designed to map naturally onto an executable specification, a state transition diagram is an integral part of the original specification and describes the available locutions at each turn of the dialogue. This means that the original specification cannot be executed directly without further work, for example, by translating the diagram into an executable format. An advantage of applying DGDL is that a complete and cohesive game can be described that no longer requires a diagram to understand the transitions between locutions which therefore simplifies the computational implementation of TDG.

```
TDG{
  {turns, magnitude:single, ordering:liberal };
  {players, min:3, max:3 };
  {player, id:init, roles:{ proponent, speaker } };
  {player, id:resp, roles:{ opponent, listener } };
  {player, id:referee, roles:{ listener } };
  {store, id:CS, owner:init, structure:set, visibility:public };
  {store, id:CS, owner:resp, structure:set, visibility:public };
  {store, id:Claims, owner:shared, structure:stack, visibility:public };

  {Initial, scope:initial, { move(mandate, next, Claim, speaker) } };

  {Claim, {p},
    { store(add, {p}, claims) & store(add, {p}, CS, proponent) &
    assign(proponent, listener) & assign(opponent, speaker) } };
  { Why, {p},
    { if { inrole(speaker, opponent) & inspect(in, {p}, Claims) } then
    { assign(proponent, speaker) & assign(opponent, listener) } &
    { { move(mandate, next, Withdraw, {p}, proponent) } || { move(mandate, next, SupplyData, {p}, proponent) } } } } };
  { OK, {p},
    { if { inrole(speaker, opponent) & inspect(in, p, Claims) } then
    { store(remove, {p}, Claims) & store(add, {p}, opponent) & store(remove, {!p}, opponent) &
    assign(referee, speaker) & assign(opponent, listener) } } } ;
  { So, {p},
    { if { inrole(speaker, opponent) & inspect(!in, <p,{q}>, CS, speaker) & inspect(in, {p}, Claims) }
    then { assign(proponent, speaker) & assign(opponent, listener) & move(mandate, next, SupplyWarrant) } } };
  { Presupposing, {p},
    { if { inrole(speaker, opponent) & inspect(in, <p,{q}>, Claims) }
    then { { assign(proponent, speaker) & assign(opponent, listener) } &
    { { move(mandate, next, SupplyPresupposition, {p}) } || { move(mandate, next, Withdraw, {p}) } } } } };
  { OnAccountOf, {p},
    { if { inrole(speaker, opponent) & inspect(in, <p,{q}>, Claims) &
    event(last, SupplyWarrant, {p}) & relation(<p,{q}>, {c}, warrant) }
    then { { assign(proponent, speaker) & assign(opponent, listener) } & { {move(mandate, next, Withdraw, p) }
    || { move(mandate, next, SupplyBacking, {s}) & relation(<p,{q}>, {c}, backing) } } } } };
  { SupplyData, {p},
    { if { inrole(speaker, proponent) & event(last, Why, {q}) & inspect(in, {q}, Claims) }
    then { { store(add, {p}, CS, proponent) & store(add, {p}, Claims) &
    assign(opponent, speaker) & assign(proponent, listener) } &
    { { move(mandate, next, SwitchFocus, {q}) } || { move(mandate, next, OK, {p}) }
    || { move(mandate, next, So, {p}) } || { move(mandate, next, OK, {p}) } } } } };
  { SupplyWarrant, {p},
    { if { inrole(speaker, proponent) & event(last, So, {q}) & inspect(in, {q}, Claims) }
    then { { store(add, {p}, CS, proponent) & store(add, {p}, Claims) &
    assign(opponent, speaker) & assign(proponent, listener) } &
    { { move(mandate, next, presupposing, {q}) } || { move(mandate, next, OnAccountOf, {r}) }
    || { move(mandate, next, OK, {p}) } } } } };
  { SupplyPresupposition, {p},
    { if { inrole(speaker, proponent) & inspect(in, <q,{r}>, claims) & inspect(in, {r}, Claims) }
    then { store(add, {p}, CS, proponent) & store(add, {p}, Claims) & assign(opponent, speaker)
    & assign(proponent, listener) } } };
  { SupplyBacking, {p},
    { if { inrole(speaker, proponent) & event(last, OnAccountOf, {p}) & event(past, SupplyWarrant, {p})
    & inspect(in, <p,{q}>, claims) } then { store(add, <p,{q}>, CS, opponent)
```

```
    & store(remove, <p,{q}>, Claims) & assign(referee, speaker) & assign(proponent, listener) } } };
  { Withdraw, {p},
    { if { inrole(speaker, proponent) & inspect(in, {p}, claims) } then{ store(remove, {p}, Claims)
    & store(remove, {p}, CS, proponent) & assign(referee, speaker) & assign(proponent, listener) } } };
  { SwitchFocus, {p},
    { if{ inrole(speaker, opponent) & inspect(!in, {p}, CS, speaker) & inspect(on, {p}, Claims)
    & inspect(!top, p, Claims) }then{ store(add, {p}, Claims) & assign(opponent, speaker)
    & assign(proponent, listener) } } };
  { CurrentClaim, {p},
    { if{ inspect(top, {p}, Claims) & inrole(speaker, referee) & inspect(in, {p}, CS, proponent) }
    then { assign(opponent, speaker) & assign(proponent, listener) } } };
  { End, {p},
    { if{ inrole(speaker, referee) & size(empty, Claims) } then{ status(terminate, TDG) } } };
  { Rebut, {p},
    { if{ inrole(speaker, referee) & inspect(top, {p}, Claims) & inspect(!in, {p}, CS, opponent)
    & inspect(!in, <q,{p}>, CS, opponent) }then{ assign(opponent, speaker) & assign(referee, listener) } } };
  { Rebuttal, {p},
    { if{ inrole(speaker, opponent) & event(last, Rebut, {q}) & inspect(!in, {q}, CS, speaker)
    & inspect(!in, {q}, CS, listener) } then { { store(add, {p}, Claims) & store(add, {q}, Claims)
    & store(add, {!q}, CS, speaker) & store(add, {p}, CS, speaker) & store(add, <!q,{p}>, CS, speaker)
    & store(add, <!q,{p}>, Claims) & assign(speaker, proponent) & assign(listener, opponent)
    & assign(opponent, speaker) } & { { move(mandate, next, Presupposing, {p}) }
    || { move(mandate, next, OnAccountOf, {p}) } || { move(mandate, next, OK, {p}) } } } } };
}
```

