How to tell a logical story

Michael Schroeder
City University, London, msch@soi.city.ac.uk

Abstract

At the center of most plots in literature is a main character, who is stuck in a conflict and considers different arguments and options to resolve the conflict. In this paper, we show how to formalise such an argumentation process and we develop a formal argumentation framework, which caters for a declarative semantics of an argumentation process and an operational, efficient, goal-driven, top-down proof procedure to compute the argumentation process.

Next, we give an overview of how the argumentation framework is used in Ultima Ratio (Schroeder, Plewe, & Raab 1998; 1999), whose core is an argument knowledge base taken from a variety of literature pieces. The system allows one to define an agent, which is constituted by a set of arguments and assumptions. Facing a particular world, the agent's believes may be inconsistent triggering a rational monologue to deal with the situation.

Introduction

At the center of many literature pieces is a hero or heroine, who is stuck in a conflict and tries to decide which of the various options lead to the best solution. Consider, for example, the excerpt of the third scene in the third act of Shakespeare's Hamlet: The hero is unsure whether to kill Claudius - the assassin of Hamlet's father - or not. He argues that he should kill Claudius to revenge the murder. However, if he does kill him, Claudius, who is praying at that very moment, goes to heaven, which is too good for such a villain. A conflict.

In this paper, we use argumentation to model such plots. Agents, such as Hamlet, are defined by a set of arguments and assumptions. When facing a particular world, the agent's beliefs may be inconsistent triggering a rational monologue to deal with the situation. We give a theoretical account on how arguments attacking each other are used for conflict resolution. Our work is closely related to rational agents (Kowalski 1995) and logic programming and formal argumentation (Prakken & Sartor 1997; Dung 1993; 1995; Schroeder, Móra, & Alferes 1997; Schroeder 1999). Intuitively, argumentation treats the evaluation of a logic program as an argumentation process, where a goal *G* holds if all arguments supporting *G* cannot be attacked anymore. Thus, logic programming is seen as a discourse involving attacking and counter-attacking arguments.

Given such an argumentation framework, one can tell logical stories by formalising plots as argumentation processes. We review Ultima Ratio (Schroeder, Plewe, & Raab 1998; 1999), which allows one to choose agents from different literature pieces or to construct new agents from scratch. When facing a particular world, the agents may be stuck in a conflict and an argumentation process starts to unfold the conflict and possibly remove it.

Formal Argumentation in Philosophy

Since Leibniz's described the *calculus raciocinator* in 1679, researchers have been investigating how to automate argumentation. A problem is that many figures of arguments cannot be described formally. The Encyclopedia Britanica lists for example the following figures:

1. Semantical figures: Arguing by example, authority, or analogy,

2. Syntactical figures: arguing from the consequences, a pari (arguing from similar propositions), a fortiori (arguing from an accepted conclusion to an even more evident one), a contrario (arguing from an accepted conclusion to the rejection of its contrary), undercut (Attacking premisses), or rebut (Attacking conclusions)

The syntactical figures can be formally described by their form, i.e. syntax, and can therefore be easily automated. Although semantical figures such as arguing by authority may be formalised for particular domains (see e.g. (Sierra *et al.* 1997)), they are in general, not formalisable. This should not put us off, because it turns out that the syntactical figures of undercut and rebut are already sufficient to define the semantics of logic programs, which - in turn - makes logic programming the implementation language of choice for argumentation tools (see also (Kraus, Sycara, & Evenchik 1998)).

The relevance of an argument, i.e. should an agent accept it or immediately reject it, is an important issue in classical argumentation. Copi and Cohen (Copi & Cohen 1994) list, for example, 17 fallacies of relevance of arguments, only three of which can be expressed formally:

1. An *argument from ignorance* argues that a proposition is true simply on the basis that it has not been proved false, or that it is false because it has not been proved true.
2. *Begging the question* - also called *circular argumentation* - assumes the truth of what it seeks to prove in the effort to prove it.
3. *Division* assumes that what is true as a whole must be true in its parts and vice versa.

Interestingly, these three examples of fallacies involve all non-monotonic reasoning and require two kinds of negation:

1. Implicit negation *not a* to express the lack of evidence for *a*;
2. explicit negation $\neg a$ to state that there is an explicit proof for $\neg a$.
3. The two negations are related in that $\neg a$ implies *not a*.

