Using Automated Planning to Obtain Extensions in AFs and Solve Credulous Acceptance of Arguments under Admissibility Semantics

Arturo González-Ferrer and Raquel Fuentetaja

Universidad Carlos III de Madrid
Spain

Abstract. This paper presents a novel Automated Planning solution for two argumentation problems: finding extensions of abstract argumentation frameworks; and credulous acceptance of arguments under admissibility semantics, providing a sequence of arguments to be included in the final extension set. In this paper we propose a PDDL planning model based on conditional effects and numeric functions. Then, we provide some preliminary results on the performance of this model on existing benchmark sets. We also discuss on the pros and cons that planning solvers may provide for argumentation.

1 Introduction and Background

The joint application of Automated Planning and Argumentation is not new. The work of Ferguson [10] might be the first that had real impact in the community. It raised the relevance of using arguments when two or more agents (e.g. a computer system and a human being) need to cooperate in order to decide what to include in a plan and to execute it. Each agent may have a knowledge base to take decisions. The knowledge in it can be defeasible, meaning that two arguments may conflict, therefore a preferred action needs to be selected. The proposed approach is to include defeasible knowledge about actions, preconditions and effects using rules of the form $\langle \Phi_i, p_i \rangle$ where each Φ_i is a set of *premises* and each p_i is a *conclusion* in a set Δ of argument steps that an agent uses for reasoning. So, arguments will support the fact that executing an action aims to achieve some goals, and they can be in conflict with or defeated by other arguments. When undefeated, related actions will take part of the solution plan.

Beyond the AI planning field, "Argumentation" has been progressing as an independent research area. Solutions based on SAT, CSP, ASP or dynamic programming [3] have been released recently to find consistent conclusions to so-called "abstract argumentation frameworks" [5] and formally reviewed in the next subsection. Informally, an abstract argumentation framework is a network of nodes where each node represents a possible argument, supporting some belief or action, and each edge represents an attack between two arguments, meaning that the arguments are confronted somehow. While this may seem as a simplistic representation of how agents would use such a network to argument among them, it is a challenging problem [7] to find subsets of nodes that can survive together to the attacks in the network, following some determined semantics [1]. These subsets are usually known as *extensions*.

Thus, the goal of our work is to present a model suitable for domain-independent automated planning solvers that aims to find extensions of abstract argumentation frameworks under *admissible semantics*, where the specification of some constraints can be asserted in the goal (e.g. number of arguments included or which arguments do we want to be included). Results will be shown about performance of our solution in different situations using existing benchmarks [4]. We will also discuss the knowledge representation challenges for Automated Planning found during our experiments and the pros and cons of our solution. The rest of the paper is organized as follows. Next we introduce a formal definition of AFs and admissibility semantics and the field of Automated Planning. Section 2 describes our planning model. Then, we present some preliminary results. Section 4 revises the related work; and finally we include some conclusions.

1.1 Abstract Argumentation Frameworks

Definition 1. *An abstract* **argumentation framework** *(AF) is a tuple $\langle S, R \rangle$ where S is a non-empty set of arguments and R is a binary relation on S representing $attacks$ between the arguments.*

The objective of an AF may be understood as identifying sets of justified arguments in S, based on the interactions represented by R and appropriate *semantics* that determine which subsets of S can be accepted as coherent. Such subsets are called *extensions*.

Definition 2. *Given an argumentation framework $\langle S, R \rangle$, an argument $x \in S$ is* **acceptable** *wrt. $E \subseteq A$ iff E defends x, i.e. $\forall y \in S$ such that $(y, x) \in R, \exists z \in E$ such that $(z, y) \in R$.*

Definition 3. *Given an argumentation framework $\langle S, R \rangle$, an extension $E \subseteq S$ is* **admissible** *iff (1) E is conflict-free, i.e. there is no attack between its components: $\forall\, x, y \in E, (x, y) \notin R$; and (2) all its arguments are acceptable arguments wrt. E.*

An argumentation semantics is the formal definition of a method (either declarative or procedural) ruling the argument evaluation process. An excellent introduction to well-known semantics existing in the literature is presented by Baroni et al. [1]. These semantics influence how the *justification* of an argument in the network takes place, meaning that it survives the attacks, either by self-defense or because any other argument in the solution defends it. We cope with admissibility (and so with conflict-freeness), which is a common base requirement for the rest of existing semantics.