TDG is of interest because it was the first dialectical game to offer a turn structure different to the traditional two-player game with a strict progression of turns between the players. TDG instead provides a more liberal turn organisation in which the speaker and listener roles are assigned dependent upon the specific locutions that are played rather than just that a move has been played. Thus a player may make multiple moves during a single turn, only stopping once a specific move is made that mandates a response by another player.

4.6. ASD

ASD is the first game to incorporate argumentation schemes at the dialectical level [44]. ASD is based on the moves of CB but incorporates an additional Pose move. However this move conflates two semantically distinct moves into the single Pose move. The Pose move enables a player to attack an argument by posing a critical question from the scheme associated with the argument. However, there are two types of attack, an attack on an exception, and an attack on an assumption each of which leads to a different set of legal responses. Because there are different requirements for each aspect of the move, and different effects as a result, it is sensible to treat them as two separate moves in the reformulation. In ASD$_{DGDL}$ two new moves are introduced in place of the Pose move to account for the asking of critical questions associated with an argumentation scheme, the AssumptionAttack and ExceptionAttack. Neither of the attack moves is available to the players until an argument is established that corresponds to an argumentation scheme, at which point the player can pose a critical question related to the associated argumentation scheme.

```
ASD{
  { turns, magnitude:single, ordering:strict };
  {players, min:2, max:2 };
  { player, id: black };
  { player, id: white };
  { store, id:CS, owner:black, structure:set, visibility:public } ;
  { store, id:CS, owner:white, structure:set, visibility:public } ;

  {statement, {p},
    { if { corresponds(<p,{q}>, ImmediateConsequence) } then { store(add, {p}, CS, speaker) &
    store(add, {p}, CS, listener) & store(add, {q}, CS, listener)} else { store(add, {p}, CS, speaker) } } };
  {withdraw, {p},
    { if { corresponds(<p,{q}>, ImmediateConsequence) & inspect(!in, {q}, CS, listener, current ) }
    then { store(remove, p, CS, speaker) } } };
  {question, {p}
    { { move(mandate, next, statement, {p}) } || { move(mandate, next, statement, {!p}) }
    || { move(mandate, next, withdraw, {p}) } } };
  {challenge, {p}
    { store(add, {p}, CS, listener) & { { move(mandate, next, withdraw, {p}) }
    || { move(mandate, next, statement, {q}, corresponds(<p,q>, ImmediateConsequence)) } } } };
  {AssumptionAttack, {p}
    { if { event(last, <statement,{q}>) & corresponds(<p,{q}>, schemeID) }
    then{ { move(mandate, next, statement, {p}) } || { move(mandate, next, statement, !{p}) }
    || { move(mandate, next, withdraw, {p}) } } } };
  {ExceptionAttack, {p}
    { if { event(last, <statement,{q}>) & corresponds(<p,{q}>, schemeID) }
    then { { move(mandate, next, statement, {p}) } || { move(mandate, next, statement, {!p}) } ||
    { move(mandate, next, withdraw, {p}) } || { move(mandate, next, challenge, {!p}) } } } };
  {WithdrawPlusChallenge, {p}
    { if { corresponds(<p,{q}>, ImmediateConsequence) & inspect(!in, {q}, CS, listener, current ) }
    then { { store(remove, {p}, CS, speaker) & store(add, {p}, CS, listener) } & {
    { move(mandate, next, withdraw, {p}) }
    || { move(mandate, next, statement, {q}) & corresponds(<p,q>, ImmediateConsequenceScheme) &
    extCondition(Scheme, ImmediateConsequenceScheme) } } } } ;
}
```

The reformulation of a range of extant games into a common language has demonstrated some of the expressive power of DGDL. This has been achieved because the language was developed primarily to support comparative testing of dialectical games prior to deployment as agent communication protocols and therefore the game features expressly supported by DGDL originated from an investigation of existing games that might be applied to agent communication. The exercise of

reformulating the disparate games into a single description language has also shed light on some problems with existing specifications. A number of the games suffer from locutional overloading, where a single locution actually plays many different roles within a dialogue. This can be tackled by splitting each role of the overloaded locution into a separate closely defined locution. The lack of termination and victory conditions in most philosophical games makes them unsuitable for computational implementation in any situation except where an open-ended dialogue of indeterminate length is to occur.

4.7. Lorenzen's dialogue logic

A survey of dialogue games would not be complete without at least reference to Lorenzen's dialogue logic which has provided a semantics for intuitionistic and classical logics [19] and which has provided useful tools for the study of non-classical logics [48]. By integrating Lorenzen's dialogue logic approach into to Hamblin-style formal dialectics there is an opportunity to integrate inference steps into arbitrary dialogue systems and thereby construct a flexible bridge between dialogical argumentation processes and monological inference processes.