With the two kinds of negation we can express the three fallacies mentioned above:

1. Arguments from Ignorance have the form $a \leftarrow not\ \neg a$ or $\neg a \leftarrow not\ a$.
2. Begging the question has the form of a positive loop, e.g. $a \leftarrow a$ or $a \leftarrow not\ a$ in its most reduced form. For many reasoning systems these positive loops and also negative loops $a \leftarrow not\ a$ lead to non-termination, while WFSX (Alferes & Pereira 1996) and its implementation in the REVISE system (Damásio, Pereira, & Schroeder 1997) deal properly with these invalid arguments.
3. Division requires non-monotonic reasoning (NMR) and contradiction removal. A typical example dealt with extensively in the NMR literature is a contradiction between flying birds and not-flying penguins: “Since birds can fly and penguins are birds, Tweety can fly”. He doesn’t, however. Statements like “birds fly” are usually intended to be read as “birds usually fly” or “birds fly, unless they are abnormal”. This can be expressed by implicit negation: $flies(X) \leftarrow bird(X), not\ ab(X)$. Adding the rule $bird(X) \leftarrow penguin(X)$ and $\neg flies(X) \leftarrow penguin(X)$ and $penguin(tweety)$ leads to a the contradiction for Tweeety. It can, however, be resolved by changing the assumption $not\ ab(tweety)$ and assuming $ab(tweety)$. For details see e.g. (Damásio, Pereira, & Schroeder 1997; Schroeder 1998).

In the next section, we present a formal argumentation framework capable to represent and automate the features discussed above.

Formal Foundations of Argumentation

During the last 5-10 years, researchers have been devoting much work to the semantics of logic programs. Among the different approaches that usually either emphasise an operational or a declarative view, argumentation semantics turned out to be a very intuitive approach. Rather than defining a semantics in technical terms, argumentation semantics uses the metaphor of argumentation as used in politics, law, discourse, etc. and formalises a part of it sufficient to give a meaning to extended logic programs (Alferes & Pereira 1996). Intuitively, argumentation semantics treats the evaluation of a logic program as an argumentation process, where a goal G holds if all arguments supporting G cannot be attacked anymore. Thus, logic programming is seen as a discourse involving attacking and counter-attacking arguments.

While argumentation in rhetoric comprises a variety of figures as mentioned above, logic programming can be described in terms of two figures: Reductio ad absurdum- and ground-attack (Dung 1995) or equivalently rebut and undercut (Prakken & Sartor 1997). The former classifies an argument that leads to a contradiction under the current believes and arguments, and the latter an argument that falsifies the premise of one of the current arguments.

Argumentation with Extended Logic Programming

Argumentation semantics in extended logic programming has been defined in (Dung 1993; Prakken & Sartor 1997). Our framework is based on (Prakken & Sartor 1997) with a modification avoiding some unintuitive results (Schroeder, Móra, & Alferes 1997).

Definition 1 *Extended Logic Program, Constraints*
An extended logic program is a (possibly infinite) set of rules of the form $L_0 \leftarrow L_1, \dots, L_l, notL_{l+1}, \dots, notL_m$ $(0 \leq l \leq m)$ *where each* L_i *is an objective literal* $(0 \leq i \leq m)$*. An objective literal is either an atom A or its explicit negation* $\neg A$*. Literals of the form not L are called default literals. A subset R of default literals which do not occur in the head of a rule is called revisables. The set of all objective literals is called the herbrand base* $\mathcal{H}(P)$*. A rule with head* $L_0 = \bot$ *is called* integrity constraint. *The symbol* $\bot$ *stands for falsity. A program P is inconsistent iff* $P \models \bot$*, otherwise it is consistent.*

While objective literals have a fixed truth values, default literals have a default truth value. For revisables this truth value may be changed. The limitation of revisability to default literals which do not occur as rule heads is adopted for efficiency reasons, but without loss of generality. We want to guarantee that the truth value of revisables is independent of any rules. Thus we can change the truth value of a revisable whenever necessary without considering an expensive derivation of the default literal's truth value.

Consider the excerpt of the third scene in the third act of Shakespeare's Hamlet below:

1 *Hamlet. [approaches the entry to the lobby]*
Now might I do it pat, now a'is a-praying-
2 And now I'll do't, *[he draws his sword]* and so a' goes to heaven,
3 And so am I revenged. That would be scanned:
4 A villain kills my father, and for that
5 I his sole son do this same villain send
6 To heaven...
7 Why, this is bait and salary, not revenge.

