Precompiled knowledge support for dynamic argumentation

Marcela Capobianco Carlos I. Chesñevar Guillermo R. Simari

Laboratorio de Investigación y Desarrollo en Inteligencia Artificial (LIDIA)
Departamento de Ciencias e Ingeniería de la Computación
Universidad Nacional del Sur - Bahía Blanca - ARGENTINA
e-mail: {mc,cic,grs}@cs.uns.edu.ar

Abstract

Argumentative formalisms have been widely recognized as knowledge representation and reasoning tools able to deal with incomplete and potentially contradictory information. All such formalisms are computationally demanding. Hence, optimizing argumentative systems has been approached from different views.

We have developed a new proposal to solve the aforementioned problem. The key to our approach consists in maintaining a module associated with the knowledge base of the system, with additional information that may help to speed up the inference process. As truth maintenance systems optimize general problem solvers, this module could play a similar role in argumentative systems.

Keywords: knowledge representation, argumentation, rational agents.

1 Introduction

Argumentative formalisms [16, 19, 20, 9, 17] have been widely recognized as knowledge representation and reasoning tools able to deal with incomplete and potentially contradictory information. The main obstacle to use these system in practical applications is that they are computationally demanding.

Optimizing argumentative systems has been approached from different views [5, 13]. The basic idea of our proposal is to maintain a module associated with the knowledge base of the system, with additional information that may help to speed up the inference process. As truth maintenance systems [8] optimize general problem solvers, this module could play a similar role in argumentative systems.

The use of precompiled knowledge for argumentatión formalism was previously addressed in [2]. In that article, the basic idea was to maintain a repository of the conclusions previously computed to avoid repeating the reasoning process on the same input. The disadvantages of that formulation were twofold. Firstly, its applicability was restricted as it only helped in the solution of recurrent queries. Secondly, the recorded information had to be updated each time the agent acquired a new observation, and the restoring operation was a demanding task.

This paper presents a new proposal that addresses those shortcomings. Basically, we redefine the inference mechanism of ODeLP [4] to take advantage of a precompiled knowledge component.

This approach was first considered by the authors in [3] and has undergone several refinements [1]. In this article we summarize the main results obtained through this work.

The remainder of this paper is organized as follows. Section 2 describes the basic ideas in argumentation and truth maintenance systems. Section 3 presents the language of the ODeLP formalism and describes the perception mechanisms devised for this system. Section 4 defines the inference mechanism for ODeLP, incorporating precompiled knowledge. Finally, section 5 states the main conclusions obtained through this work.

2 Preliminaries

This section surveys the basic ideas in argumentation and truth maintenance systems that will be used in the rest of this paper. Firstly, we present a general view on argumentation formalisms. Truth maintenance system are analyzed later according to Doyle's original proposal [8].

2.1 Argumentation

Argumentative systems [16, 19, 20, 9, 17] arose in the late eighties as knowledge representation and reasoning tools able to deal with incomplete information. In general, argumentative systems represent the world by means of a *knowledge base*. Conclusions are drawn from this knowledge base using an appropriate *inference mechanism*, defined accordingly.

Usually the knowledge base of an argumentative system allows to represent two different kinds of information: *strict* and *defeasible*. Strict information is assumed to be correct and irrefutable, while defeasible knowledge is tentative and can be subject to exceptions. Rules that encode the former are called *defeasible* or *weak*, and the rules modelling strict knowledge are called *strong* or *strict*.

In argument-based formalisms, the inference mechanism relies on the construction of arguments. In classical logic rules are chained to obtain a certain *derivation*, and in argumentative systems *arguments* are built to support a given conclusion. The information in the knowledge base (both strict and defeasible rules) is used to obtain arguments, which represent pieces of tentative reasoning.

Given an argumentative formalism, different arguments can be derived from its associated knowledge base. As the inference process is based on tentative knowledge, conflicting arguments can be obtained. In this situation some preference criterion is adopted to decide which argument prevails or *defeats* its opponent and therefore can support its conclusion. The comparison criterion as well as the definition of conflict between arguments are a particular element of every argumentation system and may vary accordingly.

Using the notion of *defeat* among arguments it can be defined when a given argument is *acceptable*, *i.e.*, is able to survive every conflict with other arguments. In a first approach, the system may deem an argument as acceptable if it can defeat every one of its opponents. Nevertheless, suppose that an argument A is defeated by an argument B which in turn is defeated by a third argument C. In this case the argument A may be acceptable, as B does not have the force to eliminate A (since B is a defeated). This gives raise to a recursive definition of acceptability. Once again, there is no agreement about this notion, and its formalization depends on the particular system under consideration.