A Dialogue Logic game is a two-person, perfect information game, the players, a proponent and opponent, alternate turns and have complete knowledge of the state of the game. Initially, the proponent asserts a thesis, and the opponent may assert zero or more hypotheses. The aim of the dialogue is for the players to show the validity of the proponents thesis.

Procedural rules roughly equate to Hamblin's locutional rules and define the general structure of how the players can interact with earlier assertions. These rules do not depend upon the logical form of a player's assertion, with the loose exception of the first rule.

(1) The proponent may assert an atomic formula only after the opponent has asserted it.
(2) A player may defend against only the most recent attack that has not yet been responded to.
(3) An attack may be defended against only once.
(4) An assertion made by the proponent may be attacked only once.

Particle rules roughly equate to Hamblin's syntactical rules and define how a response can be formulated depending upon the actual content of the earlier move that is being responded to. For example, if the content contained a conjunction then the player can attack the left conjunct or the right conjunct. In the following reformulation, we confine ourselves to particle rules for the conjunction, disjunction, negation, and implication as follows:

Formula	Attack	Defense
$\alpha \wedge \beta$	?L	α
	?R	β
$\alpha \vee \beta$	?	α or β
$\neg\alpha$	α	–
$\alpha \rightarrow \beta$	α	β

It should be noted that the particle rules for further connectives and the quantifiers can also be formulated to extend this basic game in order to work with specific logics using a DGDL formulation of a Dialogue Logic game. Termination of a Dialogue Logic game occurs when a player runs out of legal moves. If, according to the particle and structural rules, a player cannot make another move then that player has lost and the other player has won.

```
Lorenzen{
  { turns, magnitude:single, ordering:strict };
  { players, min:2, max:2 };
  { player, id:proponent };
  { player, id:respondent };
  { store, id:assertions, owner:proponent, structure:set, visibility:public };
  { store, id:assertions, owner:opponent, structure:set, visibility:public };
  { store, id:attacks, proponent:, structure:stack, visibility:public };
  { store, id:attacks, opponent:, structure:stack, visibility:public };

  {Initial, scope:initial,
    { move(mandate, next, Thesis, proponent) } };
  {WinLoss, scope:movewise,
    { if { size(LegalMoves, speaker, empty) }
    then { assign(listener, winner) & assign(speaker, loser) & status(complete, Lorenzen) } } };

  {Thesis, {p},
    { store(add, {p}, assertions, proponent) } };
  {Hypothesis, {P},
    { if { event(last, Thesis) } then { store(add, {P}, assertions, opponent) } } };
  {AttackLeftConjunct, {p},
    { if { inspect(in, {p}, assertions, listener) & inspect(!in, {p}, attacks, speaker) }
    then { store(add, {p}, attacks, speaker) & move(mandate, future, DefenceLeft) } } };
  {AttackRightConjunct, {p},
    { if { inspect(in, {p}, assertions, listener) & inspect(!in, {p}, attacks, speaker) }
    then { store(add, {p}, attacks, speaker) & move(mandate, future, DefenceRight) } } };
  {AttackDisjunction, {p},
```

```
    { if { inspect(in, {p}, assertions, proponent) & inspect(!in, {p}, attacks, opponent) }
    then { { store(add, {p}, attacks, speaker) } &
    { move(mandate, future, DefenceLeft) } || { move(mandate, future, DefenceRight) } } } };
  {AttackNegation, {!p},
    { if { inspect(in, {!p}, assertions, listener) & inspect(!in, {!p}, attacks, speaker) }
    then { store(add, {!p}, attacks, speaker) &
    store(add, {!p}, assertions, speaker) & move(mandate, future, Defense, {!p}) } } };
  {AttackImplication, {p,q},
    { if { inspect(in, {p}, assertions, listener) & inspect(!in, {p}, attacks, speaker) }
    then { store(add, {p}, attacks, speaker) & move(mandate, future, Defence, {q}) } } };
  {DefenceLeft, {p},
    {if { inspect(top, {p}, attacks, listener) & event(past, AttackLeftConjunct, {p}, listener) }
    then { store(add, {p}, assertions, speaker) & store(remove, {p}, attacks, listener) } } };
  {DefenceRight, {p},
    {if { inspect(top, {p}, attacks, listener) & event(past, AttackRightConjunct, {p}, listener) }
    then { store(add, {p}, assertions, speaker) & store(remove, {p}, attacks, listener) } } };
  {Defence, {p},
    {if { inspect(top, {p}, attacks, listener) & event(past, AttackDisjunction, {p}, listener) }
    then { store(add, {p}, assertions, speaker) & store(remove, {p}, attacks, listener) } } };
}
```

This description uses an artifact stack to store a last-in-first-out ordered set of attacks so that a player can always respond to the last attack that their opponent made and a public artifact store for each player is used to track their assertions. Because DGDL does not specify a content language the particle rules are reformulated into equivalent moves, so that each type of attack and defence is individually represented. This enables the description to specify that for a DefenceLeft move to be legal an AttackLeftConjunct move must have been played.