Hamlet is caught in a conflict. On the one hand he wants revenge for his father being murdered. On the other he knows that having revenge by killing Claudius, the murderer, is not possible since Claudius is praying at that very moment and would go to heaven which contradicts the goal of having revenge. The text can be formalised as follows:

a	$\mathit{praying}(\mathit{claudius})$.
b	$\mathit{in\_heaven}(X) \leftarrow \mathit{kills}(Y,X), \mathit{praying}(X)$.
c	$\mathit{took\_revenge\_on}(X,Y) \leftarrow \mathit{kills}(X,Y)$.
d	$\mathit{killed}(\mathit{claudius},\mathit{king})$.
e	$\neg \mathit{took\_revenge\_on}(X,Y) \leftarrow \mathit{in\_heaven}(Y)$.
f	$\mathit{goal\_revenge}(X,Y) \leftarrow \mathit{close}(X,Z), \mathit{killed}(Y,Z), \mathit{not\ justified}(\mathit{killed}(Y,Z))$.
g	$\mathit{close}(\mathit{hamlet},\mathit{king})$.
h	$\perp \leftarrow \mathit{goal\_revenge}(X,Y), \mathit{not\ took\_revenge\_on}(X,Y)$.
i	$\mathit{revisable}(\mathit{kills}(\mathit{hamlet},\mathit{claudius}),\mathit{false})$.

In line 1, Hamlet realizes that Claudius is praying. This is represented as a fact (a). In line 2, Hamlet continues that Claudius would go to heaven if killed while praying. Formally, this is an instantiation of the general rule (b). In line 3, Hamlet states that killing Claudius satisfies Hamlet's desire for revenge. Or more general (c). In line 4, Hamlet starts another line of thinking by mentioning the fact that Claudius killed Hamlet's father, the king (d). In line 7, Hamlet finds that he does not have revenge if he sends Claudius to heaven (e). Beside this direct translation of Hamlet's monologue to logic, we have to add further facts and rules which are mentioned throughout the scenes before or which are given implicitly. First of all, we need a rule to express when someone wants revenge (f). I.e. X wants to take revenge on Y if Y killed a person Z being close to X, and the killing is not justified. Left implicitly in the piece is the fact that Hamlet and his father are close to each other (g). To specify conflicting goals we use besides facts and rules integrity constraints. In this scene we state formally that it is contradictory to want to take revenge and not have it (h). Finally, we have to specify the assumptions Hamlet is willing to change to solve conflicts. For this scene, Hamlet adopts the default assumption of not killing Claudius. I.e. (i) states that Hamlet killing Claudius is assumed false initially, but may be changed in the course of argumentation.

To formalise human argumentation as shown in the previous example one has to detect first the assumptions the protagonist is willing to change. These assumptions are made revisable and are assigned to a default value. Secondly, the problem domain has to be modeled in terms of facts and rules. The two negations (*not* and $\neg$) are important for this modeling task. For example, $\mathit{not\ justified}(\mathit{killed}(X,Y))$ expresses that a murder is not justified as long as there is no explicit proof for the contrary. In contrast, $\neg \mathit{took\_revenge\_on}(X,Y) \leftarrow \mathit{in\_heaven}(Y)$ states that there is explicit evidence that X did not take revenge on Y if Y ends up in heaven. Besides the three ingredients of revisable assumptions, facts, and rules, we have to define which conclusions are contradictory. Naturally, we say that, for example, $\mathit{took\_revenge\_on}(X,Y)$ and its explicit negation $\neg \mathit{took\_revenge\_on}(X,Y)$ are contradictory, i.e. $\perp \leftarrow \mathit{took\_revenge\_on}(X,Y), \neg \mathit{took\_revenge\_on}(X,Y)$, but for convenience we are at liberty to define further conflicts such as, for example, $\perp \leftarrow \mathit{goal\_revenge}(X,Y), \mathit{not\ took\_revenge\_on}(X,Y)$.

The following definitions for argumentation are based on (Dung 1993; Prakken & Sartor 1997). In contrast to the latter we do not distinguish between strict and defensible rules. However, our results can be extended to this direction.

Definition 2 *Argument*
Let P be an extended logic program. An argument for a conclusion L is a finite sequence $A = [r_m, \dots r_n]$ of ground instances of rules $r_i \in P$ such that 1. for every $n \leq i \leq m$, for every objective literal L_j in the antecedent of r_i there is a $k < i$ such that L_j is the consequent of r_k. 2. L is the consequent of some rule of A; 3. No two distinct rules in the sequence have the same consequent. A sequence of a subset of rules in A being an argument is called subargument.