In the past few years numerous contributions to the field of argumentation have been developed. These works have acknowledged defeasible argumentation as a powerful tool for knowledge representation and common-sense reasoning. Besides, many of applications have been

implemented based on this discipline, mainly in the areas of agent negotiation and argument-based decision making.

2.2 Precompiled Knowledge

John Doyle pioneered in the formulation and use of precompiled knowledge. He defined the concept of *Truth Maintenance Systems* (TMS) [8] as tools for problems solvers. The function of a TMS is to record and maintain the reasons for the agent's beliefs. Doyle describes a series of procedures that determine the current set of beliefs and update it in accord with new reasons. Under this view, *rational thought* is deemed as the process of finding reasons for attitudes [8]; that is to say, some attitude (such as belief, desire, etc) is rational if there is some acceptable explanation for holding it. Next, we describe the basic terminology of Doyle's TMS.

TMS record arguments [1] for potential program beliefs. It manipulates two basic data structures: *nodes*, which represent beliefs, and *justifications* which model reasons for the nodes. We say that the TMS *believes* in a node if it has an argument for the node and believes in the nodes involved in the argument. Although this may seem circular, there are arguments which involve no other nodes (these are considered as assumptions).

The fundamental actions of a TMS are:

- create a new node, to which the problem solving program using the TMS can attach the statement of a belief.
- add (or retract) a justification for a node, to represent a step of an argument for the belief represented by the node.
- mark a node as a contradiction, to represent the inconsistency of any set of beliefs which enters into an argument for the node.

A new justification for a node may lead the TMS to believe in it. If the TMS did not believe in the node previously, this may in turn allow other nodes to be believed by virtue of existing but incomplete arguments. In this case, a *truth maintenance procedure* [8] is invoked, to make any necessary revisions in the set of beliefs.

TMS employ a particular type of justifications, called *non-monotonic*, to make tentative guesses. A non-monotonic justification bases an argument for a node not only on current beliefs in certain nodes, but also on lack of beliefs in other nodes. Any node supported by a non-monotonic justification is called an *assumption*.

Every node in the TMS has an associated set of justifications. Each justification represents a different reason for asserting it. The node is believed if and only if at least one of the justifications is *valid*.[2] In this case it is say to be *in* the set of beliefs. Otherwise, the node is *out* of this set. It is important to remark that the distinction between *in* and *out* is not that between *true* and *false*. The former classification refers to current possession of valid reason for belief, where as *true* and *false* are the result of evaluating inferences according to their truth value, independently of any reason.

In the TMS, each potential belief to be used as a hypothesis or a conclusion of an argument must be given its own distinct node. When uncertainty about some inference P exists, nodes for both P and its negation must be provided. Either of these nodes can have or lack well-founded arguments, leading to a four-element belief set (neither P nor $\neg P$ are believed, exactly one is

[1] It is important to note that from section 3 onward the term argument is used with a different meaning

[2] See [8] for a precise definition.

believed, or both are believed). The author details the procedures needed to establish the state of every node, and to update these states in case new justifications or facts are added to the TMS.

As advantages of the TMS, Doyle mentions that the system solves part of the belief revision problem and provides a mechanism for making non-monotonic assumptions. He also affirms that performance is significantly improved. In his own words:

> ... the overhead required to record justifications for every program belief might seem excessive. However, the pressing issue is not the expense of keeping records of the sources of beliefs. Rather, we must consider the expense of not keeping these records. If we throw away information about derivations, we may be condemning ourselves to continually re-deriving information in large searches caused by changing irrelevant assumptions...

Doyle's truth maintenance system motivated the development of a large body of literature [6, 7, 10, 15, 11]. The seminal idea appears not to have been any particular technical mechanism, but rather the general concept of an independent module for belief maintenance [15]. Moreover, TMS have been implemented and used in practical applications with satisfying results.

3 ODeLP: Observation-based DeLP

The language of ODeLP is based on the formalism of DeLP [13, 12]. Therefore, it inherits its advantages as a knowledge representation and reasoning tool. ODeLP also incorporates mechanism for perception that make it more suitable for dynamic domains, where modifying the knowledge base in terms of new information is a mandatory issue. In what follows, we briefly define the language of ODeLP.

Observations and *defeasible rules* are the main components of ODeLP language. *Observations* describe the knowledge of agent about the world and *defeasible rules* represent ways of extending observations with tentative information (*i.e.* information that can be used if nothing is posed against it). Following we present the corresponding formal definitions.