4.8. MAgtALO

The reformulations previously described have all pertained to games extant in either the philosophical or agent communication literature. An additional area that DGDL can usefully be applied to is supporting the interchange of dialogue protocols between argumentation software applications. Two good examples are MAgtALO [45] and InterLoc [41]. In MAgtALO, agents and users interact via communicative acts that are regulated by a dialectical game protocol.

```
MAgtALO{
  { turns, magnitude:single, ordering:liberal };
  { players, min:2, max:undefined };
  { player, id:initiator, role:speaker };
  { player, id:$PlayerID$, role:speaker };

  {Initial, scope:initial,
    { move(mandate, next, Start, initiator) } };

  {Start, {p},
    { if { player(initiator) } then { { move(mandate, next, Agree, initiator) }
    || { move(mandate, next, Disagree, initiator) } || { move(mandate, next, WhoAgrees, initiator) } } } };
  {Agree, {p},
    { move(mandate, next, GiveReason) } };
  {Disagree, {p},
    { move(mandate, next, GiveReason) } };
  {WhoAgrees, {p},
    { if { player(initiator) } then { { move(permit, next, Agree) } || { move(permit, next, Disagree) } } } };
  {GiveReason, {p},
    { { move(mandate, next, Agree) } || { move(mandate, next, Disagree) } } };
  {MakeCurrent, {p},
    { if { player(initiator) } then { { move(mandate, next, Agree) } || { move(mandate, next, Disagree) }
    || { move(mandate, next, WhoAgrees) } } } };
  {ElicitAgreement, {p},
    { if { player(initiator) } then { { move(mandate, next, Agree) } || { move(mandate, next, Disagree) } } } };
  {Why, {p},
    { if { player(initiator) } then { move(mandate, next, GiveReason) } } };
  {DifferentReason, {p},
    { if { player(initiator) } then { move(mandate, next, GiveReason) } } };
  {MoreReasons, {p},
    { if { player(initiator) } then { move(mandate, next, GiveReason) } } };
}
```

The MAgtALO game does not enforce a turn structure, any player can interject at any point by playing a permissible move. However the game puts the initiating player into a privileged position which enables them to, to some degree, direct the dialogue by opening up the dialogue through undirected questions such as asking “who agrees?”, or by directing questions towards particular players to elicit agreement. The main aim of this game is to facilitate an open-ended dialogue in which the players explore the issues raised during the dialogue. The game thus supports this end and does not attempt to regulate the things that are said beyond attempting to get reasons supporting every position that is taken.

4.9. Creative Thinking Game (CTG)

Another dialogue game used in online argumentation is the Creative Thinking Game (CTG) found in InterLoc. CTG is novel because of its wide range of available locutions and its extensive use of scaffolding. Scaffolding is the use of openers such as “Let me say more about that…” to indicate to users the way that the locution is expected to be used. Currently CTG is expressed using XML which is used to permit or prohibit moves through the InterLoc interface. As a first step towards supporting autonomous agents engaging in InterLoc dialogues, CTG has been reformulated using DGDL. Unfortunately the

CTG DGDL description is quite long because of its extensive set of locutions and so only a short but representative extract is presented here which includes the header describing the game components and a selection of interaction rules.

```
CTG{
  { turns, magnitude:single, ordering:liberal };
  { players, min:1, max:undefined };
  { player, id:$PlayerID$, role:speaker };

  {Initial, scope:initial,
    { { move(,next,Suggest1) } || { move(propose,next,Suggest3) } || { move(propose,next,Suggest6) } } };

  {Suggest1, {p}, "My idea is", { { move(propose,next,Suggest3) } || { move(propose,next,Check6) }
    || { move(propose,next,Agree2) }
    || { move(propose,next,Transform6) } || { move(propose,next,Agree4) } } };
  {Suggest2, {p}, "Just imagine", { { move(propose,next,Suggest3) } || { move(propose,next,Check6) }
    || { move(propose,next,Agree2) }
    || { move(propose,next,Transform6) } || { move(propose,next,Agree4) } } };
  {Suggest3, {p}, "What if", { { move(propose,next,Agree3) } || { move(propose,next,Transform6) }
    || { move(propose,next,Agree2) }
    || { move(propose,next,Agree4) } || { move(propose,next,Check8) } } };
  {Suggest4, {p}, "How about", { { move(propose,next,Check6) } || { move(propose,next,Agree3) }
    || { move(propose,next,Transform6) }
    || { move(propose,next,Agree4) } || { move(propose,next,Suggest3) } } };
  {Suggest5, {p}, "I feel", { { move(propose,next,Suggest3) } || { move(propose,next,Check6) }
    || { move(propose,next,Agree2) }
    || { move(propose,next,Transform6) } || { move(propose,next,Suggest3) } } };
  {Suggest6, {p}, "I think", { { move(propose,next,Suggest3) } || { move(propose,next,Check6) }
    || { move(propose,next,Agree2) }
    || { move(propose,next,Transform6) } } };
  {Suggest7, {p}, "Let me say more about that", { { move(propose,next,Question6) } || { move(propose,next,Check6) }
    || { move(propose,next,Agree5) } || { move(propose,next,Agree1) } } };
  {Suggest8, {p}, "An example", { { move(propose,next,Check6) } || { move(propose,next,Check8) }
    || { move(propose,next,Transform7) }
    || { move(propose,next,Agree5) } || { move(propose,next,Agree1) } } };
  {Question1, {p}, "Why?", { { move(propose,next,Suggest7) } || { move(propose,next,Transform6) }
    || { move(propose,next,Suggest5) } } };
  {Question2, {p}, "Can you say more on that?", { { move(propose,next,Suggest7) }
    || { move(propose,next,Suggest5) } } };
  {Question3, {p}, "Does this connect with anything for you?", { { move(propose,next,Suggest5) }
    || { move(propose,next,Transform2) }
    || { move(propose,next,Maintain2) } } };
  {Question4, {p}, "What do you mean when you say?", { { move(propose,next,Suggest7) }
    || { move(propose,next,Transform6) } } };
  {Question5, {p}, "Why do you think that?", { move(propose,next,Suggest6) } };
  {Question6, {p}, "Why do you feel that?", { { move(propose,next,Suggest5) } || { move(propose,next,Suggest6) }
    || { move(propose,next,Suggest7) } } };
  {Question7, {p}, "What are the possible alternatives?", { { move(propose,next,Transform6) }
    || { move(propose,next,Check4) } } };
  {Question8, {p}, "Has anyone got another idea?", { { move(propose,next,Suggest1) } || { move(propose,next,Suggest4) }
    || { move(propose,next,Suggest3) } || { move(propose,next,Suggest6) } } };
}
```

