An argument-based framework to model an agent's beliefs in a dynamic environment

Marcela Capobianco[1], Carlos I. Chesñevar[1,2], and Guillermo R. Simari[1]

[1] Artificial Intelligence Research and Development Laboratory
Department of Computer Science and Engineering
Universidad Nacional del Sur – Av. Alem 1253, (8000) Bahía Blanca, ARGENTINA
EMAIL: {mc,grs}@cs.uns.edu.ar
[2] Artificial Intelligence Research Group – Departament of Computer Science
Universitat de Lleida – Campus Cappont – C/Jaume II, 69 – E-25001 Lleida, SPAIN
EMAIL: cic@eup.udl.es

Abstract. One of the most difficult problems in multiagent systems involves representing knowledge and beliefs of agents in dynamic environments. New perceptions modify an agent's current knowledge about the world, and consequently its beliefs. Such revision and updating process should be performed efficiently by the agent, particularly in the context of real time constraints.
This paper introduces an argument-based logic programming language called *Observation-based Defeasible Logic Programming* (ODeLP). An ODeLP program is used to represent an agent's knowledge in the context of a multiagent system. The beliefs of the agent are modeled with warranted goals computed on the basis of the agent's program. New perceptions from the environment result in changes in the agent's knowledge handled by a simple but effective updating strategy. The process of computing beliefs in a changing environment is made computationally attractive by integrating a "dialectical database" with the agent's program, providing precompiled information about inferences. We present algorithms for creation and use of dialectical databases.

1 Introduction

Knowledge representation issues play a major role in practically all areas of Artificial Intelligence, and MAS is not an exception. Well-known problems in MAS involve the need of complex abilities for reasoning, planning and acting in dynamic environments ([1]). In the last years, argumentation has gained wide acceptance in the multiagent systems (MAS) community by providing tools for designing and implementing different features which characterize interaction among rational agents.

Logic programming approaches to argumentation [2, 3] have proven to be suitable formalization tools in the context of MAS, as they combine the powerful features provided by logic programming for knowledge representation together with the ability to model complex, argument-based inference procedures in unified, integrated frameworks.

Most MAS approaches based on logic programming rely on *extended logic programming* (ELP) ([4]) as underlying formalism. Thus, the agent's knowledge is codified in terms of an ELP program and the well-founded semantics of the program represents the agent's beliefs. Although ELP is expressive enough to capture different kinds of negation (strict and default negation), it has limitations for modeling incomplete and potentially contradictory information. In a MAS context it is common that agents require such capabilities, as they interact with the environment and among themselves, processing new inputs, changing dynamically their beliefs and intentions, etc. Clearly, in such a setting, the argumentation formalism underlying such MAS should be able to incorporate new information into the knowledge base of the agent and reason accordingly.

In this paper we present (ODeLP) (*Observation based Defeasible Logic Programming*), an argument-based formalism for agents reasoning in dynamic environments. Some of the basic notions of ODeLP come from *Defeasible Logic Programming* [5] (DeLP). As in DeLP, the ODeLP formalism uses a knowledge representation language in the style of logic programming and inference is based on argumentation.

To provide the agents with the ability to sense the changes in the world and integrate them into its existing beliefs, in ODeLP we have adapted the knowledge representation system to handle perceptions. Real time issues also play an important role when modeling agent interaction. In an argument-based MAS setting, a timely interaction is particularly hard to achieve, as the inference process involved is complex and computationally expensive. To solve this issue, we will enhance the behavior of ODeLP by incorporating *dialectical databases*, that is, data structures for storing precompiled knowledge. These structures can be used to speed up the inference process when answering future queries.

The remainder of this paper is organized as follows. Section 2 summarizes the main features of the ODeLP formalism. Section 3 reviews briefly previous work on truth maintenance systems, which provided the basis for our notion of dialectical databases, and introduces the notion of dialectical databases, discussing its role as a tool to speed up inference in the ODeLP formalism. Section 4 presents a worked example. Finally, section 5 summarizes the conclusions that have been obtained.

2 ODeLP: Observation-based DeLP

Defeasible Logic Programming (DeLP) [5] provides a language for knowledge representation and reasoning that uses *defeasible argumentation* to decide between contradictory conclusions through a *dialectical analysis*. Codifying the knowledge base of the agent by means of a DeLP program provides a good trade-off between expressivity and implementability. Extensions of DeLP that integrate possibilistic logic and vague knowledge along with an argument-based framework have also been proposed [6]. Recent research has shown that DeLP provides a suitable framework for building real-world applications (e.g. clustering algorithms [7],

intelligent web search [8] and critiquing systems [9]) that deal with incomplete and potentially contradictory information.

In such applications, DeLP is intended to model the behavior of a single intelligent agent in a *static* scenario. DeLP lacks the appropriate mechanisms to represent knowledge in dynamic environments, where agents must be able to perceive the changes in the world and integrate them into its existing beliefs [10]. The ODeLP framework aims at solving this problem by modeling perception as new facts to be added to the agent's knowledge base. Since adding such new facts may result in inconsistencies, an associated updating process is used to solve them. The definitions that follow summarize the main features of ODeLP.

2.1 Language

The language of ODeLP is based on the language of logic programming. In what follows we use concepts like signature, alphabet and atoms with their usual meaning. Literals are atoms that may be preceded by the symbol "$\sim$" denoting *strict* negation, as in ELP. ODeLP programs are formed by *observations* and *defeasible rules*. Observations correspond to facts in the context of logic programming, and represent the knowledge an agent has about the world. *Defeasible rules* provide a way of performing tentative reasoning as in other argumentation formalisms [11, 12].