Definition 3.1. (*Signature*)
A signature Σ is an ordered tuple denoted as $(\mathcal{V}, \mathit{Pred}, \mathit{Func})$, where $\mathcal{V}$ is an enumerable set of variables, *Pred* is a finite set of predicates and *Func* is a finite set of functions such that $\mathcal{V} \cap (\mathit{Pred} \cup \mathit{Func}) = \emptyset$. ■

The concepts of function, predicate and variable are defined as usual. Literals are considered as atoms that may be preceded by the symbol "$\sim$" denoting strong negation [14].

Definition 3.2. (*Observation*)
An *observation* is a ground literal. ■

Definition 3.3. (*Defeasible rule*)
A *defeasible rule* is an ordered pair, denoted as "`Head` $\prec$ `Body`", where `Head` is a literal and `Body` is a non-empty finite set of literals such that intersecting the variables in the literals of `Head` and `Body` results in a non-empty set. ■

Defeasible rules are not elements of the object language in ODeLP. They play the role of inference rules for the inference mechanism. Therefore they must contain variables in order to describe a general rule of the domain under consideration. A defeasible rule with ground

literals codifies only a particular situation that should not be expressed with a defeasible rule. It must be remarked that variables appearing with the same name in the head and the body of a given defeasible rule are regarded as the same object. Every defeasible rule is in fact an scheme representing a set of ground instances.

Definition 3.4. (*Ground instance of a defeasible rule*)
Let r be a defeasible rule. A *ground instance* r_1 of r is obtained by replacing every variable in r with a constant of the language such that variables with the same name are replaced by the same constant. ■

Definition 3.5. (*Defeasible logic program*)
A *defeasible logic program* is composed by set Ψ of observations and a set Δ of defeasible rules such that:

1. Ψ and Δ are finite sets.
2. Ψ is a non-contradictory set of literals (*i.e.* it is not the case that $p \in \Psi$ and $\sim p \in \Psi$).

When required, we denote $\mathcal{P}$ as (Ψ, Δ). ■

The second restriction in the definition above demands a kind of internal coherence in the set of premises (observations). This condition is also used in the formalism of Simari and Loui [19] and DeLP [13]. In these systems the set of strict knowledge is required to be non-contradictory.

Example 3.1. The following rules codify an ODeLP program holding information about a given company and its workers.

```
poor_performance(john).
sick(john).
poor_performance(peter).
high_salary(rose).
applicant(rose).
good_referencies(rose).
hire(X) -< low_salary(X), applicant(X).
~hire(X) -< high_salary(X), applicant(X).
hire(X) -< high_salary(X), applicant(X), good_referencies(X).
suspend(X) -< ~responsible(X).
~suspend(X) -< responsible(X).
~responsible(X) -< poor_performance(X).
responsible(X) -< good_performance(X).
responsible(X) -< poor_performance(X), sick(X).
```

Observations describe that John has a poor performance at his job, John is currently sick, Peter also has a poor performance and Rose is an applicant that demands a high salary. Defeasible rules express that the company prefers to hire employers that require a low salary. Besides, an employee that demands a high salary is usually not hired, but in the exceptional case where he/she has good references it is recommended to hire the applicant. The remaining rules deal with the performance of the employees. If a given person is not responsible in his/her job then he/she should usually be suspended, a responsible person should not be suspended, and a person is hold as responsible (or not responsible) considering his/her performance in the company. ■

It is interesting to remark the differences between the language of ODeLP and its predecessor DeLP[13]. In DeLP programs are composed by a set of strict rules and a set of defeasible rules. In contrast, ODeLP does not include strict rules as part of the knowledge base. This new approach is particularly useful for practical applications of the system, since it makes easier to implement perception mechanisms in ODeLP. In addition, despite the elimination of strict rules, the formalism of ODeLP does not lack expressive power and is suitable for a wide range of applications.

4 An inference engine for ODeLP

Having defined the concept of defeasible logic programs, we focus on the *consequence operator* for these programs. Inference in ODeLP is driven by *defeasible queries*. A defeasible query (from now on a query) is defined in a logic programming style using defeasible rules as Horn-like clauses. The first step to answer a given query is to construct a *defeasible derivation* for it.

Definition 4.1. (*Defeasible derivation*)
Let $\mathcal{P} = (\Psi, \Delta)$ be an ODeLP program and let q be a ground literal (also called query). A finite sequence of ground literals, $s = q_1, q_2, \ldots, q_{n-1}, q$, is said to be a *defeasible derivation* for q from $\mathcal{P}$ (abbreviated $\mathcal{P} \mathrel{|\!\sim} q$) if for every q_i there exists a defeasible rule $r \in \Delta$ and a ground instance t of r, $t = q_i \prec l_1, \ldots, l_m$, such that $l_1, \ldots, l_m$ are ground literals previously occurring in s. ■

Even though the set Ψ must be non-contradictory, $\mathcal{P}$ may allow the defeasible derivation of complementary literals. Therefore new mechanisms are explored to find out which literals are supported by the program. This leads to the concept of *argument*.