The complete reformulation however is available from the DGDL repository.[6]

4.10. The bargaining game

When applied to original formulations of game rules, DGDL provides clean well specified game descriptions that can easily be implemented without suffering from the ambiguity and overloading problems found in extant games. An example of this can be seen in the formulation and presentation of the bargaining game (BG) developed to explore the fallacy of bargaining [55]. BG comprises two separate rule-sets, one describing interactions for a persuasion game (PG0) and the other for a negotiation game (NG0), and rules for making a natural shift from a persuasion dialogue to a negotiation dialogue when the situation dictates. BG is novel because it incorporates a *glissement* style of dialectical shift [52, p. 102] which is not found in any other game. Although CPD outlines the process for embedding an RPD game into a PPD game, the shift is a *deplacement* [52, p. 102] and the rules are not sufficiently delineated to support a reformulation.

```
BG{
  PG0{
    { turns, magnitude:single, ordering:strict };
    { players, min:2, max:2 };
    { player, id:initiator, role:speaker };
    { player, id:respondent, role:listener };
    { store, id:CS, owner:initiator, structure:set, visibility:public };
    { store, id:CS, owner:respondent, structure:set, visibility:public };

    {Initial, scope:initial,
      { status(active, PG0) & move(mandate, next, Request, {p}) } };
    {Progression, scope:turnwise,
      { if { inspect(in, {p}, CS, initiator, initial) & inspect(in, {p}, CS, initiator, current)
      & inspect(in, <p,{q}>, CS, initiator, current) & event(last, Reject, {p}, respondent) }
      then { status(active, NG0)} } };
    {Termination, scope:turnwise,
      { if { { inspect(in, {p}, CS, initiator, initial) & inspect(!in, {p}, CS, initiator, current) }
      || { inspect(in, {p}, CS, initiator, initial) & inspect(in, {p}, CS, respondent, current) } }
      then { status(terminate, BG0) } } };
```

[6] https://github.com/siwells/DGDL.

```
    {Request, {p},
      { { store(add, {p}, CS, speaker) & } { { move(mandate, next, Accept, {p}) }
      || { move(mandate, next, Reject, {p}) } || { move(mandate, next, Challenge, {p}) } } } };
    {Accept, {p},
      { if { event(last, Request, {p}) } then { store(add, {p}, CS, speaker) } } };
    {Reject, {p},
      { if { event(last, Request, {p}) } then { { store(remove, {p}, CS, speaker) }
      & { { move(mandate, next, Challenge, {p}) } || { move(mandate, next, Withdraw, {p}) } } } };
    {Challenge, {p},
      { if { { event(last, Request, {p}) } || { event(last, Reject, {p}) } || { event(last, Defense, {p}) } }
      then { { move(mandate, next, Defense, {p}) } || { move(mandate, next, Reject, {p}) }
      || {move(mandate, next, Withdraw, {p}) } } } };
    {Defense, {p,q},
      { { update(add, {p}, CS, speaker) & store(add, {q}, CS, speaker) & store(add, <p,{q}>, CS, speaker) }
      & { { move(mandate, next, Challenge, {p}) } || { move(mandate, next, Challenge, {q}) }
      || { move(mandate, next, Challenge, <p, {q}) } || { move(mandate, next, Reject, {p}) }
      || { move(mandate, next, Reject, {q}) } || { move(mandate, next, Reject, <p,{q}>) }
      || { move(mandate, next, Accept, {p}) } || { move(mandate, next, Accept, {q}) }
      || { move(mandate, next, Accept, <p,{q}>) } } } };
    {Withdraw, {p},
      { if { { event(last, Challenge, {p}) } || { event(last, Reject, {p}) } } then { status(terminate, BG0) } } } ;
  }
  NG0{
    { turns, magnitude:single, ordering:strict };
    { players, min:2, max:2 };
    { player, id:initiator, role:speaker };
    { player, id:respondent, role:listener };
    { store, id:CS, owner:initiator, structure:set, visibility:public };
    { store, id:CS, owner:respondent, structure:set, visibility:public };

    {Initial, scope:initial,
      { if { inspect(in, {p}, CS, initiator, initial) & inspect(in, {p}, CS, initiator, current)
      & inspect(!in, {p}, CS, respondent, current) & move(mandate, next, Request, {p}) }
      then { move(mandate, next, Offer, {p,q}) } } };
    {Termination, scope:turnwise,
      { if { { inspect(in, {p}, CS, initiator, initial) & inspect(!in, {p}, CS, initiator, current) }
      || { inspect(in, {p}, CS, initiator, initial) & inspect(in, {p}, CS, respondent, current) } }
      then { status(terminate, BG0) } } };

    {Offer, {p,q},
      { if { inspect(!in, <Offer, {p,q}>, CS, speaker) } then { { update(add, {p}, CS, speaker)
      & update(add, {q}, CS, speaker) & update(add, <Offer, {p,q}>, CS, speaker) }
      & { move(mandate, next, Accept, {p}) || move(mandate, next, Reject, {p})
      || move(mandate, next, Withdraw, {p}) || move(mandate, next, Offer, {p,r})
      || move(mandate, next, Offer, {s,q}) || move(mandate, next, Offer, {t,u}) } } } };
    {Accept, {p,q},
      { if { event(last, Offer, {p,q}) } then { update(add, {p}, CS, speaker)
      & update(add, {q}, CS, speaker) & update(add, <Offer, {p,q}>, CS, speaker) } } };
    {Reject, {p,q},
      { if { event(last, Offer, {p,q} ) then { { move(mandate, next, Offer, {p,r}) }
      || { move(mandate, next, Offer, {s,q}) } || { move(mandate, next, Offer, {t,u}) } } } };
    {Withdraw, {p},
      { if { { event(last, Offer, {p,q} ) || { event(last, Reject, {p,q} ) } then { status(terminate, BG0) } } } };
  }
}
```