Usual research problems addressed for AFs are to enumerate one or all the possible extensions of an AF under some determined semantics and to find out the *credulous* or *skeptical* acceptance of a concrete argument in a network under some determined semantics (i.e. an argument belongs to *at least one* or to *all* the extensions of a network, respectively). The aim of this paper is to explore how to use Automated Planning to enumerate one non-empty extension and to check credulous acceptance under admissible semantics, which has been shown to be a NP-complete problem [7], by finding a credulous-accepted solution.

1.2 Automated Planning

Automated Planning is an Artificial Intelligence area that develops mechanisms and techniques to choose and organize actions in order to achieve a set of goals from a given initial state [12]. Formally, a classical planning problem may be defined as a state transition system specified by the 3-tuple $\Sigma = (\mathcal{I}, \mathcal{A}, \mathcal{G})$, where $\mathcal{I}$ is the initial state of the world, represented explicitly by a set of instantiated predicates; $\mathcal{A}$ is a set of possibly applicable actions in a state, always that corresponding preconditions are fulfilled, producing some concrete effects; and $\mathcal{G}$ is the specification of a set of goals that the planner will try to reach through a deliberative search process. The output of this process is a plan, or sequence of actions, such that its consecutive application leads the system from the initial state to a state containing all goals.

The de facto standard for the representation of planning domains is the Planning Domain Definition Language (PDDL) [11]. Automated planners accepting PDDL have two inputs: the planning domain and the planning problem. The planning domain defines the domain actions. These actions are usually expressed in a general way by using variables and lifted predicates. The planning problem defines the specific facts of the initial state and the goals. The specification of a planning domain in PDDL implies the definition of a set of predicates and a set of domain actions, defined by their preconditions and effects. The great advantage of a representation language like PDDL is that the definition of the problem is separated from its resolution. In this way, with one encoding we can test different search algorithms and heuristics without any change to the implementation, just requiring the support of some features of the language by the planner. Additionally, PDDL separates the definition of the domain dynamics, i.e. the actions, from the specific problem to be solved, defined by the initial state and the set of goals. Domain actions are defined in a generic way by means of lifted predicates. Thus, the same domain definition can be used to solve any problem in this domain.

2 Proposed Model

We consider the problem of finding extensions as a search process modeled as a planning problem. The idea is to generate a plan that builds a set of arguments which is an admissible extension. That is: a) there are no conflicts and b) there are no unacceptable arguments. We also impose the requirement that c) the extension contains more than one element. This last requirement can be changed to any number, or could even be established as criteria to be minimized or maximized. Checking the credulous acceptance of an argument x, i.e. if x belongs to at least an extension, requires an additional constraint: d) the argument x belongs to the extension. These requirements are encoded as goals of the planning problem.

There are different possible PDDL encodings to model this problem. Some useful PDDL features [11,19,9] could be:

- **Numerical functions**: functions which value is a number instead of a truth value.
- **Derived predicates**: higher-level concepts derived from other predicates.
- **Conditional effects**: action effects that only occur under additional constraints.

– **Quantification in formulas and over effects**: existential/ universal quantified preconditions/effects.

Our first implementation considered using PDDL axioms through derived predicates. Derived predicates allow to define conditions on subsets and looking for goals that reach a certain type of subset. We only used derived predicates to specify in the problem goals the properties to be met by the extensions. For instance, to force admissibility we imposed reaching a state with `(admissible e)` in the goal. The domain contains two actions IN and OUT, to include and remove an argument from a set. The preconditions of these actions only check if the argument is/is not in the set.