Definition 4.2. (*Argument – Sub-argument*)
Given an ODeLP program $\mathcal{P}$, an *argument* $\mathcal{A}$ for a ground literal q, also denoted $\langle \mathcal{A}, q \rangle$ or simply $\mathcal{A}$, is a subset of ground instances of the defeasible rules in $\mathcal{P}$ such that: (1) there exists a defeasible derivation for q from $\Psi \cup \mathcal{A}$, (2) $\Psi \cup \mathcal{A}$ is non-contradictory, and (3) $\nexists\, \mathcal{A}' \subset \mathcal{A}$ such that $\Psi \cup \mathcal{A}' \mathrel{|\!\sim} q$. An argument $\langle \mathcal{A}_1, q_1 \rangle$ is a *sub-argument* of $\langle \mathcal{A}_2, q_2 \rangle$ if $\mathcal{A}_1 \subseteq \mathcal{A}_2$. ■

If $\mathcal{A}$ is a set of defeasible rules, then $heads(\mathcal{A})$ (respectively $bodies(\mathcal{A})$) denotes the literals occurring in the head (respectively body) of a defeasible rule in $\mathcal{A}$, $literals(\mathcal{A})$ denotes the set composed by $heads(\mathcal{A}) \cup bodies(\mathcal{A})$, and the set $\mathcal{G}(\mathcal{A}) = bodies(\mathcal{A}) - heads(\mathcal{A})$ is called the *ground of* $\mathcal{A}$.
An argument $\mathcal{A}$ for a query q provides a tentative proof for q, which may be in conflict with other arguments that contradict $\mathcal{A}$ (in the example above $\mathcal{A}_2$ contradicts $\mathcal{A}_3$). This leads to the formal definition of *counter-argument*.

Definition 4.3. (*Counter-argument*)
An argument $\langle \mathcal{A}_1, q_1 \rangle$ *counter-argues* an argument $\langle \mathcal{A}_2, q_2 \rangle$ (or $\langle \mathcal{A}_1, q_1 \rangle$ is a *counter-argument* for $\langle \mathcal{A}_2, q_2 \rangle$) at a literal q if and only if there is a sub-argument $\langle \mathcal{A}, q \rangle$ of $\langle \mathcal{A}_2, q_2 \rangle$ such that q_1 and q are not contradictory literals. ■

An argument $\langle \mathcal{A}_1, q_1 \rangle$ *defeats* an argument $\langle \mathcal{A}_2, q_2 \rangle$ if $\langle \mathcal{A}_1, q_1 \rangle$ is a counter-argument for $\langle \mathcal{A}_2, q_2 \rangle$, and (1) $\langle \mathcal{A}_1, q_1 \rangle$ is deemed better than $\langle \mathcal{A}_2, q_2 \rangle$ according to some preference criterion or (2) $\langle \mathcal{A}_1, q_1 \rangle$ is unrelated to $\langle \mathcal{A}_2, q_2 \rangle$ using this criterion. The former case is called *proper defeat* and the latter *blocking defeat*. In this paper we adopt *specificity* as the preference

criterion (as defined in [4]), although any criterion that induces a partial order on the set of possible arguments could be used.

A query q will succeed if there exists a supporting argument $\mathcal{A}$ for q such that $\mathcal{A}$ is ultimately undefeated. In that case $\mathcal{A}$ is said to be a *warrant* for q (or equivalently, q is *warranted*). To establish whether an argument $\langle \mathcal{A}, q \rangle$ is ultimately undefeated, all possible defeaters for $\langle \mathcal{A}, q \rangle$ are analyzed.[3] Since defeaters are arguments, there may be defeaters for the defeaters and so on. For this reason, a complete dialectical analysis is required to determine whether the original argument is accepted. This analysis can be formalized through the construction and marking of a structure called *dialectical tree*. As an output of the marking process, undefeated arguments are labelled as `U-nodes` and defeated ones as `D-nodes`.

Definition 4.4. (*Dialectical tree*)
Let $\mathcal{A}$ be an argument for q. A *dialectical tree* for $\langle A, q \rangle$, denoted $\mathcal{T}_{\langle \mathcal{A}, q \rangle}$, is recursively defined as follows:

1. A single node labeled with an argument $\langle A, q \rangle$ with no defeaters (proper or blocking) is by itself the dialectical tree for $\langle A, q \rangle$.