Definition 1. [Observation]–[Defeasible Rule] *An* observation *is a ground literal L representing some fact about the world, obtained through the perception mechanism, that the agent believes to be correct. A* defeasible rule *has the form $L_0 \prec L_1, L_2, \ldots, L_k$, where L_0 is a literal and $L_1, L_2, \ldots, L_k$ is a non-empty finite set of literals.*

Definition 2. [ODeLP Program]*An* ODeLP program *is a pair $\langle \Psi, \Delta \rangle$, where Ψ is a finite set of observations and Δ is a finite set of defeasible rules. In a program $\mathcal{P}$, the set Ψ must be* non-contradictory *(i.e., it is not the case that $Q \in \Psi$ and $\sim Q \in \Psi$, for any literal Q).*

Example 1. Fig. 1 shows an ODeLP program for assessing the status of employees in a given company. 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. 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 evaluation of the employees' performance, according with their responsibility in the job.

2.2 Inference mechanism

Given an ODeLP program $\mathcal{P}$, a query posed to $\mathcal{P}$ corresponds to a ground literal Q which must be supported by an *argument* [11, 5]. Arguments are built on

```
poor_performance(john).
sick(john).
poor_performance(peter).
high_salary(rose).
applicant(rose).
good_references(rose).
hire(X) -≺ ~high_salary(X), applicant(X).
~hire(X) -≺ high_salary(X), applicant(X).
hire(X) -≺ high_salary(X), applicant(X), good_references(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).
```

Fig. 1. An ODeLP program for assessing the status of employees in a company

the basis of a *defeasible derivation* computed by backward chaining applying the usual SLD inference procedure used in logic programming. Observations play the role of facts and defeasible rules function as inference rules. In addition to provide a proof supporting a ground literal, such a proof must be non-contradictory and minimal for being considered as an argument in ODeLP. Formally:

Definition 3. [Argument – Sub-argument] *Given a ODeLP program $\mathcal{P}$, an* argument $\mathcal{A}$ *for a ground literal Q, also denoted $\langle \mathcal{A}, Q \rangle$, 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,*
3. *$\mathcal{A}$ is minimal with respect to set inclusion in satisfying (1) and (2).*

Given two arguments $\langle \mathcal{A}_1, Q_1 \rangle$ and $\langle \mathcal{A}_2, Q_2 \rangle$, we will say that $\langle \mathcal{A}_1, Q_1 \rangle$ is a sub-argument *of $\langle \mathcal{A}_2, Q_2 \rangle$ iff $\mathcal{A}_1 \subseteq \mathcal{A}_2$.*

Note that to use defeasible rules in arguments we must first obtain their *ground instances*, changing variables for ground terms, such that variables with the same name are replaced for the same term.

As in most argumentation frameworks, arguments in ODeLP can attack each other. This situation is captured by the notion of *counterargument*. Defeat among arguments is defined combining the counterargument relation and a preference criterion (partial order) "$\preceq$". Specificity [13, 11, 14] is the syntactic-based preference criterion used by default in ODeLP, although other alternative criteria can be easily used. Specificity favors those arguments which are *more direct* or *more informed* (i.e., contain more specific information).

Definition 4. [Counter-argument] *An argument $\langle \mathcal{A}_1, Q_1 \rangle$* counter-argues *an argument $\langle \mathcal{A}_2, Q_2 \rangle$ 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 complementary literals.*

Definition 5. [Defeater] *An argument* $\langle \mathcal{A}_1, Q_1 \rangle$ defeats $\langle \mathcal{A}_2, Q_2 \rangle$ *at a literal* Q *if and only if there exists a sub-argument* $\langle \mathcal{A}, Q \rangle$ *of* $\langle \mathcal{A}_2, Q_2 \rangle$ *such that* $\langle \mathcal{A}_1, Q_1 \rangle$ *counter-argues* $\langle \mathcal{A}_2, Q_2 \rangle$ *at* Q*, and either:*

1. $\langle \mathcal{A}_1, Q_1 \rangle$ *is strictly preferred over* $\langle \mathcal{A}, Q \rangle$ *according to the preference criterion "*$\preceq$*" (then* $\langle \mathcal{A}_1, Q_1 \rangle$ *is a* proper defeater *of* $\langle \mathcal{A}_2, Q_2 \rangle$*), or*
2. $\langle \mathcal{A}_1, Q_1 \rangle$ *is unrelated to* $\langle \mathcal{A}, Q \rangle$ *by "*$\preceq$*" (then* $\langle \mathcal{A}_1, Q_1 \rangle$ *is a* blocking defeater *of* $\langle \mathcal{A}_2, Q_2 \rangle$*).*

Defeaters are arguments and may in turn be defeated. Thus, a complete dialectical analysis is required to determine which arguments are ultimately accepted. Such analysis results in a tree structure called *dialectical tree*, in which arguments are nodes labeled as undefeated (`U-nodes`) or defeated (`D-nodes`) according to a marking procedure. Formally:

Definition 6. [Dialectical Tree] *The* dialectical tree *for an argument* $\langle \mathcal{A}, Q \rangle$*, denoted* $\mathcal{T}_{\langle \mathcal{A}, Q \rangle}$*, is recursively defined as follows:*

