An argumentative formalism for implementing rational agents

Marcela Capobianco Carlos I. Chesñevar Guillermo R. Simari

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

Abstract

The design of *intelligent agents* is a key issue for many applications. Although there is no universally accepted definition of intelligence, a notion of *rational agency* has been proposed as an alternative for the characterization of *intelligent agency*.

Modeling the epistemic state of a rational agent is one of the most difficult tasks to be addressed in the design process, and its complexity is directly related to the formalism used for representing the knowledge of the agent. This paper presents the main features of Observation-based Defeasible Logic Programming (ODeLP), a formalism tailored for agents that perform defeasible reasoning in dynamic domains.

Most agents must have a timely interaction with their environment. Since the cognitive process of rational agents is complex and computationally expensive, this interaction is particularly hard to achieve. To solve this issue, we propose an optimization of the inference process in ODeLP based on the use of precompiled knowledge. This optimization can be efficiently implemented using concepts from pattern matching algorithms.

Keywords: knowledge representation, argumentation, rational agents.

1 Introduction

The design of *intelligent agents* is a key issue for many applications. Although there is no universally accepted definition of intelligence, a notion of *rational agency* was proposed by Russell [12] as an alternative for the characterization of *intelligent agency*. In short, an agent is said to be rational if it performs the *right* actions according to the information it possesses and the goals it wants to achieve.

Modeling the epistemic state of a rational agent is the most difficult task to be addressed in its design process. We believe that using an adequate formalism for representing the knowledge and modeling the reasoning of the agent can be the cornerstone to solve this problem. Consequently, we have developed a system tailored for this task, called Observation based Defeasible Logic Programming (ODeLP). ODeLP is based on the formalism of *Defeasible Logic Programming* [7] (DeLP), a system that combines the advantages of logic programming and *defeasible argumentation* [1] by using an *argumentation process* to decide between contradictory goals.

Since DeLP provides a good trade-off between expressivity and implementability [1], codifying the knowledge of the agent with a DeLP program is an interesting alternative. Nevertheless,

[1] See [3, 11, 5] for a more detailed analysis of this concept.

DeLP does not provide perception abilities. When a dynamic environment is considered, the agent must be able to perceive the changes in the world and integrate them into its existing beliefs [10]. To do this, the agent should have the capability of sensing its surroundings, and incorporating the new observations into its knowledge. This accounts for the need of a more specific framework. ODeLP addresses these shortcomings by supplying an updating process that efficiently reflects changes in the world into the agent's knowledge base.

Most agents must have a timely interaction with its environment. Since the cognitive process of rational agents is complex and computationally expensive, this interaction is particularly hard to achieve. To solve this issue, we propose an optimization for the inference process of ODeLP based on the use of precompiled knowledge. This optimization can be efficiently implemented using concepts from pattern matching algorithms.

The remainder of this paper is organized as follows. Section 2 presents the ODeLP formalism and describes the perception mechanisms devised for this system. Section 3 defines how precompiled knowledge can be used to speed up the inference process and Section 4 details an efficient implementation of the ODeLP formalism. Finally, section 5 states the main conclusions obtained through this work.

2 Observation-based DeLP

The ODeLP formalism is a modification of DeLP [7]. As such, it inherits its advantages as a knowledge representation and reasoning tool. Besides, it is more suited to model the epistemic state of rational agents. The language of ODeLP[2] is composed by a set of *observations* encoding the knowledge the agent has about the world, and a set of *defeasible rules* representing ways of extending observations with tentative information (*i.e.*, information that can be used if nothing is posed against it). In the following definitions, we consider literals as atoms that may be preceded by the symbol "$\sim$" denoting classical negation.

Definition 2.1. (*Observation*)
An *observation* is a ground literal representing some fact about the world that the agent believes to be correct. ■

Definition 2.2. (*Defeasible rule*)
A *defeasible rule* is an ordered pair, denoted as "`Head` $-\!\prec$ `Body`", where `Head` is a ground literal and `Body` is a non-empty finite set of ground literals. ■

Definition 2.3. (*Defeasible logic program*)
A *defeasible logic program* is composed by a finite set Ψ of observations and a finite set Δ of defeasible rules. In an ODeLP program $\mathcal{P}$, the set Ψ must be non-contradictory, *i.e.*, it cannot contain a literal and its complement with respect to classical negation. When required, we denote $\mathcal{P}$ as $\langle \Psi, \Delta \rangle$. ■

It is important to note that we only focus on ground programs. In ODeLP, rules with variables are viewed as "schemata" that represent all their ground instances.