A shift from the initial PG0 game to an instance of the NG0 game can occur once the dialogue initiator's thesis has been rejected by the respondent. Rather than having a hard shift from the persuasion dialogue of PG0 to the negotiation dialogue of NG0 at this point, the rules are set up to enable flexibility as to which type of dialogue continues dependent upon the move played in the next turn. A player could continue within the persuasion dialogue, by making a move from the legal rules of PG0, or can elect to shift an NG0 sub-dialogue, by making a move from the set of legal rules of NG0. This is accomplished through the activation of the NG0 rule-set triggered by the progression rule of PG0 so that both rule-sets are active, and therefore the set of legal moves available to the players in subsequent turns include legal moves from both games. In BG0 the type of shift is what Walton and Krabbe identify as a *glissement*, a gradual shift from one dialogue type to another, possibly over the course of a number of moves, as opposed to a *deplacement*, a sudden shift from one dialogue type to the next.

A workable set of rules for shifting between different sets of game rules is quite novel, although an earlier version of the game was explored in an investigation into the fallacy of bargaining [55]. Walton and Krabbe do suggest that a shift can occur between a PPD dialogue and an RPD dialogue within an instance of the CPD game (that they call PPD_1) however the rules to support the shift are not fully specified [52, pp. 163–164]. Although shifts have been explored in the context of dialogue [52, p. 102] and agent communication by McBurney and Parsons [32] and Reed [42] no games have been implemented that make use of them.

The advantage of utilising dialectical shifts is that small, carefully defined sets of rules can be constructed which describe behaviours specific to a given dialogical context. This approach, creating constrained games tailored to a particular context has proven popular in agent communication languages where there are a plethora of dialogue games to describe persuasion dialogues [1,28], negotiation dialogues [15,37,34,38], enquiry dialogues [30], and deliberation dialogues [13], amongst others. However these games are described in terms of individual games, with the overall dialogue between agents being a series of separate interactions rather than as a single extended dialogue, containing potentially many embeddings. The advantage of this latter approach is that players can set up commitment stores which persist throughout the dialogue informing each embedding whilst at the same time the rules governing available moves are specific to the aims of the dialogue associated with each individual embedding. In this way rule-bound dialogues can be generated that are closer to the flexible and efficient natural rhythms of real-world dialogues.

5. Further work

Work on the A4A is ongoing, and DGDL is but the first step towards an architecture to support the development and deployment of argumentative dialogue games in software systems. A specific area of extension will be to specify a DGDL logical language including connectives and quantifiers. This will enable better integration of Hamblin-style dialogue games with Lorenzen-style games and stronger support for bridging between monological inference and dialogical interaction. The terminal symbols of DGDL have been demonstrated to be sufficient to represent many extant dialectical games, however it is not suggested that these are sufficient to represent all possible future games. For this reason one design decision that was made when developing DGDL was to ensure that extending the grammar to account for new capabilities can be performed in a simple, structured manner through the use of the condition and effect predicates. Where this is not sufficient, for example, where new games are developed which include wholly new capabilities, then DGDL may have to be extended. As well as seeking to extend DGDL further work will seek to explore new games formulated from the current range of terminal symbols with the aim of establishing an open library of clearly and unambiguously described dialectical games. In addition to exploring dialectical games expressed using DGDL, another strand of work will seek to provide reference implementations of software engines and programming libraries for implementing and playing games described in DGDL. Implementation of game playing engines will aid the development of software that uses dialectical games and add impetus to the WWAW by easing the process of developing argumentation applications which share arguments online and communicate amongst themselves.

6. Conclusions

Communication is an important topic within agent research and is a fundamental factor in the development of robust and efficient multiagent systems (MAS). Similarly, an agent's capacity to engage in argument has been recognised as a key component of an agent's ability to act using complex, dynamic, and uncertain knowledge. Dialectical games provide a way to unify the two approaches, providing a mechanism for communication which incorporates argumentative behaviours. However, whilst there has been much research into agent communication languages there has not been a similar effort to produce languages for describing dialectical games. It is this void that DGDL fills, providing a means to share dialectical games between disparate users. Whilst agents provide an ideal deployment area for dialectical games the nascent domain of online argumentation, and particularly the development of the WWAW, is another area in which dialectical games find a natural home. The WWAW promises a semantic web of argument in which users interact to create, discover, and explore arguments and dialectical games, underpinned by DGDL, will provide an important part of the mechanism for interacting with the WWAW.