1. *A single node labeled with an argument* $\langle \mathcal{A}, Q \rangle$ *with no defeaters (proper or blocking) is by itself the dialectical tree for* $\langle \mathcal{A}, Q \rangle$*.*
2. *Let* $\langle \mathcal{A}_1, Q_1 \rangle, \langle \mathcal{A}_2, Q_2 \rangle, \ldots, \langle \mathcal{A}_n, Q_n \rangle$ *be all the defeaters (proper or blocking) for* $\langle \mathcal{A}, Q \rangle$*. The dialectical tree for* $\langle \mathcal{A}, Q \rangle$*,* $\mathcal{T}_{\langle \mathcal{A}, Q \rangle}$*, is obtained by labeling the root node with* $\langle \mathcal{A}, Q \rangle$*, and making this node the parent of the root nodes for the dialectical trees of* $\langle \mathcal{A}_1, Q_1 \rangle, \langle \mathcal{A}_2, Q_2 \rangle, \ldots, \langle \mathcal{A}_n, Q_n \rangle$

Definition 7. [Marking of the Dialectical Tree] *Let* $\langle \mathcal{A}_1, Q_1 \rangle$ *be an argument and* $\mathcal{T}_{\langle \mathcal{A}_1, Q_1 \rangle}$ *its dialectical tree, then:*

1. *All the leaves in* $\mathcal{T}_{\langle \mathcal{A}_1, Q_1 \rangle}$ *are marked as a* `U-node`*.*
2. *Let* $\langle \mathcal{A}_2, Q_2 \rangle$ *be an inner node of* $\mathcal{T}_{\langle \mathcal{A}_1, Q_1 \rangle}$*. Then* $\langle \mathcal{A}_2, Q_2 \rangle$ *is marked as* `U-node` *iff every child of* $\langle \mathcal{A}_2, Q_2 \rangle$ *is marked as a* `D-node`*. The node* $\langle \mathcal{A}_2, Q_2 \rangle$ *is marked as a* `D-node` *if and only if it has at least a child marked as* `U-node`*.*

Dialectical analysis may in some situations give rise to *fallacious argumentation* [15]. In ODeLP dialectical trees are ensured to be free of fallacies [14] by applying additional constraints when building *argumentation lines* (the different possible paths in a dialectical tree). A detailed analysis of these issues is outside the scope of this paper.

Given a query Q and an ODeLP program $\mathcal{P}$, we will say that Q is *warranted* wrt $\mathcal{P}$ iff there exists an argument $\mathcal{T}_{\langle \mathcal{A}, Q \rangle}$ such that the root of its associated dialectical tree $\mathcal{T}_{\langle \mathcal{A}, Q \rangle}$ is marked as a U-node.

Definition 8. [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`*.*

Solving a query Q in ODeLP accounts for trying to find a warrant for Q, as shown in the following example.

Example 2. Consider the program shown in Example 1, and let `suspend(john)` be a query wrt that program. The search for a warrant for `suspend(john)` will result in an argument $\langle \mathcal{A}, \texttt{suspend(john)} \rangle$ with two defeaters $\langle \mathcal{B}, \sim\texttt{suspend(john)} \rangle$ and $\langle \mathcal{C}, \texttt{responsible(john)} \rangle$, where

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

Using specificity as the preference criterion, $\langle \mathcal{B}, \sim\texttt{suspend(john)} \rangle$ is a blocking defeater for $\langle \mathcal{A}, \texttt{suspend(john)} \rangle$, and $\langle \mathcal{C}, \texttt{responsible(john)} \rangle$ is a proper defeater for $\langle \mathcal{A}, \texttt{suspend(john)} \rangle$. The associated dialectical tree is shown in Fig.2. The marking procedure determines that the root node $\langle \mathcal{A}, \texttt{suspend(john)} \rangle$ is a `D-node` and hence `suspend(john)` is not warranted.

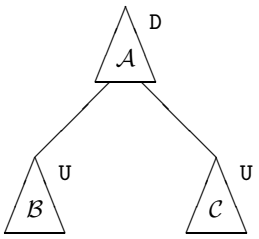

Fig. 2. Dialectical tree from Example 2

2.3 Modeling Beliefs and Perceptions in ODeLP

ODeLP models the beliefs of an agent in a simple way: given a program $\mathcal{P}$ representing an agent's knowledge, a literal Q is believed by the agent iff Q is warranted. In particular, different doxastic attitudes are distinguished wrt a given literal Q:

- Believe that Q is *true* whenever Q is warranted;
- Believe that Q is *false* (*i.e.*, believe $\sim Q$) whenever $\sim Q$ is warranted; and
- Believe that Q is *undecided* whenever none of the above cases apply.

Consistency is a basic property for agent's beliefs, in the sense that it is not possible to believe simultaneously in a literal Q and its complement $\sim Q$ [16]. Agents using ODeLP naturally satisfy this requirement. [3]

[3] For a full discussion of ODeLP properties and their proof the interested reader can consult [14].

In ODeLP, the mechanism for updating the knowledge base of an agent is simple but effective. We assume that perception is carried out by devices that detect changes in the world and report them as new facts (literals). The actual devices used will depend on the particular application domain, and their characterization is outside the scope of this paper. We also make the assumption that the perception mechanism is flawless, and new perceptions always supersede old ones. Any perception will be reported as a new fact α to be added to the set of observations Ψ. If this new perception α is contradictory with Ψ, then necessarily $\sim\alpha \in \Psi$. In such a case, we use a simple update function [17] that computes a new observation set Ψ' as $\Psi \backslash \{\sim\alpha\} \cup \alpha$. Thus new perceptions are always preferred: with a flawless perception mechanism the source of the conflict must be a change in the state of world.

3 Precompiling Knowledge in ODeLP

In *truth maintenance systems* (TMS) the use of precompiled knowledge helps improve the performance of problem solvers. A similar technique will be used in ODeLP to address the real time constrains required in a MAS setting. Next we give a brief overview of TMS and then we describe the mechanism used for precompiling knowledge in ODeLP.