2. Let $\langle \langle A, \rangle 1, q_1 \rangle, \langle \mathcal{A}_2, q_2 \rangle, \ldots, \langle \mathcal{A}_n, q_n \rangle$ be all the defeaters (proper or blocking) for $\langle A, q \rangle$. The dialectical tree for $\langle A, q \rangle$, $\mathcal{T}_{\langle \mathcal{A}, q \rangle}$, is obtained by labelling the root node with $\langle A, q \rangle$, and making this node the parent of the roots nodes for the dialectical trees corresponding to $\langle \mathcal{A}_1, q_1 \rangle, \langle \mathcal{A}_2, q_2 \rangle, \ldots, \langle \mathcal{A}_n, q_n \rangle$. ■

Every dialectical tree $\mathcal{T}_{\langle \mathcal{A}_1, q_1 \rangle}$ is marked according to the following criterion:

- all the leaves in $\mathcal{T}_{\langle \mathcal{A}_1, q_1 \rangle}$ must be marked as `U-nodes`, and

- Every $\langle \mathcal{A}_2, q_2 \rangle$ such that $\langle \mathcal{A}_2, q_2 \rangle$ is an inner node of $\mathcal{T}_{\langle \mathcal{A}_1, q_1 \rangle}$, $\langle \mathcal{A}_2, q_2 \rangle$ must be marked as `U-node` if and only if every child of $\langle \mathcal{A}_2, q_2 \rangle$ is marked as a `D-node`; otherwise $\langle \mathcal{A}_2, q_2 \rangle$ must be marked as a `D-node`.

It is worth noticing that dialectical trees may give rise to contradictory or circular argumentation, which are particular cases of the so-called *fallacious argumentation* [18], a problem suffered by most argumentation systems. These situations are considered and solved in ODeLP as special mechanisms are designed to cope with such problems [3]. For space reasons, these concepts are not considered in this paper.

Definition 4.5. (*Warrant*)
Let $\mathcal{A}$ be an argument for a literal q, and let $\mathcal{T}_{\langle \mathcal{A}, q \rangle}$ be its associated dialectical tree. $\mathcal{A}$ is a *warrant* for q if and only if the root of $\mathcal{T}_{\langle \mathcal{A}, q \rangle}$ is marked as a `U-node`. ■

Example 4.1. If the query `suspend(john)` is formulated with respect to the program in example 3.1, the dialectical tree in figure 1 will be built, based on arguments:

- $\langle \mathcal{A}, \texttt{suspend(john)} \rangle$, where
  $\mathcal{A} = \{\texttt{suspend(john)} \prec \sim\texttt{responsible(john)};$
  $\sim\texttt{responsible(john)} \prec \texttt{poor\_performance(john)}\}$.

- $\langle \mathcal{B}, \sim\texttt{suspend(john)} \rangle$, where
  $\mathcal{B} = \{\sim\texttt{suspend(john)} \prec \texttt{responsible(john)};$
  $\texttt{responsible(john)} \prec \texttt{poor\_performance(john),sick(john)}\}$.

[3] The search space can be reduced by applying an $\alpha - \beta$ pruning strategy [18].

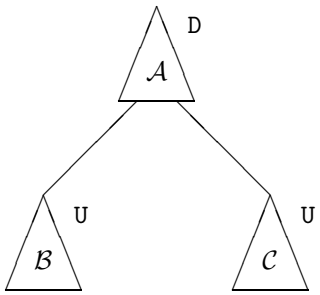

Figure 1: Dialectical tree from example 4.1.

- $\langle \mathcal{C}, \texttt{responsible(john)} \rangle$, where
  $\mathcal{C} = \{\texttt{responsible(john)} \prec \texttt{poor\_performance(john),sick(john)}\}$.

Using specificity as the preference criterion, $\mathcal{B}$ is a blocking defeater for $\mathcal{A}$, and $\mathcal{C}$ is a proper defeater for $\mathcal{A}$. Therefore argument $\mathcal{A}$ is a `D-node` and `suspend(john)` is not warranted. ■

This concludes the description of ODeLPś traditional inference mechanism. It is important to mention that this system has several desirable properties. For example, it has been shown that the inference relation of ODeLP is *coherent* in the sense that a literal and its complement cannot be inferred from the same knowledge base. For a full discussion of ODeLP's properties, the interested reader can consult [1].

The basic inference mechanism described so far may be reformulated by means of precompiled knowledge, using a module with additional information about the program that may be used in the inference process. This would improve the performance of ODeLP as a tool for solving real world problems as general problem solving are benefited by truth maintenance systems.

There are different ways to construct the module of precompiled knowledge. A simple approach could rely on keeping track of every argument that can be built from the program and the defeat relation among them. This proposal has several drawbacks. In the first place, a relatively small program may produce a large number of arguments, resulting a large repository of information. Besides, some arguments may be obtained using different ground instances of the same defeasible rule. This leads to believe that we could simply record the defeasible rules that were used to obtain the argument. Following this line of reasoning we defined the notion of *potential arguments*, cornerstone of the precompiled knowledge in ODeLP.