To realise these goals however, tools are required which will be used to build the WWAW. Whilst AIF supports the interchange of arguments and has some tools to support its use, the AIF+ is not yet sufficiently mature to underpin the interactive elements of the WWAW and therefore tools to support its use will not be available in the short term. It is into this gap that DGDL is inserted providing a solid syntactic underpinning for new tools to support argumentative interaction on the WWAW. This paper outlines the first step towards such a toolset, defining a form for syntactically correct descriptions of dialectical games so that tools to create and consume dialectical games can be developed.

The strongest message that emerges from this research is that whilst some steps are being taken to formalise and increase adoption of dialectical games, there are still many niches where they have yet to be fully exploited. It is with a flexible and powerful toolset that these niches will be better explored and it is upon DGDL that this toolset will be developed.

In brief, with the persistent trend towards distribution and interconnection of software systems it is imperative to fully exploit whichever communicative mechanisms are applicable. Dialectical games are ideal mechanisms for communication in argumentative contexts yet they will not be fully exploited unless disparate developers can work together, share their findings, build upon each others discoveries, and construct systems which interact robustly despite a dynamic and uncertain world.

Acknowledgements

The authors wish to thank the developers of InterLoc for kindly providing the descriptions of the games used in the system. Much of this paper describes work undertaken during PhD research in the EPSRC funded Information Exchange project. Wells' thesis [53] was examined by Trevor Bench-Capon, whom the authors wish to thank for his insightful comments and feedback.

References

[1] K. Atkinson, What should we do? Computational representation of persuasive argument in practical reasoning, PhD thesis, University of Liverpool, 2005.
[2] E.M. Barth, E.C.W. Krabbe, From Axiom to Dialogue: A Philosophical Study of Logics and Argumentation, Walter de Gruyter & Co., 1982.
[3] T.J.M. Bench-Capon, Specification and implementation of Toulmin dialogue game, in: Proceedings of JURIX 98, 1998, pp. 5–20.