Having defined the concept of defeasible logic programs, we focus on the description of a *consequence operator* for these programs. Inference in ODeLP is obtained by performing a *defeasible query* (or simply a *query*) which is defined in a logic programming style using defeasible rules as Horn-like clauses. Formally:

[2] In what follows, we briefly survey the definitions of the ODeLP formalism. For a more detailed analysis we refer the interested reader to a previous publication [2].

Definition 2.4. (*Defeasible derivation*)
Let $\mathcal{P} = \langle \Psi, \Delta \rangle$ be a ODeLP program and let q be a 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 $\mathcal{P} \mid\sim q$) if for every q_i, $1 \leq i \leq n$, it holds that $q_i \in \Psi$, or q_i is a consequent of a defeasible rule $r \in \Delta$, $r = q_i \prec l_1, \ldots, l_m$, where $l_1, \ldots, l_m$ are ground literals previously occurring in s. ■

Note that although the set Ψ must be non-contradictory, $\mathcal{P}$ may allow the defeasible derivation of complementary literals. In order to insure a sound defeasible inference, answers to queries must be supported by *arguments*. Formally:

Definition 2.5. (*Argument – Sub-argument*)
Given a ODeLP program $\mathcal{P}$, an *argument* $\mathcal{A}$ for a ground literal q,also denoted $\langle \mathcal{A}, q \rangle$ or simply $\mathcal{A}$, is a subset 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) $\mathcal{A}$ is minimal with respect to set inclusion in satisfying the previous conditions. 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* $\mathcal{A}$.

Example 2.1. Consider the ODeLP program $\mathcal{P}$:

`cat(tom).`	`~social(X) -< aloof(X).`
`young(tom).`	`social(X) -< cat(X), young(X).`
`pet(tom).`	`social(X) -< pet(X).`
`cat(grace).`	`aloof(X) -< cat(X).`
	`has-tail(X) -< cat(X).`

The arguments that follow can be built from the program above:

- $\langle \mathcal{A}_1, \sim$ `social(tom)`$\rangle$, where $\mathcal{A}_1 = \{$`~social(tom) -< aloof(tom), aloof(tom) -< cat(tom)`$\}$
- $\langle \mathcal{A}_2,$ `social(tom)`$\rangle$, where $\mathcal{A}_2 = \{$`social(tom) -< cat(tom), young(tom)`$\}$
- $\langle \mathcal{A}_3,$ `social(tom)`$\rangle$, where $\mathcal{A}_3 = \{$`social(tom) -< pet(tom)`$\}$

The argument $\mathcal{A}_1$ concludes that Tom is not a social creature since Tom is a cat, and cats are aloof. $\mathcal{A}_2$ supports that Tom is social, considering that young cats are friendly animals and $\mathcal{A}_3$ concludes that Tom is social since it is a pet. ■

An argument $\mathcal{A}$ for a query q provides a tentative proof for q, which may be in conflict with other arguments that contradict $\mathcal{A}$ (*e.g.*, see the example above where $\mathcal{A}_2$ contradicts $\mathcal{A}_3$). This leads to the formal definition of *counter-argument*.

Definition 2.6. (*Counter-argument*)
An argument $\langle \mathcal{A}_1, q_1 \rangle$ *counter-argues* an argument $\langle \mathcal{A}_2, q_2 \rangle$ (or, equivalently, $\langle \mathcal{A}_1, q_1 \rangle$ 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 the set $\{q_1, q\}$ is contradictory.[3] ■

[3]We say that two literals are contradictory if they are complementary with respect to classical negation.