Definition 4.6. (*Instance for a set of defeasible rules*)
Let A be a set of defeasible rules. A set $\mathcal{B}$ formed by ground instances of the defeasible rules in A is an *instance of* A if and only if every instance of a defeasible rule in $\mathcal{B}$ is an instance of a defeasible rule in A. ■

Definition 4.7. (*Potential argument*)
Let Δ be a set of defeasible rules. A subset A of Δ is a *potential argument* for a literal q, noted as $[\![A, q]\!]$ if there exists a non-contradictory set of literals ϕ and an instance $\mathcal{B}$ of the rules in A such that $\mathcal{B}$ is an argument with respect to (Φ, Δ). ■

In the definition above the set of observations ϕ represents a state of the world in which it is possible to build the instance $\mathcal{B}$. It is important to remark that a contradictory set of literals includes at least a pair of complementary literals. The set ϕ must be non-contradictory to model a coherent scenario.

Note that potential arguments are independent from the current set of observations Ψ. This stems from the fact that observations may change as the world evolves. ODeLP has been designed to accommodate perceptions mechanisms in a simple way and defining potential argument independent from the observations respects this philosophy. Otherwise, potential arguments would change as the agent perceives new observations, decreasing the efficiency of the system.

The first step for building the precompiled knowledge associated with an ODeLP program $\mathcal{P} = (\Psi, \Delta)$ is to produce and record every potential argument of $\mathcal{P}$. We have developed a set of algorithms for this task, not detailed in this paper for space reasons.

Potential arguments can be used in the inference process in the following way. Instead of generating the arguments for a given query q, the system finds the potential argument such that at least one of its instances may be an argument for q. If we record the information about the defeat relation in the module of precompiled knowledge, it can also be used for find the defeaters of a given argument and thus construct the dialectical tree. Therefore we extend the concepts of counterargument and defeat for potential arguments. A potential argument $[\![A_1, q_1]\!]$ *counter-argues* another potential argument $[\![A_2, q_2]\!]$ at a literal q if and only if there is a potential sub-argument $[\![A, q]\!]$ of $[\![A_2, q_2]\!]$ such that q_1 and q are contradictory literals. Note that this kind of counter-arguments represents only a possible conflict between the potential arguments in contest. It might be the case that these arguments cannot co-exist in any scenario (*e.g.* consider two potential arguments based on contradictory observations).

To check whether a potential argument defeats one of its counter-arguments, the criterion used to compare pairs of arguments must be adapted to pairs of potential arguments. In particular, we have redefined specificity to compare arguments independently from the set of observations.[4] The definition of defeat between hypothetical arguments can be stated analogously to the standard definition of defeat, parameterized with respect to the defeat criterion.

Having defined the preliminary notions, we can introduce the concept of *dialectical bases*, the structure that represents the module of precompiled knowledge for a given ODeLP program.

Definition 4.8. (*Dialectical Base*)
Let $\mathcal{P} = (\Psi, \Delta)$ be an ODeLP program. The tuple $(\mathbb{A}, D_p, D_b)$ is said to be the *dialectical base* of $\mathcal{P}$ with respect to Δ, denoted as $\mathcal{DB}_\Delta$, if and only if: (1) $\mathbb{A}$ is the set formed by every potential argument that can be built from Δ; and (2) D_p and D_b are relations over the elements of $\mathbb{A}$ such that for every pair $([\![A_1, q_1]\!], [\![A_2, q_2]\!])$ in D_p (respectively D_b) it holds that $[\![A_2, q_2]\!]$ is a proper (respectively blocking) defeater of $[\![A_1, q_1]\!]$. ■

To build the dialectical base of an ODeLP program the inference engine first generates the set $\mathbb{A}$ of potential arguments and then calculates the associated defeat relation. [5] The modified inference process selects the potential arguments that could be used to support the main query. Next, selected every potential argument is instanciated accordingly and the links to the defeaters are used to continue the dialectical analysis that leads to the dialectical tree for the given query.

Example 4.2. Consider the program in example 3.1. The dialectical base of $\mathcal{P}$ is composed by the following potential arguments:

- $[\![A_1, \texttt{hire(X)}]\!]$,
  where $A_1 = \{\texttt{hire(X)} \prec \texttt{low\_salary(X), applicant(X)}\}$.

[4] See [4] for the definition and analysis of this criterion.

[5] We have developed algorithms for the creation and use of dialectical bases that for space reasons are not presented in this paper.