While having a very intuitive specification[1] expressed as a nested hierarchy of definitions (i.e. the `admissible` axiom is defined in terms of the `acceptable` axiom, which is further defined in terms of the `conflict-free` axiom), the use of nested universal quantifiers for some definitions made the preprocessing time of the derived predicates to increase exponentially with the number of arguments in the network. We tested this domain using the FF-X planner [21] and a short number of arguments (up to 15). We observed that the preprocessing was unmanageable after this limit.[2]

Our final representation uses a different encoding based on numeric functions and conditional effects. Table 1 includes a description of the predicates and numeric functions used to model our domain, shown in Listing 1.1. The action `in(?x - arg ?s - set )` includes the argument $?x$ in the set $?s$, having a precondition to avoid considering the arguments ?x that do not affect the rest of the network (i.e. we can ignore arguments that do not attack or are attacked to/by some other in the network). This first precondition is similarly applied in other solvers. Then, including argument ?x produces five different blocks of **effects**, depending on several conditions (see Listing 1.1):

– Independently of any condition, the element $?x$ will become member of the extension and the number of elements will be increased by one (lines 6-8).
– For all argument $?y$ outside the extension, if $?x$ attacks $?y$ and the extension has not been marked as defending from $?y$, then it is marked as defending from $?y$: `(set-already-defends ?s ?y)` (lines 11-14).
– If the extension has already some members $?y$ and some attack exists between $?x$ and $?y$, then $?x$ and $?y$ are in conflict and number of conflicts should be increased (lines 16-20).
– Check all the nodes $?y$ outside the set that attack the incoming member $?x$ and this attack is not defended either by itself or any other node in the set, so that we anticipate the situation of unacceptability for $?x$. In that case, we increase the counter of unacceptable situations and mark $?x$ as unacceptable because of $?y$ (lines 22-27).
– If the node $?x$ that we are inserting attacks a node $?z$ that was causing another node $?y$ in the set to be unacceptable, we delete the predicate `(unacceptable ?y ?s ?z)` and decrease the counter of unacceptable situations. Note that we need to check this for any pair $?y, ?z$ using the universal quantifier (`forall`) over the conditional effect, otherwise we cannot guarantee the soundness of the solution (lines 29-33).

[1] See the specification through axioms at http://www.ugr.es/ arturogf/aetap/dom-axioms.pddl
[2] FF-X is one of the few planners planner supporting derived predicates.

Table 1. Description of the PDDL predicates and numeric functions.

PDDL predicates/numeric functions	Description
(attacks ?x - arg ?y - arg)	Argument ?x attacks argument ?y
(belongs-to-set ?x - arg ?s - set)	Argument ?x belongs to the extension set ?s
(set-already-defends ?s - set ?x - arg)	This predicate is activated when some argument(s) in set ?s already defends it from argument ?x, meaning that there is some argument in ?s that defends it from any attack of ?x
(unacceptable ?x - arg ?s - set ?y - arg)	The argument ?x is unacceptable in set ?s because argument ?y is attacking it and there is no defender in ?s, i.e. any argument in ?s attacks ?y
(num-conflicts ?s - set)	Numeric fluent expressing the number of conflicts in the set, i.e. the times that some arg1 attacks some arg2, both being in the set. It is increased when an IN action anticipates this happening.
(num-unacceptable ?s - set)	Numeric fluent expressing the number of attacks that make a situation for an argument unacceptable when included in the set. For example, if we include an argument with 3 attacking arguments without any defense from the set, this will be increased three times. Need to be decreased when the set self-defends from each attack.
(num-elements ?s - set)	Numeric fluent controlling the number of elements already in the set. It can be used to establish a minimum/maximum number of arguments in the extension.