3.1 Truth maintenance systems: a brief overview

Truth Maintenance Systems (TMS) were defined by Doyle in [18] as support tools for problems solvers. The function of a TMS is to record and maintain the reasons for an agent's beliefs. Doyle describes a series of procedures that determine the current set of beliefs and update it in accord with new incoming reasons. Under this view, *rational thought* is deemed as the process of finding reasons for attitudes [18]. Some attitude (such as belief, desire, etc.) is rational if it is supported by some acceptable explanation.

TMS have two basic data structures: *nodes*, which represent beliefs, and *justifications* which model reasons for the nodes. The TMS believes in a node if it has a justification for the node and believes in the nodes involved in it. Although this may seem circular, there are assumptions (a special type of justifications) which involve no other nodes. Justifications for nodes may be added or retracted, and this accounts for a *truth maintenance procedure* [18], to make any necessary revisions in the set of beliefs. An interesting feature of TMS is the use of 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*.

TMS solve part of the belief revision problem in general problem solvers and provide a mechanism for making non-monotonic assumptions. As Doyle mentions in [18] performance is also significantly improved, Even though the overhead required to record justifications for every program belief might seem excessive,

we must consider the expense of not keeping these records. When information about derivations is discarded, the same information must be continually re-derived, even when only irrelevant assumptions have changed.

3.2 Dialectical databases in ODeLP

Based on the existing work in TMS, our goal is to integrate precompiled knowledge into an agent framework based on ODeLP in order to address real-time constraints in a MAS setting. To do so, we want an ODeLP-based agent to be able to answer queries efficiently, by avoiding recomputing arguments which were already computed before.

Note that there are different options for integrating precompiled knowledge with an ODeLP program $\mathcal{P}$. A simple approach would be recording every argument that has been computed so far. However, a large number of arguments can be obtained from a relatively small program, resulting thus in a large database. On the other hand, many arguments are obtained using *different* instances of the *same* defeasible rules. Recording every generated argument could result in storing many arguments which are structurally identical, only differing in the constant names being used to build the corresponding derivations.

Another important problem arises with perceptions. Note that the set of arguments that can be built from a program $\mathcal{P} = \langle \Psi, \Delta \rangle$ also depends on the observation set Ψ. When Ψ is updated with new perceptions, arguments which were previously derivable from $\mathcal{P}$ may no longer be so. If precompiled knowledge depends on Ψ, it should be updated as new perceptions appear. Clearly such an alternative is not suitable, as new perceptions are frequent in dynamic environments. As a consequence, precompiled knowledge should be managed independently from the set of observations Ψ.

Based on the previous analysis we will define a database structure called *dialectical database*, which will keep a record of all possible *potential arguments* in an ODeLP program $\mathcal{P}$ as well as their defeat relationships among them. Potential arguments are formed by non-grounded defeasible rules, depending thus only on the set of rules Δ in $\mathcal{P}$. As we will discuss later, attack relationships among potential arguments can be also captured. Potential arguments and the defeat relationships among them will be stored in the dialectical database. Next we introduce some formal definitions:

Definition 9. [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 iff every instance of a defeasible rule in $\mathcal{B}$ is an instance of a defeasible rule in A.*

Example 3. If $A = \{$ `s(X)` $\prec$ `~r(X)`; `~r(X)` $\prec$ `p(X)`$\}$ then $\mathcal{B} = \{$ `s(t)` $\prec$ `~r(t)`; `~r(a)` $\prec$ `p(a)`$\}$ is an instance of A.

Definition 10. [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 $\langle\langle A, Q \rangle\rangle$ if there exists a non-contradictory set of literals Φ and an instance $\mathcal{B}$ of the rules in A such that $\langle \mathcal{B}, Q \rangle$ is an argument wrt $\langle \Phi, \Delta \rangle$.*

In the definition above the set Φ stands for a state of the world (set of observations) in which we can obtain the instance $\mathcal{B}$ from the set A of defeasible rules such that $\langle \mathcal{B}, Q \rangle$ is an argument (as defined in Def.3). Note that the set Φ must necessarily be non-contradictory to model a coherent scenario.

Precompiled knowledge associated with an ODeLP program $\mathcal{P} = \langle \Psi, \Delta \rangle$ will involve the set of all potential arguments that can be built from $\mathcal{P}$ as well as the defeat relationships among them.

- **Potential arguments**: to obtain and record every potential argument of $\mathcal{P}$ we have devised an algorithm that efficiently identifies all potential arguments as distinguished subsets of the rules in Δ.[4] Potential arguments will save time in computing arguments when solving queries. Instead of computing a query for a given ground literal Q, the ODeLP interpreter will search for a potential argument A for Q such that a particular instance $\mathcal{B}$ of A is an argument for Q wrt $\mathcal{P}$.
- **Defeat relationships among potential arguments**: Recording information about defeat relationships among potential arguments is also useful as it helps to speed up the construction of dialectical trees when solving queries, as we will see later. To do this, we extend the concepts of counterargument and defeat for potential arguments. A potential argument $\langle\langle A_1, Q_1 \rangle\rangle$ *counter-argues* $\langle\langle A_2, Q_2 \rangle\rangle$ at a literal Q if and only if there is a potential sub-argument $\langle\langle A, Q \rangle\rangle$ of $\langle\langle A_2, Q_2 \rangle\rangle$ such that Q_1 and Q are contradictory literals.[5] Note that potential counter-arguments may or may not result in a real conflict between the instances (arguments) associated with the corresponding potential arguments. In some cases instances of these arguments cannot co-exist in any scenario (*e.g.*, consider two potential arguments based on contradictory observations).
  The notion of defeat is also extended to potential arguments. Since specificity is a syntactic-based criterion, a particular version of specificity [14] is applicable to potential arguments, determining when a potential argument is more informed or more direct than another.