[4] T.J.M. Bench-Capon, T. Geldard, P.H. Leng, A method for the computational modelling of dialectical argument with dialogue games, Artificial Intelligence and Law 8 (2000) 223–254.
[5] L. Carlson, Dialogue Games, D. Reidel Publishing Company, 1983.
[6] C. Chesnevar, J. McGinnis, S. Modgil, I. Rahwan, C. Reed, G. Simari, M. South, G. Vreeswijk, S. Willmott, Towards an argument interchange format, Knowledge Engineering Review 21 (4) (2006) 293–316.
[7] R.A. Girle, Dialogue and entrenchment, in: Proceedings of the 6th Florida Artificial Intelligence Research Symposium, 1993, pp. 185–189.
[8] R.A. Girle, Knowledge organized and disorganized, in: Proceedings of the 7th Florida Artificial Intelligence Research Symposium, 1994, pp. 198–203.
[9] R.A. Girle, Commands in dialogue logic, in: Practical Reasoning: International Conference on Formal and Applied Practical Reasoning, in: Lecture Notes in AI, vol. 1085, Springer, 1996, pp. 246–260.
[10] C.L. Hamblin, Fallacies, Methuen and Co. Ltd., 1970.
[11] C.L. Hamblin, Mathematical models of dialogue, Theoria 37 (1971) 130–155.
[12] J. Hintikka, Language games for quantifiers, in: Studies in Logical Theory, in: American Philosophical Quarterly Monograph Series, vol. 2, Blackwell, Oxford, 1968.
[13] D. Hitchcock, P. McBurney, S. Parsons, A framework for deliberation dialogues, in: Argumentation and Its Applications, Proceedings of the Fourth Biennial Conference of the Ontario Society for the Study of Argumentation (OSSA 2001), 2001.
[14] W. Kamlah, P. Lorenzen, Logical Propaedeutic: Pre-School of Reasonable Discourse, University Press of America, 1984.
[15] S. Kraus, K. Sycara, A. Evenchik, Reaching agreements through argumentation: A logical model and implementation, Artificial Intelligence 104 (1–2) (1998) 1–69.
[16] Y.K. Labrou, T. Finin, History, state of the art and challenges for agent communication languages, Informatik/Informatique 1 (2000) 17–24.
[17] A.R. Lodder, DiaLaw, Law and Philosophy Library, Kluwer Academic Publishers, 1999.
[18] A.R. Lodder, A. Herczog, Dialaw: A dialogical framework for modeling legal reasoning, in: International Conference on Artificial Intelligence and Law, 1995, pp. 146–155.
[19] P. Lorenzen, Logik und Agon, in: Arti del XII Congresso Internationale de Filosofia, Venezia, 1958, pp. 187–194.
[20] J.D. Mackenzie, How to stop talking to tortoises, Notre Dame Journal of Formal Logic XX (4) (1979) 705–717.
[21] J.D. Mackenzie, Question begging in non-cumulative systems, Journal of Philosophical Logic 8 (1979) 117–133.
[22] J.D. Mackenzie, Why do we number theorems? Australasian Journal of Philosophy 58 (2) (1980) 135–149.
[23] J.D. Mackenzie, The dialectics of logic, Logique et analyse 24 (1981) 159–177.
[24] J.D. Mackenzie, Begging the question in dialogue, Australasian Journal of Philosophy 62 (2) (1984) 174–181.
[25] J.D. Mackenzie, No logic before Friday, Synthese 63 (1985) 329–341.
[26] J.D. Mackenzie, Four dialogue systems, Studia Logica 49 (1990) 567–583.
[27] J.D. Mackenzie, Context of begging the question, Argumentation 8 (1994) 227–240.
[28] N. Maudet, B. Chaib-draa, M.A. Labrie, Request for action reconsidered as dialogue game based on commitments, in: Workshop on Agent Communication Language (AAMAS'02), 2002.
[29] N. Maudet, F. Evrard, A generic framework for dialogue game implementation, in: Proceedings of the Second Workshop on Formal Semantics and Pragmatics of Dialogue, 1998.
[30] P. McBurney, S. Parsons, Representing epistemic uncertainty by means of dialectical argumentation, Annals of Mathematics and Artificial Intelligence 32 (1–4) (2001) 125–169.
[31] P. McBurney, S. Parsons, Dialogue games in multi-agent systems, Informal Logic 22 (3) (2002) 257–274.
[32] P. McBurney, S. Parsons, Games that agents play: A formal framework for dialogues between autonomous agents, Journal of Logic, Language and Information 11 (3) (2002) 315–334.
[33] P. McBurney, S. Parsons, M. Wooldridge, Desiderata for agent argumentation protocols, in: Proceedings of the First AAMAS, 2002, pp. 402–409.
[34] P. McBurney, R.M. van Eijk, S. Parsons, L. Amgoud, A dialogue-game protocol for agent purchase negotiations, Journal of Autonomous Agents and Multi-Agent Systems 7 (3) (2001) 235–273.
[35] D. Moore, D. Hobbes, Computational uses of philosophical dialogue theories, Informal Logic 18 (2–3) (1996) 131–163.
[36] D.J. O'Keefe, Two concepts of argument, The Journal of the American Forensic Association 13 (3) (1977) 121–128.
[37] S. Parsons, C. Sierra, N. Jennings, Agents that reason and negotiate by arguing, Journal of Logic and Computation 8 (3) (1998) 261–292.
[38] I. Rahwan, S.D. Ramchurn, N.R. Jennings, S. McBurney, P. Parsons, L. Sonenberg, Argumentation-based negotiation, Knowledge Engineering Review 18 (4) (2003) 343–375.
[39] I. Rahwan, F. Zablith, C. Reed, Laying the foundations for a world wide argument web, Artificial Intelligence 171 (2007) 897–921.
[40] I. Rahwan, F. Zablith, C. Reed, Towards large scale argumentation support on the semantic web, in: Proceedings of AAAI-07, 2007, pp. 1446–1451.
[41] A. Ravenscroft, Promoting thinking and conceptual change with digital dialogue games, Computer Assisted Learning 23 (6) (2007) 453–465.
[42] C. Reed, Dialogue frames in agent communication, in: Proceedings of the 3rd International Conference on Multi Agent Systems, IEEE Press, 1998, pp. 246–253.
[43] C. Reed, K. Budzynska, How dialogues create arguments, in: Proceedings of the 7th Conference of the International Society for the Study of Argumentation, ISSA 2010, Sic Sat, Amsterdam, 2010.
[44] C. Reed, D. Walton, Argumentation schemes in dialogue, in: H.V. Hansen, C.W. Tindale, R.H. Johnson, J.A. Blair (Eds.), Dissensus and the Search for Common Ground (Proceedings of OSSA 2007), 2007.
[45] C. Reed, S. Wells, Using dialogical argument as an interface to complex debates, IEEE Intelligent Systems Journal: Special Issue on Argumentation Technology 22 (6) (2007) 60–65.
[46] C. Reed, S. Wells, G.W.A. Rowe, J. Devereux, Aif+: Dialogue in the argument interchange format, in: 2nd International Conference on Computational Models of Argument (COMMA 2008), 2008.
[47] N. Rescher, Dialectics, State University of New York Press, Albany, 1977.
[48] H. Rückert, Why dialogical logic?, in: H. Wansing (Ed.), Essays on Non-Classical Logic, in: Advances in Logic, vol. 1, World Scientific, River Edge, 2001.
[49] D. Thomas, Programming Ruby, The Pragmatic Programmers, 2005.
[50] S. Toulmin, The Uses of Argument, Cambridge University Press, 1958.
[51] D.N. Walton, Logical Dialogue-Games and Fallacies, University Press of America, 1984.
[52] D.N. Walton, E.C.W. Krabbe, Commitment in Dialogue, SUNY Series in Logic and Language, State University of New York Press, 1995.
[53] S. Wells, Formal dialectical games in multi-agent argumentation, PhD thesis, University of Dundee, 2007.
[54] S. Wells, C. Reed, Formal dialectic specification, in: I. Rahwan, P. Moraitis, C. Reed (Eds.), First International Workshop on Argumentation in Multi-Agent Systems, 2004.
[55] S. Wells, C. Reed, Knowing when to bargain, in: P.E. Dunne, T.J.M. Bench-Capon (Eds.), Computational Models of Argument, IOS Press, 2006, pp. 235–246.
[56] S. Wells, C. Reed, Dialectical games: Features, presentation, and representation, Technical report, School of Applied Computing, University of Dundee, 2008, http://quiddity.computing.dundee.ac.uk/swells/html/pubs.html.

[57] N. Wirth, What can we do about the unnecessary diversity of notation for syntactic definitions? Communications of the ACM 20 (11) (1977) 822–823.
[58] J. Woods, D.N. Walton, Arresting circles in formal dialogues, Journal of Philosophical Logic 7 (1978) 73–90.
[59] J. Woods, D.N. Walton, Question-begging and cumulativeness in dialectical games, Nous 16 (1982) 585–605.
[60] T. Yuan, Human computer debate, a computational dialectics approach, PhD thesis, Leeds Metropolitan University, 2004.