Listing 1.1. PDDL encoding of the action IN.

```
(:action in
:parameters (?x - arg ?s - set)
:precondition (and
               (not (belongs-to-set ?x ?s))
               (exists (?j - arg) (or (attacks ?x ?j)(attacks ?j ?x))))
:effect (and
         (belongs-to-set ?x ?s)
         (increase (num-elements ?s) 1.0)

         ; mark defenses of ?x to ?y arguments
         (forall (?y - arg)
                 (when (and (attacks ?x ?y) (not (belongs-to-set ?y ?s))
                            (not (set-already-defends ?s ?y)))
                       (set-already-defends ?s ?y)))

         ; detect the breakage of conflict-freeness
         (when (and (>= (num-elements ?s) 1)
                    (exists (?y - arg) (and (belongs-to-set ?y ?s)
                                            (or (attacks ?x ?y) (attacks ?y ?x)))))
               (increase (num-conflicts ?s) 1.0))

         ; detect the incoming unacceptable situations
         (forall (?y - arg)
                 (when (and (attacks ?y ?x) (not (belongs-to-set ?y ?s))
                            (not (attacks ?x ?y)) (not (set-already-defends ?s ?y)))
                       (and (increase (num-unacceptable ?s) 1.0)
                            (unacceptable ?x ?s ?y))))

         ; solved unacceptance situations
         (forall (?y - arg ?z - arg)
         (when (and (unacceptable ?y ?s ?z)(attacks ?x ?z))
               (and (decrease (num-unacceptable ?s) 1.0)
                    (not (unacceptable ?y ?s ?z)))))))
```

Listing 1.2. PDDL encoding of the problem, including the goal.

```
(define (problem BSname)
(:domain ARGUMENT-TEST)
(:objects
        e - set
        arg0 arg1 arg2 arg3 ... arg149 - arg)
(:init
(= (num-elements e) 0)
(= (num-conflicts e) 0)
(= (num-unacceptable e) 0)
(argument arg0)
(attacks arg0 arg5)
(attacks arg0 arg4)
(argument arg1)
(attacks arg1 arg4)
...
(:goal  (and (= (num-conflicts e) 0)
             (= (num-unacceptable e) 0)
             (>= (num-elements e) 1))))
```

Listing 1.2 shows the encoding of a planning problem generated from a problem in the benchmark set. The predicates are also those introduced in Table 1. The initial state contains all counters initialized to zero. It also contains the existing arguments as objects, and the instances of the attack relationship as binary predicates. The extension e is initially empty, since there are not arguments belonging to it. We specify a goal where there are neither conflicts[3] nor unacceptable arguments, and the extension contains more than one element. Thus, this encoding represents the problem of *finding an admissible extension (e)* with more than one element. For the problem of *credulous acceptance* of one or several arguments $x_1, \ldots x_n$ we include in the goals additional facts to force the membership of these arguments to the set: $(\texttt{belongs-to-set}\ x_1, e), \ldots, (\texttt{belongs-to-set}\ x_n, e)$.

3 Experimental Results

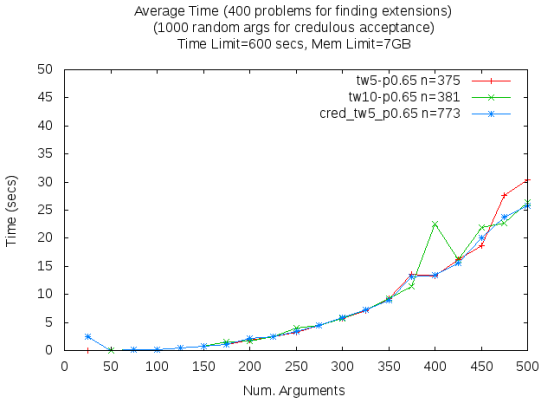
We present some preliminary experiments to evaluate the performance of our model for the two presented challenges. A deeper evaluation would include to compare with existing argumentation solvers, considering the results of the first International Competition on Computational Models of Argumentation (ICCMA)[4].