Using potential arguments and their associated defeat relation we can formally define the notion of *dialectical databases* associated with a given ODeLP program $\mathcal{P}$.

Definition 11. [Dialectical Database] *Let $\mathcal{P} = \langle \Psi, \Delta \rangle$ be an ODeLP program. The* dialectical database *of $\mathcal{P}$, denoted as $\mathcal{DB}_\Delta$, is a 3-tuple* $(\text{PotArg}(\Delta), D_p, D_b)$ *such that:*

1. $\text{PotArg}(\Delta)$ *is the set* $\{\langle\langle A_1, Q_1 \rangle\rangle, \ldots, \langle\langle A_k, Q_k \rangle\rangle\}$ *of all the potential arguments that can be built from Δ.*

[4] For space reasons this algorithm is not detailed in this paper. The interested reader is referred to [14].

[5] Note that $Q(X)$ and $\sim Q(X)$ are contradictory literals although they are non-grounded. The same idea is applied to identify contradiction in potential arguments.

2. *D_p and D_b are relations over the elements of* $\mathrm{PotArg}(\Delta)$ *such that for every pair* $(\langle\!\langle A_1, Q_1\rangle\!\rangle, \langle\!\langle A_2, Q_2\rangle\!\rangle)$ *in* D_p *(respectively* D_b*) it holds that* $\langle\!\langle A_2, Q_2\rangle\!\rangle$ *is a proper (respectively blocking) defeater of* $\langle\!\langle A_1, Q_1\rangle\!\rangle$.

Example 4. Consider the program in example 1. The dialectical database of $\mathcal{P}$ is composed by the following potential arguments:

- $\langle\!\langle A_1, \texttt{hire(X)}\rangle\!\rangle$,
  where $A_1 = \{\texttt{hire(X)} \prec \texttt{\sim high\_salary(X), applicant(X)}\}$.
- $\langle\!\langle A_2, \texttt{\sim hire(X)}\rangle\!\rangle$,
  where $A_2 = \{\texttt{\sim hire(X)} \prec \texttt{high\_salary(X), applicant(X)}\}$.
- $\langle\!\langle A_3, \texttt{hire(X)}\rangle\!\rangle$,
  where $A_3 = \{\texttt{hire(X)} \prec \texttt{high\_salary(X), applicant(X), good\_references(X)}\}$.
- $\langle\!\langle B_1, \texttt{suspend(X)}\rangle\!\rangle$,
  where $B_1 = \{\texttt{suspend(X)} \prec \texttt{\sim responsible(X)}\}$.
- $\langle\!\langle B_2, \texttt{suspend(X)}\rangle\!\rangle$,
  where $B_2 = \{\texttt{suspend(X)} \prec \texttt{\sim responsible(X)};$
  $\texttt{\sim responsible(X)} \prec \texttt{poor\_performance(X)}\}$.
- $\langle\!\langle B_3, \texttt{\sim suspend(X)}\rangle\!\rangle$,
  where $B_3 = \{\texttt{\sim suspend(X)} \prec \texttt{responsible(X)}\}$.
- $\langle\!\langle B_4, \texttt{\sim suspend(X)}\rangle\!\rangle$,
  where $B_4 = \{\texttt{\sim suspend(X)} \prec \texttt{responsible(X)};$
  $\texttt{responsible(X)} \prec \texttt{good\_performance(X)}\}$.
- $\langle\!\langle B_5, \texttt{\sim suspend(X)}\rangle\!\rangle$,
  where $B_5 = \{\texttt{\sim suspend(X)} \prec \texttt{responsible(X)};$
  $\texttt{responsible(X)} \prec \texttt{poor\_performance(X), sick(X)}\}$.
- $\langle\!\langle C_1, \texttt{responsible(X)}\rangle\!\rangle$,
  where $C_1 = \{\texttt{responsible(X)} \prec \texttt{poor\_performance(X)}\}$.
- $\langle\!\langle C_2, \texttt{\sim responsible(X)}\rangle\!\rangle$,
  where $C_2 = \{\texttt{\sim responsible(X)} \prec \texttt{poor\_performance(X)}\}$.
- $\langle\!\langle C_3, \texttt{responsible(X)}\rangle\!\rangle$,
  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 3, 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.

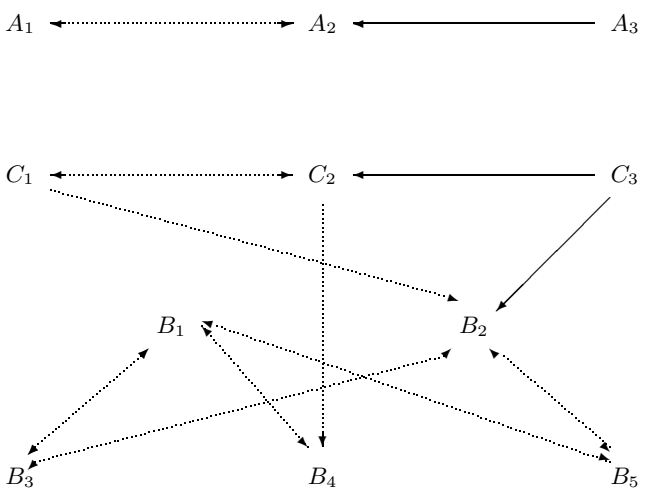

Fig. 3. Dialectical database corresponding to Example 4.