Example 3 *An argument for the conclusion $\mathit{goal\_revenge}(\mathit{hamlet},\mathit{claudius})$ is the sequence:*

$$\begin{aligned}
&\mathit{goal\_revenge}(\mathit{hamlet},\mathit{claudius}) \leftarrow \\
&\quad \mathit{close}(\mathit{hamlet},\mathit{king}), \\
&\quad \mathit{killed}(\mathit{claudius},\mathit{king}), \\
&\quad \mathit{not\ justified}(\mathit{killed}(\mathit{claudius},\mathit{king})); \\
&\mathit{close}(\mathit{hamlet},\mathit{king}) \leftarrow \mathit{true}; \\
&\mathit{killed}(\mathit{claudius},\mathit{king}) \leftarrow \mathit{true};
\end{aligned}$$

The Process of Argumentation

There are two ways of attacking an argument for a conclusion L. We may prove that the argument for L leads to a contradiction since there is also proof for $\neg L$. Such a counter-argument is called *rebut*. The second possibility is to attack the premises of the argument for L. If L's argument is based on an assumption *not* L' we can attack the argument with a counter-argument for L'. Such an attack is called undercut.

Definition 4 *Undercut, Rebut*
Let A_1 and A_2 be two arguments, then A_1 undercuts *A_2 iff A_1 is an argument for L and A_2 is an argument with assumption not L, i.e. there is an $r : L_0 \leftarrow L_1,\dots,L_l, not\ L_{l+1},\dots, not\ L_m \in A_2$ and a $l+1 \leq j \leq m$ such that $L = L_j$. A_1* rebuts *A_2 iff A_1 is an argument for L and A_2 is an argument for $\neg L$. A_1* attacks *A_2 iff A_1 undercuts or rebuts A_2.*

Example 5 *The argument $took\_revenge\_on(hamlet, claudius) \leftarrow kills(hamlet, claudius)$ can be attacked by the rebut*

$$\begin{aligned}
&\neg took\_revenge\_on(hamlet, claudius) \leftarrow \\
&\qquad in\_heaven(claudius);\\
&in\_heaven(claudius) \leftarrow \\
&\qquad kills(hamlet, claudius), praying(claudius);\\
&praying(claudius) \leftarrow true.
\end{aligned}$$

Definition 6 *Coherent, Conflict-free*
An argument is coherent if it does not contain subarguments attacking each other. A set Args of arguments is called conflict-free if no two arguments in Args attack each other.

Definition 7 *Defeat, Acceptable*
Let A_1 and A_2 be two arguments, then A_1 defeats *A_2 iff A_1 is empty and A_2 incoherent or A_1 undercuts A_2 or A_1 rebuts A_2 and A_2 does not undercut A_1. A_1* strictly defeats *A_2 iff A_1 defeats A_2 but not vice versa. A_1 is acceptable wrt. a set Args of arguments iff each argument undercutting A_1 is strictly defeated by an argument in Args.*

Our notion of acceptability deviates from Prakken and Sartor's definition (Prakken & Sartor 1997) where an argument A_1 is accepted if each *defeating* argument is accepted. Our notion is more credulous and leads to more intuitive results.

Example 8 *Consider the program $P = \{a \leftarrow not\ b; b \leftarrow not\ a; \neg a\}$, then $\neg a$ and $b \leftarrow not\ a$ are acceptable, whereas $a \leftarrow not\ b$ is not. For Prakken and Sartor's definition of acceptability there is no acceptable argument which contradicts the intuition of $\neg a$ being a fact.*

Definition 9 *Characteristic Function*
Let P be an extended logic program and S be a subset of arguments of P, then $F_P(S) = \{A \mid A$ is acceptable wrt. $S\}$ is called characteristic function. *A is* justified *iff A is in the least fixpoint of F_P. A is* overruled *iff A is attacked by a justified argument. A is* defensible *iff A is neither justified nor overruled.*

Argumentation is closely related to logic programming. While Dung uses argumentation to define a declarative semantics for extended logic programs, Prakken and Sartor's work is driven by their application in legal reasoning. To relate argumentation and extended logic programming, we review WFSX (Alferes & Pereira 1996), a semantics for extended logic programs. While the above fixpoint definition of justified, overruled, and defensible arguments is suitable to give a declarative semantics to argumentation we need additionally an efficient operational semantics. In particular, we need the top-down approach to avoid heavy recomputation when only deciding for or a against a conclusion or when generating a set of conflicting assumptions which lead to a contradiction. Concerning the top-down proof procedure defined in (Alferes & Pereira 1996) there is an important connection to argumentation. The distinction between undercut and rebut is not necessary in the proof procedure because the simple transformation of adding the default and strong negated *not* $\neg L$ to the body of a rule for L covers any rebut $\neg L$ against L by an undercut $\neg L$ against *not* $\neg L$.

Proposition 10 *Let $A_1 = L \leftarrow Body; r_n; \dots; r_m$ be an argument, then A_2 is a rebut against A_1 iff A_2 is an undercut against $A_1' = L \leftarrow Body, not\ \neg L; r_n; \dots; r_m$.*

With this proposition we can define the top-down proof procedure as follows.