- $[\![A_2, \sim\texttt{hire(X)}]\!]$,
  where $A_2 = \{\sim\texttt{hire(X)} \prec \texttt{high\_salary(X), applicant(X)}\}$.
- $[\![A_3, \texttt{hire(X)}]\!]$,
  where $A_3 = \{\texttt{hire(X)} \prec \texttt{high\_salary(X), applicant(X), good\_refencies(X)}\}$.
- $[\![B_1, \texttt{suspend(X)}]\!]$,
  where $B_1 = \{\texttt{suspend(X)} \prec \sim\texttt{responsible(X)}\}$.
- $[\![B_2, \texttt{suspend(X)}]\!]$,
  where $B_2 = \{\texttt{suspend(X)} \prec \sim\texttt{responsible(X)};$
  $\sim\texttt{responsible(X)} \prec \texttt{poor\_performance(X)}\}$.
- $[\![B_3, \sim\texttt{suspend(X)}]\!]$,
  where $B_3 = \{\sim\texttt{suspend(X)} \prec \texttt{responsible(X)}\}$.
- $[\![B_4, \sim\texttt{suspend(X)}]\!]$,
  where $B_4 = \{\sim\texttt{suspend(X)} \prec \texttt{responsible(X)};$
  $\texttt{responsible(X)} \prec \texttt{good\_performance(X)}\}$.
- $[\![B_5, \sim\texttt{suspend(X)}]\!]$,
  where $B_5 = \{\sim\texttt{suspend(X)} \prec \texttt{responsible(X)};$
  $\texttt{responsible(X)} \prec \texttt{poor\_performance(X), sick(X)}\}$.
- $[\![C_1, \texttt{responsible(X)}]\!]$,
  where $C_1 = \{\texttt{responsible(X)} \prec \texttt{poor\_performance(X)}\}$.
- $[\![C_2, \sim\texttt{responsible(X)}]\!]$,
  where $C_2 = \{\sim\texttt{responsible(X)} \prec \texttt{poor\_performance(X)}\}$.
- $[\![C_3, \texttt{responsible(X)}]\!]$,
  where $C_3 = \{\texttt{responsible(X)} \prec \texttt{poor\_performance(X), sick(X)}\}$.

and the defeat relations:

- $D_p = \{(A_3, A_2), (C_3, C_2), (C_3, B_2)\}$
- $D_b = \{(A_1, A_2), (A_2, A_1), (C_1, C_2), (C_2, C_1), (C_1, B_2), (C_2, B_4), (B_1, B_3), (B_1, B_4),$
  $(B_3, B_1), (B_4, B_1), (B_1, B_5), (B_5, B_1), (B_2, B_5), (B_5, B_2), (B_2, B_3), (B_3, B_2)\}$.

The relations are also depicted in figure 2, where A_1 properly defeats A_2 is indicated with an arrow from A_1 to A_2 and blocking defeat is distinguished with a dotted arrow.
Is is interesting to note that a dialectical base is a graph and as such stands for several dialectical trees. If the query `hire(rose)` is posed to the system, this can be solved using the potential argument A_3 (resulting into a dialectical tree formed only by one argument, or using the potential argument A_1 and resulting into a dialectical tree with three chained arguments. The dialectical base contains both alternatives trees. ■

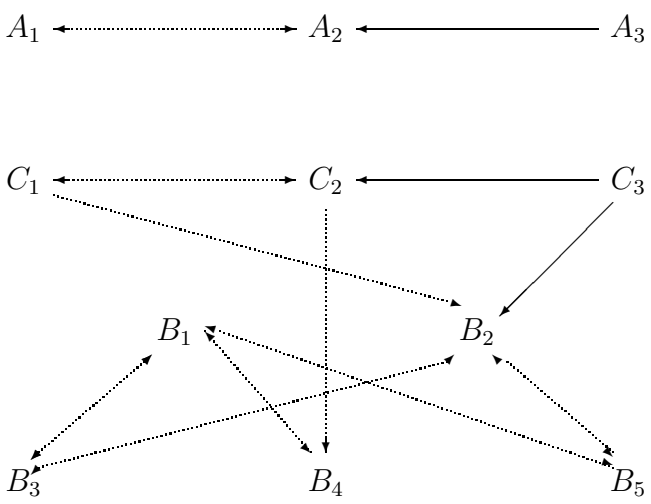

Figure 2: Dialectical base in example 4.2.

5 Conclusions

We have designed a system in the style of logic programming that also enjoys the expressive power of argumentation frameworks. Usually, this expressive power usually results in a lack of efficiency. To address this problem we have developed an inference mechanism based on the notion of precompiled knowledge, that reformulates the traditional approach for drawing conclusions in ODeLP.