3.3 Speeding up inference in ODeLP with dialectical databases

Given a ODeLP program $\mathcal{P}$, its dialectical database $\mathcal{DB}_\Delta$ can be understood as a graph from which *all* possible dialectical trees computable from $\mathcal{P}$ can be obtained. In the original ODeLP framework (as detailed in Section 2), solving a query Q wrt a given program $\mathcal{P} = \langle \Psi, \Delta \rangle$ accounted for obtaining a warranted argument $\langle \mathcal{A}, Q \rangle$. As already discussed, computing warrant involves many intermediate steps which are computationally expensive (computing arguments, detecting defeaters, building a dialectical tree, etc.).

Using the dialectical database we can speed up the inference process in ODeLP by keeping track of all possible potential arguments and the defeat relationshisp among them. Given a query Q, the extended ODeLP framework (i.e. including a dialectical database) will select first a potential argument $\langle\langle A, S \rangle\rangle$ (such that Q is a ground instance of S) that can be instantiated into $\langle \mathcal{A}, Q \rangle$, supporting Q. From the D_p and D_b relationships in $\mathcal{DB}_\Delta$ the potential defeaters for $\langle\langle A, Q \rangle\rangle$ can be identified, and also instantiated.

To describe how the inference process is assisted by dialectical databases we present algorithm 1. It obtains a warrant for a query Q from a program $\mathcal{P} = \langle \Psi, \Delta \rangle$. To do this, the algorithm considers the potential arguments $\langle\langle A, S \rangle\rangle$ such that Q is an instance of S, an tries to find an instance $\langle \mathcal{B}, Q \rangle$ of $\langle\langle A, S \rangle\rangle$ that is also an argument with respect to $\mathcal{P}$, according to definition 9. This is done in function `argument` which in case such instance exists returns it in the parameter $\langle \mathcal{B}, Q \rangle$. Next, $\langle \mathcal{B}, Q \rangle$ is analyzed to see whether it is a warrant for Q. To do this, the relations D_p and D_b are used to find the defeaters of $\langle \mathcal{B}, Q \rangle$. Once the system finds an instance of the potential defeaters that is in conflict with $\langle \mathcal{B}, Q \rangle$, the

function `acceptable` checks if they are arguments with respect to $\mathcal{P}$. Then the `state` function (see algorithm 2) determines the marking of these defeaters (*i.e.*, if they are marked as `U-nodes` or `D-nodes`) and finally this information is used to compute the state of $\langle \mathcal{B}, Q \rangle$.

Algorithm 1 Inference process
input: $\mathcal{P} = \langle \Psi, \Delta \rangle$, Q
output:$\langle \mathcal{B}, Q \rangle$ *(a warrant for Q, if any)*

For every $\langle\langle A, S \rangle\rangle$ in $PotArg(\Delta)$ such that argument($\langle\langle A, S \rangle\rangle$,$Q$,$\mathcal{P}$,$\langle \mathcal{B}, Q \rangle$)
//Looks for instances of the potential arguments that support Q
state := undefeated
For every $\langle\langle A_2, X \rangle\rangle$ in $PotArg(\Delta)$ such that $(A_2, A_1) \in D_p$ or $(A_2, A_1) \in D_p$
//and then determines the state of their defeaters
For every instance $\langle \mathcal{C}, R \rangle$ of $\langle\langle A_2, X \rangle\rangle$ such that $\langle \mathcal{C}, R \rangle$ defeats $\langle \mathcal{B}, Q \rangle$
and acceptable($\langle \mathcal{C}, R \rangle$,$\mathcal{P}$)
if state($\langle \mathcal{C}, R \rangle$, $\mathcal{P}$, $\emptyset$, $\{\langle \mathcal{B}, Q \rangle\}$) = undefeated
then state := defeated
//Sets the state of the main argument according to its defeaters
if state = undefeated
then return($\langle \mathcal{B}, Q \rangle$)
//If any of the instances remains undefeated it is a warrant

The `state` algorithm used in the inference process takes as input an ODeLP program $\mathcal{P}$, an argument $\langle \mathcal{B}, Q \rangle$ based on it, and the *interference* and *support* argumentative lines up to this point, respectively denoted as `IL` and `SL`. Simply put, `IL` represents the set of arguments with an even level in the actual path of the tree under construction, and `SL` the arguments with an odd level. Then the `state` algorithm works like algorithm 1, analyzing the defeaters of $\mathcal{B}$ to define its state. However, one extra condition must be met: defeaters must also comply with the rules established for avoiding fallacies [14]. This test is performed by the function `valid`.

To conclude, figure 4 summarizes the main elements of the ODeLP-based agent architecture. The agent's knowledge is represented by an ODeLP program $\mathcal{P}$. Perceptions from the environment result in changes in the set of observations in $\mathcal{P}$, handled by an appropriate updating mechanism as discussed previously. In order to solve queries from other agents, the agent relies on the ODeLP inference engine. Queries are speeded up by first searching on the potential arguments stored in the dialectical database, applying the algorithms discussed before. The final answer to a given query will be *yes*, *no* or *undecided*, according to the warrant status of the query with respect to $\mathcal{P}$.