Definition 11 $\models$
Let P be an extended logic program, then

$$\begin{aligned}
&P \models L \text{ iff } P, \emptyset, \emptyset, t \models L\\
&P, M \models L \text{ iff } P, \emptyset, \emptyset, M \models L\\
&P, LA, GA, M \models true\\
&P, LA, GA, M \models (L_1, L_2) \text{ iff}\\
&\qquad P, LA, GA, M \models L_1 \ \& \ P, LA, GA, M \models L_2
\end{aligned}$$

$$\begin{aligned}
&P, LA, GA, M \models not\ L \text{ iff}\\
&\qquad P, GA, GA, M \models \neg L \text{ or}\\
&\qquad M = t \ \& \ P, \emptyset, GA, tu \not\models L \text{ or}\\
&\qquad M = tu \ \& \ P, GA, GA, t \not\models L\\
&P, LA, GA, t \models L \text{ iff}\\
&\qquad L \notin LA \ \& \ L \leftarrow L_1, \dots, L_l, notL_{l+1}, \dots, notL_m \in P \ \&\\
&\qquad P, LA \cup \{L\}, GA \cup \{L\}, t \models L_1, \dots, L_l, notL_{l+1}, \dots, notL_m\\
&P, LA, GA, tu \models L \text{ iff}\\
&\qquad L \notin LA \ \& \ P, GA, GA, t \not\models \neg L \ \&\\
&\qquad L \leftarrow L_1, \dots, L_l, notL_{l+1}, \dots, notL_m \in P \ \&\\
&\qquad P, LA \cup \{L\}, GA \cup \{L\}, tu \models L_1, \dots, L_l, notL_{l+1}, \dots, notL_m
\end{aligned}$$

The inference operator has three parameters M, LA, and GA, where M is either t or tu indicating that we want to prove verity (t) and non-falsity (tu), respectively, LA and GA are lists of local and global ancestors that allow to detect negative and positive loops which lead to inference of non-falsity and failure, respectively; for details see (Alferes & Pereira 1996). For consistent programs the above inference operator yields the same results as the argumentation process (Schroeder, Móra, & Alferes 1997):

Proposition 12 *Relation of $\models$ and Argumentation*
Let P be consistent. $P,t \models L$, iff L is a conclusion from a justified argument. $P,t \models notL$, iff L is a conclusion from an overruled argument. $P,t \not\models L$ and $P,tu \models L$ iff L is a conclusion from a defensible argument.

To compute revisions, we define conflicts which are sets of default assumptions that lead to a contradiction and show how to solve the conflicts by changing the assumptions so that all conflicts are covered. Such a cover is called hitting set, since all conflicts involved are hit.

Definition 13 *Conflict*
Let P be an extended logic program with default literals D and revisables R. Then $C \subset D$ is a conflict *iff*

$$P \cup \{\neg c \mid not\ c \in C\} \cup \{c \mid c \in C\} \models \bot$$

and a positive subset $R' \subset not\ R$ of the defaults negated revisables R is called revision iff $P \cup R'$ has no conflicts.

Example 14 *Consider the Hamlet example above. There are two conflicts $\{not\ kills(hamlet, claudius)\}$ and $\{kills(hamlet, claudius)\}$ and no revisions. I.e. Hamlet is in the unfortunate situation that whether he kills Claudius or not, he is always caught in a conflict.*

To compute revisions we use REVISE 2.4 (Damásio, Pereira, & Schroeder 1997) an algorithm which adopts Reiter's hitting set algorithm (Reiter 1987; Greiner, Smith, & Wilkerson 1989) for contradiction removal in extended logic programs. The derivation procedure and contradiction removal algorithm are extended to generate proof traces which are then used for visualisation.

Having developed the argumentation theory we turn now to the functionality of the implemented Utlima Ratio (Schroeder, Plewe, & Raab 1998; 1999) system.

Reasoning Agents in Ultima Ratio

Similar to Hamlet, we formalised also Ilsa and Rick in Casablanca, Siegfried, Krimhild, Brunhild, Hagen and Etzel of the German saga Nibelungenlied, Euripides' Medea, Molière's Don Juan, the artist Duchamp and his ready-mades, Robocop, and Macchiavelli. As it turns out, many patterns occur in different settings. The same revenge motif governing Hamlet occurs also in the Nibelungenlied or in Don Juan; the betrayal rule is part of Medea and Don Juan; etc. Therefore, we grouped arguments not only according to their literature pieces, but defined agent properties such as revengeful, christian, aiming at happy love and marriage, rivaling, idealistic, offensive, opportunistic, and faithful.