The results obtained through this work lead to believe that using these techniques on argumentative systems establishes a new line of research. We think that different frameworks for defeasible argumentation could benefit from the ODeLP approach by incorporating the techniques and algorithms discussed in this paper. Research in this direction is currently being pursued.

References

[1] Capobianco, M. Argumentacion rebatible en entornos dinámicos. Master's thesis, Universidad Nacional del Sur, Bahía Blanca, Argentina, 2002. To appear.

[2] Capobianco, M., Chesñevar, C. I., and Simari, G. R. A Formalization of Dialectical Bases for Defeasible Logic Programming. In *Proceedings of the 6th Internacional Congress of Informatics Engineering* (Capital Federal, Apr. 2000), Universidad Buenos Aires.

[3] Capobianco, M., Chesñevar, C. I., and Simari, G. R. An argumentative formalism for implementing rational agents. In *Proceedings of the 2st Workshop en Agentes y Sistemas Inteligentes (WASI), 7th Congreso Argentino de Ciencias de la Computación (CACIC)* (El Calafate, Santa Cruz, Oct. 2001), Universidad Nacional de la Patagonia Austral, pp. 1051–1062.

[4] Capobianco, M., and Simari, G. R. Defeasible Reasoning in Dynamic Domains. In *Proceedings of the 1st Workshop en Agentes y Sistemas Inteligentes (WASI), 6th Congreso Argentino de Ciencias de la Computación (CACIC)* (Ushuaia, Oct. 2000), Universidad Nacional de la Patagonia San Juan Bosco, pp. 1481–1492.

[5] Chesñevar, C. I. El Problema de la Inferencia en Sistemas Argumentativos: Alternativas para su solución. Master's thesis, Universidad Nacional del Sur, Bahía Blanca, Argentina, Nov. 1996.

[6] de Kleer, J. An assumption based TMS. *Artificial Intelligence 28*, 2 (1986), 127–162.

[7] de Kleer, J. A comparison of ATMS and CSP techniques. In *Proceedings of the 11th International Joint Conference on Artificial Intelligence, Workshop on Practical Reasoning and Rationality* (Detroit, USA, Aug. 1989), N. S. Sridharan, Ed., Morgan Kaufmann, pp. 290–296.

[8] Doyle, J. A Truth Maintenance System. *Artificial Intelligence 12*, 3 (1979), 231–272.

[9] Dung, P. M. On the Acceptability of Arguments and its Fundamental Role in Nonmonotonic Reasoning and Logic Programming and n-Person Games. *Artificial Intelligence 77*, 2 (1995), 321–357.

[10] Elkan, C. A Rational Reconstruction of Nonmonotonic Truth Maintenance Systems. *Artificial Intelligence 43* (1990), 219–234.

[11] Forbus, K., and de Kleer, J. *Building Problem Solvers.* MIT Press, Cambridge, Massachusetts, 1993.

[12] García, A., and Simari, G. R. Logic programming: an argumentative approach, 2002. To appear.

[13] García, A. J. *La Programación en Lógica Rebatible: Lenguaje, Semántica Operacional, y Paralelismo.* PhD thesis, Universidad Nacional del Sur, Bahía Blanca, Argentina, Dec. 2000.

[14] Gelfond, M., and Lifschitz, V. Logic Programs with Classical Negation. In *Proceedings of the 7th International Conference on Logic Programming* (June 1990), D. H. D. Warren and P. Szeredi, Eds., pp. 579–597.

[15] McAllester, D. Truth Maintenance. In *Proceedings of the 8th National Conference on Artificial Intelligence* (Aug. 1990), R. Smith and T. Mitchell, Eds., vol. 2, American Association for Artificial Intelligence, AAAI Press, pp. 1109–1116.

[16] Pollock, J. L. *Cognitive Carpentry: a blueprint for how to build a person.* Bradford/MIT Press, 1995.

[17] Prakken, H. *Logical Tools for Modelling Legal Argument.* Kluwer Academic Publishers, 1997.

[18] Simari, G. R., Chesñevar, C. I., and García, A. J. The Role of Dialectics in Defeasible Argumentation. In *Proceedings of the 14th Conferencia Internacional de la Sociedad Chilena para Ciencias de la Computación* (Nov. 1994), pp. 111–121. http://cs.uns.edu.ar/giia.html.

[19] Simari, G. R., and Loui, R. P. A Mathematical Treatment of Defeasible Reasoning and its Implementation. *Artificial Intelligence 53*, 1–2 (1992), 125–157.

[20] Vreeswijk, G. Abstract Argumentation Systems. *Artificial Intelligence 90*, 1–2 (1997), 225–279.