We selected the benchmark set 'benchmarks 1-2011' available in the DBAI website [4]. We transform problems automatically into the corresponding PDDL problem, that together with the domain, are used as inputs to the state-of-art Metric-FF planner[13].

All experiments were carried out on a PC with "Intel(R) Xeon(R) X3470 @ 2.93GHz" CPU with 8GB RAM, but limited to 600secs and 7GB of virtual memory.

[3] Note that the zero-conflicts is set as goal and not as precondition of the IN action because there are transition situations where conflicts must exist in order to generate a final solution later on that has no conflict.

[4] http://argumentationcompetition.org/2015/



(a)

	Metric-FF
N. FINISHED	**567** times
N. TIMEOUT	6 times
N. MEMOUT	427 times
FIN. AVGTIME	8.84 secs
TIME < 10 secs	352 times
Q1	2.26 secs
Q2	5.59 secs
Q3	14.865 secs

(b)

Fig. 1. (a) Average times for finding an extension (tw5-p0.65 and tw10-p0.65) and finding out credulous acceptance (tw5-p0.65); (b) Performance of Metric-FF planner for finding out the credulous acceptance of 1000 random arguments.

3.1 Finding Extensions

Benchmark sets were created with probabilities 0.4, 0.65 and 0.9, so we decided that 0.65 would be an intermediate value good enough to check our proof of concept. Regarding tree-width, it is a parameter of the graph that is relevant for the performance of some argumentation algorithms that work on tree decompositions of the graph. The benchmark set was generated for tree widths 3, 5, 7 and 10, so we selected 5 and 10 to see if it affects performance of our solver too. Results are shown in Fig. 1 (a). This figure shows the average time according to number of arguments for finding an extension with at least 2 elements for tw5-p0.65 and tw10-p0.65, using Metric-FF. Number n in the legend refers to the number of finished runs over the total problem instances of the benchmark set. We can see that performance keeps within reasonable times (less than 30 secs), all within the low memory limit of 7GB we imposed. The *soundness* of the solution is guaranteed by the planning domain and problem specifications themselves. During development of the model we developed a Bash script to check that the solutions provided are admissible, checking the solution against the input AF. The output of this script helped us to identify mistakes easily than using VAL plan validation tool [14] by analysing defend and attack situations of the arguments involved in the planning problem.

3.2 Finding Out Credulous Acceptance

For checking the credulous acceptance we randomly selected 1000 arguments of a random subset of 400 instances of tw3_p0.4, and checked the credulous acceptance of those arguments on those instances. Results in Fig. 1 (b) show the performance of Metric-FF planner in terms of the number of finished runs with a solution, the number of unfinished runs for time or memory reasons, the average time to those who finished,

the number of times that ended in less than 10 seconds, and the quartiles (Q1, Q2, Q3) of times for those who finished.

Metric-FF is able to return an admissible extension containing the random credulous-accepted argument 567 times. The efficient dynPARTIX solver [8] was further used to double-check the correctness (i.e. admissibility) of the solutions and to have a feeling of performance compare to a state of the art solver not based on planning. It returns a "YES" to credulous acceptance checking under admissible semantics 572 times. The 6 problems not solved by Metric-FF (TIMEOUT) are probably cases where we are trying to check an argument that is isolated in the network (does not attack or is attacked by anyone). As described in our model, we included the precondition to not consider these arguments for the IN action, so it is a problem derived from this situation, that could be easily fixed. We can also see how the performance evolves with the number of arguments of the network for the random subset of instances of tw5_p0.65 in Fig. 1 (a), where dynPARTIX solved 787 as credulous acceptance, while Metric-FF solved 773.