In the Ultima Ratio project (Schroeder, Plewe, & Raab 1998; 1999), we organised these arguments in a database and implemented a system that allows the user to select a predefined agent such as Hamlet, or to construct one, say a revengeful christian, or to write one from scratch. These

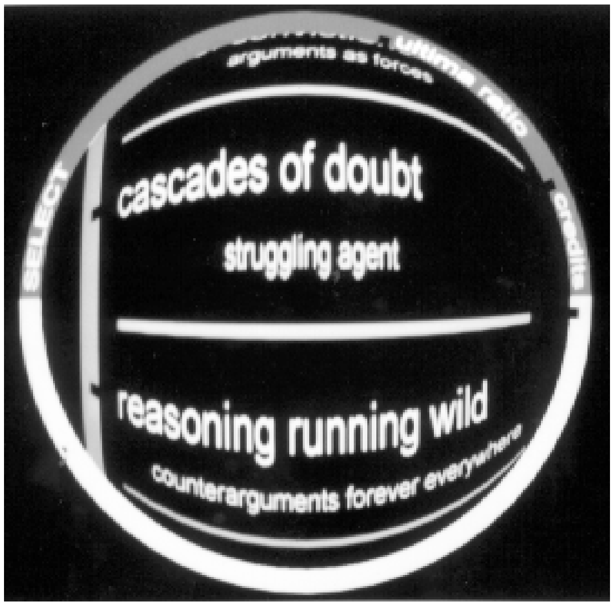

Figure 1: Compass-like User Interface of Ultima Ratio

agents just comprise arguments and assumptions, but no facts. Therefore there is not yet a conflict. But if the user additionally lets them face a world, i.e. a set of facts, the agents may be stuck in a conflict that needs to be resolved. Here the user can select another option. Either conflicts are only detected or also resolved (if possible). Alternatively, it is also possible to ask specific queries or use an inverse logic where all sensible arguements are negated.

As we already noted above, there is no solution to Hamlet's conflict, because by killing Claudius Hamlet sends him to heaven which is too good. But what if there is no heaven. Rather than Shakespearce's Hamlet we select a revengeful agent and let it face the same world as Hamlet where the king is dead and Claudius is praying. Not being christian, the revengeful agent does not hasitate and comes up with the solution of killing Claudius. Shakespearce's piece would have been considerably shorter and probably less bloody.

Ultima Ratio's Functionality

Figure 1 shows Ultima Ratio's graphical user interface whose form is derived from a compass. At the rim of the compass one can select menus which appear as a sphere. With a joystick the user rotates the sphere until the entry of choice is in the centre. Via the compass menu the user selects one of the following functions:

1. **Cascades of Doubt - Struggling Agents.** The user selects an agent according to the author's version. Currently the knowledge base contains Shakespearce's Hamlet, Ilsa and Rick in Casablanca, Siegfried, Krimhild,

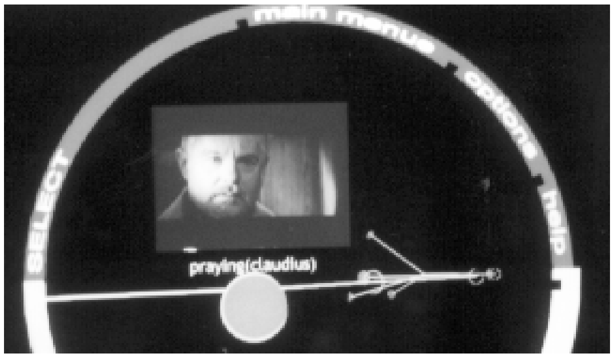

Figure 2: Cascades of Doubt - Struggling Agents: Should Hamlet take revenge on Claudius who is praying? In the foreground the orange node labelled *praying(claudius)* indicates the successful proof of this fact. Where possible conclusions are adorned with suitable video sequences. The background shows the rest of the proof tree.

Brunhild, Hagen and Etzel of the German saga Nibelungenlied, Euripides' Medea, Molière's Don Juan, the artist Duchamp and his readymades, Robocop, and Macchiavelli. The arguments for these agents are online.[1]

Consider Hamlet, for example. The argumentation process which is represented by a proof tree is visualised as a 3D tree. Premises which are not yet proven are represented by rotating circular rims. If they are successfully proven an orange disc moves to the rotating rim and fills it out (see Figure 2). If the proof failed the disc is blue. If the argumentation finally results in a conflict two rotating and intersecting ellipses appear at the conflicting nodes. If the potential conflict did not occur the ellipses do not intersect. The user can navigate in the argumentation space or switch to auto-pilot. For the auto-pilot, we combined top-down and bottom-up derivation. Technically, the system performs the proof top-down, but users felt more comfortable with a bottom-up proof starting with the facts and leading to a conflict in the end. We have combined both by the camera moving directly from the root of the proof tree to the leaves and then moving step by step bottom-up from the facts.