Algorithm 2 State

input: $\langle \mathcal{B}, Q \rangle$, $\mathcal{P}$, IL, SL

output:state

state := undefeated
For every pair $(A_1, A_2) \in D_p$ or D_b such that $\langle \mathcal{B}, Q \rangle$ is an instance of A_1
//Uses the stored defeat relation to find the defeaters of $\langle \mathcal{B}, Q \rangle$
For every instance $\langle \mathcal{C}, S \rangle$ of A_2 such that acceptable($\langle \mathcal{C}, S \rangle$,$\mathcal{P}$) and valid($\langle \mathcal{C}, S \rangle$,IL,SL)
//Then checks for every defeater whether it gives raise to fallacies.
if $\langle \mathcal{B}, Q \rangle$ is in SL and state($\langle \mathcal{C}, S \rangle$,$\mathcal{P}$,IL,SL $\cup \{\mathcal{B}\}$) = undefeated
then state := defeated
if $\langle \mathcal{B}, Q \rangle$ is in IL and state($\langle \mathcal{C}, S \rangle$,$\mathcal{P}$,IL $\cup \{\mathcal{B}\}$,SL) = undefeated
then state := defeated
//The recursive call does the same for the defeaters of $\langle \mathcal{C}, S \rangle$
return(state)

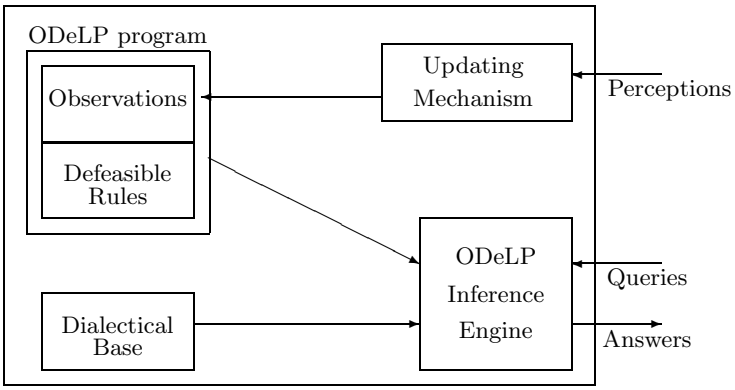

Fig. 4. Agent architecture using ODeLP as underlying framework

4 A worked example

In this section we present a toy example to illustrate the use of a dialectical database to speed up inference in ODeLP. Let us consider the program $\mathcal{P}$ in Example 1 as an agent's knowledge to model the status of different employees in a company. The associated dialectical database $\mathcal{DB}_\Delta$ is shown in Example 4.

Suppose that the agent has to decide whether John should be suspended or not, considering the query `suspend(john)`. As shown in Example 2, solving this query wrt $\mathcal{P}$ involved a dialectical tree with three arguments (see Figure 2). Let us analyze now how the agent would proceed to perform the same inference using the dialectical database $\mathcal{DB}_\Delta$. Following Algorithm 1, the potential argument $\langle\!\langle B_2, Q \rangle\!\rangle$ will be instantiated resulting in the argument $\langle \mathcal{A}, Q \rangle$, with

$$\mathcal{A} = \{\sim\texttt{suspend(john)} \prec \texttt{responsible(john)};\ \texttt{responsible(john)} \prec \texttt{poor\_performance(john),sick(john)}\}$$

From the dialectical database $\mathcal{DB}_\Delta$ it follows that $\langle\!\langle B_2, Q \rangle\!\rangle$ has defeaters $\langle\!\langle B_3, Q_3 \rangle\!\rangle$ and $\langle\!\langle B_5, Q_5 \rangle\!\rangle$ (see the list of pairs in D_b in Example 4), which are respectively instantiated to $\langle \mathcal{B}, \sim suspend(john) \rangle$ and $\langle \mathcal{C}, responsible(john) \rangle$, with:

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

$$\mathcal{C} = \{\texttt{responsible(john)} \prec \texttt{poor\_performance(john),sick(john)}\}$$

Note that from the information in $\mathcal{DB}_\Delta$ associated with $\langle\!\langle B_3, Q_3 \rangle\!\rangle$ and $\langle\!\langle B_5, Q_5 \rangle\!\rangle$ there are no more pairs in D_p or D_b to consider (i.e, there are no more links in the graph to new defeaters for these potential arguments that can be instanciated to defeat $\mathcal{B}$ or $\mathcal{C}$). As a consequence, a dialectical tree identical to the one shown in Figure 2 has been computed on the basis of the potential arguments present in the dialectical database and their associated defeat relationships. There are no more possible potential arguments supporting `suspend(john)`. Therefore there is no warrant for `suspend(john)`.

Consider now a different situation for the same sample program $\mathcal{P}$. Suppose that the facts `applicant(susan)` and `high_salary(susan)` are added as new observations to Ψ. In order to solve the query `~hire(susan)` wrt to $\mathcal{P}$ the same dialectical database $\mathcal{DB}_\Delta$ can be used but relying on different potential arguments as those used above. Now other instances can be obtained relying on the new perceived facts. In this case, the potential argument $\langle\!\langle A_2, Q_2 \rangle\!\rangle$ is instantiated to $\langle \mathcal{D}, \sim hire(susan) \rangle$, with

$$\mathcal{D} = \{\sim\texttt{hire(susan)} \prec \texttt{high\_salary(susan), applicant(susan)}\}$$

From the information available in $\mathcal{DB}_\Delta$ a defeater for $\langle\!\langle A_2, Q_2 \rangle\!\rangle$ is detected, namely $\langle\!\langle A_1, Q_1 \rangle\!\rangle$. However there is no argument which can be obtained as an instance of $\langle\!\langle A_1, Q_1 \rangle\!\rangle$ wrt the current set Ψ, and hence there is no defeater for

$\langle \mathcal{D}, \sim hire(susan) \rangle$. Therefore ~`hire(susan)` is warranted (as it is supported by an argument $\langle \mathcal{D}, \sim hire(susan) \rangle$ with no defeaters).