While the time to check credulous acceptance in dynPARTIX is almost zero, it did not return a solution, while our solver does. In conversation with dynPARTIX authors, checking credulous acceptance requires a bottom-up traversal of the AF's decomposition tree while returning a solution would require a second up-bottom traversal enumerating the solutions, requiring more time and memory. Nonetheless, the scope of this paper does not cover a full comparison with dynPARTIX or any other solver, and it is left for future work. It seems possible to combine both solutions, checking acceptance with traditional solvers and looking for solutions with a PDDL-based solver if the answer is YES. In such a case, as shown in Fig. 1 (b), the average time to find a solution is 8.84s, the 25% are found in less than 2.26s, 50% are found in less than 5.59s and 75% are found in less than 14.8s.

4 Related Work

Scant work has been directed to address the resolution of argumentation networks by means of Automated Planning. The work by E. Black et al. [2] presents a *conformant as classical planning* solution to find strategies that a persuader can use to convince another agent (non-adversarial persuasion dialogues). Authors look for a strategy where the set and the order of beliefs asserted by the persuader affects the success of this strategy, which is influenced by a probabilistic model that the persuader has about the responder's set of beliefs. The persuader asserts subsets of beliefs avoiding those that are part of a common knowledge base between both, avoiding the repetition of beliefs already asserted; the responder just answer yes (it finds acceptable the union of the asserted beliefs of the persuader with his own real subset) or no. In their example, authors consider grounded semantics.[5] Considering the possible strategies, authors calculate a probability of success for that strategy (i.e. what happens depending on which is the common KB), and use planning to look for an optimal strategy that maximizes such probability. A. Monteserin et al. [16,17] present a framework for negotiation-based planning, introducing the use of PDDL as possibility for reaching defined agreements.

[5] The smallest extension, with regard to set inclusion, of the complete extensions, i.e. those extensions E that are admissible and all acceptable arguments respect to E are included.

Authors propose to define several *create-argument* and *accept-argument* actions, where the initial state is a conflict situation while the goal is an expected agreement. I. A. Letia et al. [15] describe how to use a concept map for representing argument networks, and to interleave planning with arguing, replanning and using a preference relation or persuasion strategy to find solutions to the network. Informal and incomplete modeling of argument networks through PDDL is explored. The work of A. R. Panisson et al. [18] describes an HTN planning approach to multi-agent negotiation. Our work is directed to find extensions in AFs but could also be adapted to a multi-agent persuasion or agreement setting.

5 Discussion and Conclusions

We have presented a solution based on Automated Planning to find extensions and check credulous acceptance of arguments under admissibility semantics. Our model can be seen as a kind of magnet to not deviate from our goal, keeping the number of conflicts and unacceptability situations in zero. Our initial domain representation can be very easily extended to cope with ordering preferences or minimization of the number of steps. Also our model can be augmented to a a multi-agent planning approach where some agents try to reach a common goal or participate in a negotiation-based approach.

In contrast to usual argumentation solvers, our approach considers the logical structure of the argumentation process. This is interesing for analysing the way we reach the final extension set, i.e. the proof derivation [6]. Also, the order of arguments is important for finding strategies associated to dynamic disputes and negotiation in multi-agent settings, where a dialectical proof is needed [20]. In this sense, establishing a policy that, for example gives a better heuristic value to an argument defending more attacks, could reach a coherent ordered solution, reducing the number of steps. Another strategy could be to include arguments that try to defend firstly from the last attacks or from those having more strength. This can be interesting from a narrative or dialogue perspective. We plan to adapt our model to this type of problem in future work.

The limitations of planning for extension-related problems arise from its inner characteristics. Mainly the fact that it is aimed to provide one solution and stop. Therefore, it does not show like a very appropriate technique for extension enumeration. However, it could definitely help in argument networks where heuristic functions could help finding a goal faster and with a less memory footprint. Another problem where our model would work to verify that a concrete subset is an extension (extension verification), with just a simple modification of the planning problem goal. The exploration of heuristic search techniques for argumentation can benefit from using domain-independent automated planners.

6 Acknowledgements

This work has been partially supported by the Spanish project TIN2014-55637-C2-1-R.