2. **Crossovers - Tracing Motifs.** Crossovers allow to trace motifs in the complete argumentation space. In the online knowledge base[2] one can see, for example, that the arguments for revenge occurring in Hamlet are also part of the Nibelungenlied and of the agent property "revengeful". Similarly, the topic of offences connects Don Juan to Medea. Visually, the tree structure of the arguments is shown and different regions in the 3D space correspond to different agents. The same argument occurring in different regions is connected by a bridge of grey cubes (see Figure 3) leading from one agent to another. Besides this 3D visualisation, the above online knowledge base provides an applet showing the dependencies of arguments in 2D.

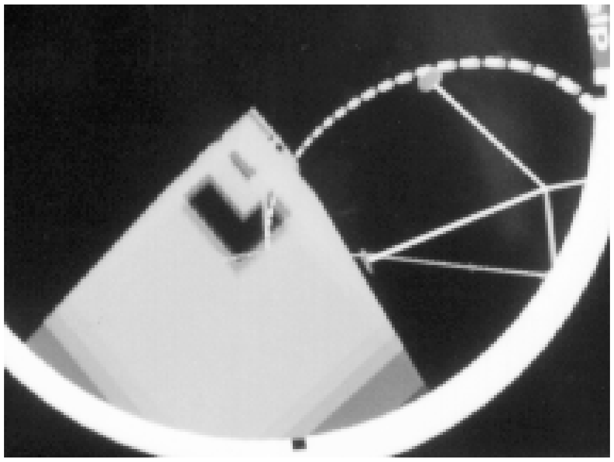

Figure 3: Crossovers - Tracing Motifs: Hamlet's rules for revenge are also part of Krimhild and Etzel, Agents of the Nibelungensaga. The cubic bridge in the foreground connects the arguments of both pieces one of which shown in the background.

3. **War of Convictions - Arguments as Forces.** The war of conviction function focuses on conflicts in general, independent of their instances for particular agents and worlds. The online knowledge base[3] lists all conflicts and arguments together with cross-references showing on which arguments conflicts are based and how arguments support and attack each other.

4. **Reasoning Running Wild.** In contrast to the above logical modes, Ultima Ratio also provides a visual metaphor for reasoning running wild: The proof trees move very fast so that their structure is not graspable anymore, additionally the user's head tracker is switched on leading to a distorted view depending on the user's head movements.

Conclusion and Future Work

At the center of most plots in literature is a main character, who is stuck in a conflict and considers different arguments and options to resolve the conflict. As we argued in section , human argumentation can not be convincingly formlised in its entirety, but many aspects of it can. To this extent, we showed how to employ extended logic

[1]http://www.soi.city.ac.uk/homes/msch/cgi/aec/kbwww/cod.html
[2]http://www.soi.city.ac.uk/homes/msch/cgi/aec/kbwww/co.html
[3]http://www.soi.city.ac.uk/homes/msch/cgi/aec/kbwww/woc.html

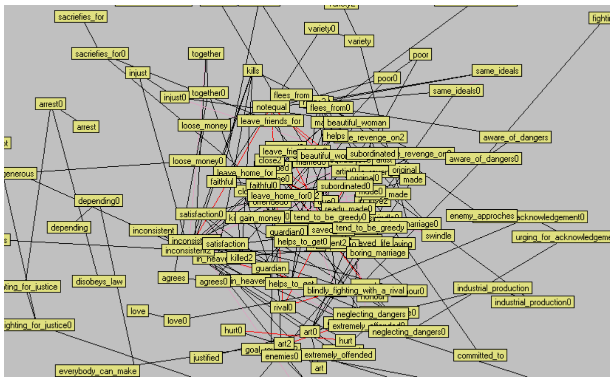

Figure 4: Crossovers - Tracing Motifs: A 2D dependency graph of all arguments.