The applicant Susan may ask now for a lower salary, given that she wants to get the job. This results in a new perception for the agent, updating Ψ by adding ~`high_salary(susan)` and consequently removing `high_salary(susan)`. As in the previous situations, no change is performed on the existing dialectical database. Nevertheless, the set of beliefs of the agent changes: ~`hire(susan)` is no longer believed, since no argument supporting ~`hire(susan)` can be built as an instance of a potential argument in $\mathcal{DB}_{\Delta}$. On the contrary, `hire(susan)` is now in the set of beliefs as it is warranted by a tree with a single node: argument $\mathcal{E} = \{$`hire(susan)` $\prec$ ~`high_salary(susan), applicant(susan)`$\}$.

5 Conclusions and future work

Solid theoretical foundations for agent design should be based on proper formalisms for knowledge representation and reasoning [19]. Thus, we have defined a framework for representing knowledge and beliefs of agents in dynamic environments, where new perceptions can modify the agent's view about its world.

To comply with real time issues when modeling agent interaction in a MAS setting we have proposed the notion of dialectical databases. We have discussed the main issues in the integration of this component into ODeLP, such as building the dialectical database, adapting the specificity criterion for potential arguments and modifying the inference process to take advantage of the new component. Based on this, we can affirm that the use of precompiled knowledge can improve the performance of argument-based systems in the same way Truth Maintenance Systems assist general problem solvers. We believe that this technique can also be applied to other argumentative frameworks, allowing its use in a new set of applications.

Part of our current work involves extending the analysis of ODeLP properties presented in [14] in the context of multiagent systems. We are also working on a complexity analisis of ODeLP that considers the construction and use of dialectical databases, and confirms the results obtained empirically.

Acknowledgements

This research was partially supported by Projects TIC2001-1577-C03-01 and TIC2003-00950, by Ramón y Cajal Program (*Ministerio de Ciencia y Tecnología*, Spain), by CONICET (Argentina), by CIC (Argentina), by the *Secretaría General de Ciencia y Tecnología de la Universidad Nacional del Sur* and by *Agencia Nacional de Promoción Científica y Tecnológica* (PICT 2002 No. 13096). The authors would like to thank anonymous reviewers for providing helpful comments to improve the final version of this paper.

References

1. Wooldridge., M.J.: Introduction to Multiagent Systems. John Wiley and Sons (2002)
2. Chesñevar, C.I., Maguitman, A., Loui, R.: Logical Models of Argument. ACM Computing Surveys **32** (2000) 337–383
3. Prakken, H., Vreeswijk, G.: Logical systems for defeasible argumentation. In Gabbay, D., ed.: Handbook of Philosophical Logic. Volume 4. Kluwer Academic Publisher (2002) 219–318
4. Gelfond, M., Lifschitz, V.: Classical negation in logic programs and disjunctive databases. New Generation Computing (1991) 365–385
5. García, A., Simari, G.: Defeasible Logic Programming: An Argumentative Approach. Theory and Practice of Logic Programming **4** (2004) 95–138
6. Chesñevar, C., Simari, G., Alsinet, T., Godo, L.: A logic programming framework for possibilistic argumentation with vague knowledge. In: Proc. of Uncertainty in Artificial Intelligence Conference (UAI 2004), Banff, Canada (to appear). (2004)
7. Gomez, S., Chesñevar, C.: A Hybrid Approach to Pattern Classification Using Neural Networks and Defeasible Argumentation. In: Proc. of Intl. 17th FLAIRS Conference. Palm Beach, FL, USA, AAAI (2004) 393–398
8. Chesñevar, C., Maguitman, A.: ArgueNet: An Argument-Based Recommender System for Solving Web Search Queries. In: Proc. of Intl. IEEE Conference on Intelligent Systems IS-2004. Varna, Bulgaria (to appear). (2004)
9. Chesñevar, C., Maguitman, A.: An argumentative approach to assesing natural language usage based on the web corpus. In: Proc. of European Conference on Artificial Intelligence (ECAI 2004). Valencia, Spain (to appear), ECCAI (2004)
10. Pollock, J.L.: Taking Perception Seriously. In: Proceedings of the 1st International Conference on Autonomous Agents. (1997) 526–527
11. Simari, G.R., Loui, R.P.: A Mathematical Treatment of Defeasible Reasoning and its Implementation. Artificial Intelligence **53** (1992) 125–157
12. Prakken, H., Sartor, G.: Argument-based extended logic programming with defeasible priorities. Journal of Applied Non-classical Logics **7** (1997) 25–752
13. Poole, D.L.: On the Comparison of Theories: Preferring the Most Specific Explanation. In: Proceedings of the Ninth International Joint Conference on Artificial Intelligence, IJCAI (1985) 144–147
14. Capobianco, M.: Argumentación rebatible en entornos dinámicos. PhD thesis, Universidad Nacional del Sur, Bahía Blanca, Argentina (2003)
15. Simari, G.R., Chesñevar, C.I., García, A.J.: The Role of Dialectics in Defeasible Argumentation. In: Proceedings of the XIV Conferencia Internacional de la Sociedad Chilena para Ciencias de la Computación. (1994) 111–121 http://cs.uns.edu.ar/giia.html.
16. Alferes, J.J., Pereira, L.M.: On logic program semantics with two kinds of negation. In: Proceedings of Joint International Conference and Symposium on Logic Programm, Washington, USA (1992) 574–588
17. Katsuno, H., Mendelzon, A.: On the difference between updating a knowledge base and revising it. In P.Gardenfors, ed.: Belief Revision. Cambridge University Press (1992) 183–203
18. Doyle, J.: A Truth Maintenance System. Artificial Intelligence **12** (1979) 231–272
19. Baral, C., Gelfond, M.: Reasoning Agents in Dynamic Domains. In Minker, J., ed.: Workshop on Logic-Based Artificial Intelligence, College Park, Maryland, Computer Science Department, University of Maryland (1999)