References

1. Baroni, P., Caminada, M., Giacomin, M.: An introduction to argumentation semantics. Knowledge Engineering Review 36(4), 365–410 (2011)
2. Black, E., Coles, A., Bernardini, S.: Automated planning of simple persuasion dialogues. In: Computational Logic in Multi-Agent Systems, pp. 87–104. Springer (2014)
3. Charwat, G., Dvořák, W., Gaggl, S.A., Wallner, J.P., Woltran, S.: Methods for solving reasoning problems in abstract argumentation–a survey. Artificial Intelligence 220, 28–63 (2015)
4. DBAI: benchmark set. `http://www.dbai.tuwien.ac.at/research/project/argumentation/dynpartix/examples/benchmarks_1-2011.zip` (2015)
5. Dung, P.M.: On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. Artificial Intelligence 77(2), 321–357 (1995)
6. Dung, P.M., Thang, P.M.: A sound and complete dialectical proof procedure for sceptical preferred argumentation. In: LPNMR-Workshop on Argumentation and Nonmonotonic Reasoning. pp. 49–63 (2007)
7. Dunne, P.E., Wooldridge, M.: Complexity of abstract argumentation. In: Argumentation in Artificial Intelligence, pp. 85–104. Springer (2009)
8. Dvořák, W., Morak, M., Nopp, C., Woltran, S.: dynPARTIX-a dynamic programming reasoner for abstract argumentation. In: Applications of Declarative Programming and Knowledge Management, pp. 259–268. Springer (2013)
9. Edelkamp, S., Hoffmann, J.: PDDL2.2: The language for the classical part of the 4th international planning competition. 4th International Planning Competition, at ICAPS'04 (2004)
10. Ferguson, G., Allen, J.: Arguing about Plans: Plan Representation and Reasoning for Mixed-initiative Planning. In: AIPS. vol. 2010, pp. 43–48 (1994)
11. Fox, M., Long, D.: PDDL2.1: An extension to PDDL for expressing temporal planning domains. Journal of Artificial Intelligence Research 20, 61–124 (2003)
12. Ghallab, M., Nau, D., Traverso, P.: Automated planning: theory & practice. Elsevier (2004)
13. Hoffmann, J., Nebel, B.: The FF planning system: Fast plan generation through heuristic search. Journal of Artificial Intelligence Research 14, 253–302 (2001)
14. Howey, R., Long, D., Fox, M.: Val: Automatic plan validation, continuous effects and mixed initiative planning using PDDL. In: Tools with Artificial Intelligence, ICTAI. pp. 294–301 (2004)
15. Letia, I.A., Groza, A.: A Planning-Based Approach for Enacting World Wide Argument Web. In: Intelligent Distributed Computer Systems & Applications. pp. 137–146 (2008)
16. Monteserin, A., Amandi, A.: Argumentation-based negotiation planning for autonomous agents. Decision Support Systems 51(3), 532–548 (Jun 2011)
17. Monteserin, A., Berdún, L., Amandi, A.: Analysing the PDDL language for argumentation-based negotiation planning. In: Computational Science and Its Applications (ICCSA). pp. 698–713 (2012)
18. Panisson, A.R., Farias, G., Freitas, A., Meneguzzi, F., Vieira, R., Bordini, R.H.: Planning interactions for agents in argumentation-based negotiation. In: 11th Int. Workshop on Argumentation in Multi-Agent Systems (2014)
19. Pednault, E.P.: ADL: Exploring the middle ground between STRIPS and the situation calculus. In: Principles of Knowledge Representation and Reasoning. pp. 324–332 (1989)
20. Prakken, H.: Relating protocols for dynamic dispute with logics for defeasible argumentation. Synthese 127(1-2), 187–219 (2001)
21. Thiébaux, S., Hoffmann, J., Nebel, B.: In defense of PDDL axioms. Artificial Intelligence 168(1), 38–69 (2005)