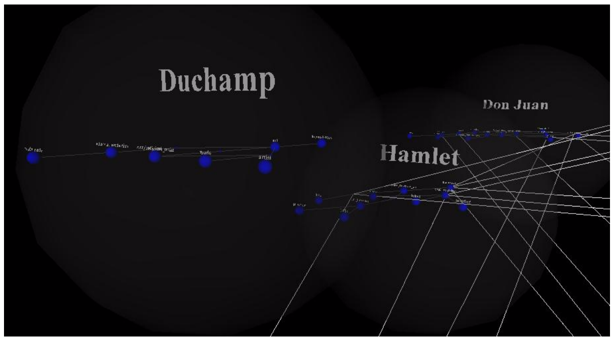

Figure 5: Crossovers - Tracing Motifs: Similar representation to Figure 3, however, implemented in VRML.

programming under well-founded semantics to model argumentation processes. In particular, we developed a declarative argumentation framework, which is equivalent to the operational top-down proof procedure introduced in definition 11. This equivalence is vital to cater for a goal-driven, and thus efficient, computation of an argumentation process. With the framework in place, we turned to the Ultima Ratio project (Schroeder, Plewe, & Raab 1998; 1999). The system allows one to select pre-defined agents (ranging from Hamlet to Robocop) or user-customised agents and a world. If the agents are stuck in a conflict an argumentation process starts to unfold the conflict and possibly remove it. The system has been exhibited at Ars Electronica, Linz, Austria, and the Canon Artlab, Tokyo, Japan.

In recent work (Schroeder 1999), we extended the framework to deal with multiple agents, which may be credulous or sceptical reasoners (see http://www.soi.city.ac.uk/homes/msch/cgi/aca). Finally, it would be interesting to extend our system to allow for interactive story telling as in (Hayes-Roth & van Gent 1997), for example.

Acknowledgement I am very much indebted to Daniela Plewe, who leads the Ultima Ratio project and developed and realised Ultima Ratio's concept, and Andreas Raab, who implemented most of the graphics.

References

Alferes, J. J., and Pereira, L. M. 1996. *Reasoning with Logic Programming*. (LNAI 1111), Springer-Verlag.

Copi, I. M., and Cohen, C. 1994. *Introduction to Logic*. Prentice Hall.

Damásio, C. V.; Pereira, L. M.; and Schroeder, M. 1997. REVISE: Logic programming and diagnosis. In *Proceedings of the Conference on Logic Programming and Nonmonotonic Reasoning LPNMR97*. LNAI 1265, Springer–Verlag.

Dung, P. M. 1993. An argumentation semantics for logic programming with explicit negation. In *Proc. of the 10th International Conference on Logic Programming*, 616–630. MIT Press.

Dung, P. M. 1995. On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. *Artificial Intelligence* 77(2):321–357.

Greiner, R.; Smith, B. A.; and Wilkerson, R. W. 1989. A correction of the algorithm in reiter's theory of diagnosis. *Artificial Intelligence* 41(1):79–88.

Hayes-Roth, B., and van Gent, R. 1997. Story-making with improvistional puppets. In *Proceedings of First Internationl Conference on Autonomous Agents, AA97*, 1–7. ACM Press.

Kowalski, R. A. 1995. Using meta-logic to reconcile reactive with rational agents. In Apt, K. R., and Turini, F., eds., *Meta-logics and Logic Programming*. The MIT Press. chapter 9, 227–242.

Kraus, S.; Sycara, K.; and Evenchik, A. 1998. Reaching agreements through argumentation: a logical model and implementation. *Artificial Intelligence*. To appear.

Prakken, H., and Sartor, G. 1997. Argument-based extended logig programming with defeasible priorities. *Journal of Applied Non-Classical Logics* 7(1).

Reiter, R. 1987. A theory of diagnosis from first principles. *Artificial Intelligence* 32(1):57–96.

Schroeder, M.; Móra, I.; and Alferes, J. J. 1997. Vivid agents arguing about distributed extended logic programs. In *Proceedings of the Portuguese Conference on Artificial Intelligence EPIA97*. LNAI 1323, Springer–Verlag.

Schroeder, M.; Plewe, D. A.; and Raab, A. 1998. Ultima Ratio - Should Hamlet kill Claudius? In *Proceedings of the second Conference on Autonomous Agents*. Minneapolis, USA: ACM Press.

Schroeder, M.; Plewe, D. A.; and Raab, A. 1999. Ultima Ratio - a visual language for argumentation. In *Proceedings of the International Conference on Information Visualisation*. London, UK: IEEE Press.

Schroeder, M. 1998. *Autonomous, Model-based Diagnosis Agents*. Kluwer Academic Publisher.

Schroeder, M. 1999. An efficient argumentation framework for negotiating autonomous agents. In *Proceedings of Modelling Autonomous Agents in a Multi-Agent World MAAMAW99*. LNAI1647, Springer-Verlag.

Sierra, C.; Jennings, N.; Noriega, P.; and Parsons, S. 1997. A framework for argumentation-based negotiation. In *Proc. Fourth Int. Workshop on Agent Theories, Architectures and Languages (ATAL-97)*, 167–182. Springer-Verlag.