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 $\langle \mathcal{A}_1, q_1 \rangle$ is deemed better than (or unrelated to) $\langle \mathcal{A}_2, q_2 \rangle$ according to some preference criterion. In this paper we adopt *specificity* (as defined in [2]), although any other criterion that induces a partial order on the set of possible arguments could be used. Formally:

Definition 2.7. (*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 $\langle \mathcal{A}_1, q_1 \rangle$ is better than $\langle \mathcal{A}, q \rangle$ according to a preference criterion "$\leq$" (then $\langle \mathcal{A}_1, q_1 \rangle$ is a *proper defeater* of $\langle \mathcal{A}_2, q_2 \rangle$); or $\langle \mathcal{A}_1, q_1 \rangle$ is unrelated to $\langle \mathcal{A}, q \rangle$ by "$\leq$" (then $\langle \mathcal{A}_1, q_1 \rangle$ is a *blocking defeater* of $\langle \mathcal{A}_2, q_2 \rangle$). ■

Informally, 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.[4] 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 labeled as `U-nodes` and defeated ones as `D-nodes`. Formally:

Definition 2.8. (*Dialectical tree*)
Let $\mathcal{A}$ be an argument for q. A *dialectical tree* for $\langle \mathcal{A}, q \rangle$, 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 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

- for 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`.

Dialectical trees may give rise to contradictory or circular argumentation, which are particular cases of the so-called *fallacious argumentation* [13, 9], a problem suffered by most argumentation systems. In ODeLP, this is solved by the concept of *acceptable argumentation lines*.

Definition 2.9. (*Argumentation line*)
Let $\mathcal{P} = \langle \Psi, \Delta \rangle$ be a ODeLP program, and let $\langle \mathcal{A}, q \rangle$ be an argument in $\mathcal{P}$. An *argumentation line* starting from $\langle \mathcal{A}, q \rangle$, denoted $\lambda^{\langle \mathcal{A}, q \rangle}$ (or simply λ) is a possibly infinite sequence of arguments $\lambda^{\langle \mathcal{A}, q \rangle} = [\langle \mathcal{A}_0, q_0 \rangle, \langle \mathcal{A}_1, q_1 \rangle, \ldots, \langle \mathcal{A}_n, q_n \rangle, \ldots]$ satisfying the following conditions:

[4] The search space can be reduced by applying an $\alpha - \beta$ pruning strategy [13].

1. If $\langle \mathcal{A}, q \rangle$ has no defeaters, then $\lambda^{\langle \mathcal{A}, Q \rangle} = [\langle \mathcal{A}, q \rangle]$.
2. If $\langle \mathcal{A}, q \rangle$ has a defeater $\langle \mathcal{B}, s \rangle$ in $\mathcal{P}$, then $\lambda^{\langle \mathcal{A}, q \rangle} = \langle \mathcal{A}, q \rangle \circ \lambda^{\langle \mathcal{B}, s \rangle}$.

where the '$\circ$' operator stands for concatenating $\langle \mathcal{A}, q \rangle$ and $\lambda^{\langle \mathcal{B}, s \rangle}$. ■

An argumentation line λ can be split into two disjoint sets: λ_S of supporting (even-indexed) arguments, and λ_I of interfering (odd-indexed) arguments. This distinction leads us to the notion of *acceptable argumentation lines*, in which fallacious sequences of arguments cannot occur.[5]

Definition 2.10. (*Acceptable argumentation line*)
Let $\mathcal{P} = \langle \Psi, \Delta \rangle$ be a ODeLP program, let $\lambda = [\langle \mathcal{A}_0, q_0 \rangle, \langle \mathcal{A}_1, q_1 \rangle, \ldots, \langle \mathcal{A}_n, q_n \rangle, \ldots]$ be an argumentation line in $\mathcal{P}$, and let $\lambda' = [\langle \mathcal{A}_0, q_0 \rangle, \langle \mathcal{A}_1, q_1 \rangle, \ldots, \langle \mathcal{A}_k, q_k \rangle, \ldots]$ be an initial segment of λ. The sequence λ' is an *acceptable argumentation line* in $\mathcal{P}$ if and only if it is the longest initial segment in λ satisfying the following conditions:

1. The sets ${\lambda'}_S$ and ${\lambda'}_I$ are each non-contradictory sets of arguments wrt $\mathcal{P}$.
2. No argument $\langle \mathcal{A}_j, q_j \rangle$ in λ' is a sub-argument of an earlier argument $\langle \mathcal{A}_i, q_i \rangle$ of λ' $(i < j)$.
3. There is no subsequence of arguments $[\langle \mathcal{A}_{i-1}, q_{i-1} \rangle, \langle \mathcal{A}_i, q_i \rangle, \langle \mathcal{A}_{i+1}, q_{i+1} \rangle]$ in λ', such that $\langle \mathcal{A}_i, q_i \rangle$, is a blocking defeater for $\langle \mathcal{A}_{i-1}, q_{i-1} \rangle$ and $\langle \mathcal{A}_{i+1}, q_{i+1} \rangle$ is a blocking defeater for $\langle \mathcal{A}_i, q_i \rangle$. ■

To avoid fallacious reasoning, branches in a dialectical tree are required to be acceptable argumentation . An argument will be a *warrant* if and only if it is ultimately undefeated in its associated dialectical tree. Formally:

Definition 2.11. (*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 2.2. Consider example 2.1.The argument $\langle \mathcal{A}_3, \texttt{social(tom)} \rangle$ has a unique defeater $\langle \mathcal{A}_1, \sim\texttt{social(tom)} \rangle$, which in turn has only one defeater that can be added to the dialectical tree, $\langle \mathcal{A}_2, \texttt{social(tom)} \rangle$. ($\langle \mathcal{A}_3, \texttt{social(tom)} \rangle$ can not be reintroduced in the tree because of the restriction for argumentation lines stated in definition 2.10.) Hence the dialectical tree for $\langle \mathcal{A}_3, \texttt{social(tom)} \rangle$ has a single branch with these three nodes. According to the marking criterion described before, both $\langle \mathcal{A}_2, \texttt{social(tom)} \rangle$. and $\langle \mathcal{A}_3, \texttt{social(tom)} \rangle$ are marked as `U-nodes`, and $\langle \mathcal{A}_1, \sim\texttt{social(tom)} \rangle$ is marked as `D-node`. From this we can conclude that $\mathcal{A}_3$ is a warrant for `social(tom)`. ■

Using the notion of warrant an agent can express the following doxastic attitudes with respect to a ground literal q present in the rules of $\mathcal{P}$.

- Believe that q is *true* if there exists an argument $\langle \mathcal{A}, q \rangle$ such that $\mathcal{A}$ is a warrant for q).
- Believe that q is *false*, *i.e.*, believe in $\overline{q}$, (where $\overline{q}$ stands for the complement of q with respect to classical negation) if there is a warrant for $\overline{q}$.
- Believe that q is *undefined* (neither q nor $\overline{q}$ are warranted).

Any literal q not present in the rules of $\mathcal{P}$ is deemed as *unknown*.

[5] See [13, 7] for an in-depth discussion.

2.1 Incorporating perception

In our system, the task of perceiving can be carried out by any mechanism that detects the changes in the world and reports the literals representing those changes. The specification of this mechanism depends on the particular application domain and it is not addressed in our work. We only assume that it cannot produce false observations. The perceived literals are added to the knowledge of the agent, into the set of observations Ψ. If new facts are carelessly added, Ψ may become inconsistent.

Example 2.3. Suppose that $\sim$`young(tom)` is to be added to the ODeLP program in example 2.1 (*i.e.*, Tom has become a grown-up cat). This contradicts the previous (and still existing) fact expressing that tom is young. ■

To avoid this we have defined an updating process [8] that removes any element of Ψ contradicting the new observation. Note that according to our criterion, new perceptions are always preferred over older ones. There is a simple reason behind this policy: given our initial assumption, both of the observations in disagreement were correct at the time of their assimilation. As a result, the only explanation for the conflict is a change in the state of world, and the new fact should be favored since it reflects the actual state. Formally:

Definition 2.12. (*Updating*)
Let $\mathcal{P} = \langle \Psi, \Delta \rangle$ be an ODeLP program and α an observation. The *updating* of Ψ by α, denoted as $\Psi * \alpha$, is defined as $(\Psi - \{\overline{\alpha}\}) \cup \{\alpha\}$ ■

It is worth mentioning that by updating the set of observations the agent can modify its beliefs, changing its previous picture of the world when faced with new information.

3 Dialectical Bases

The use of precompiled knowledge may help to optimize the inference process of ODeLP in the same way truth maintenance systems improve the performance of problem solvers. In this section, we address how to build this component and how to use it to speed-up the agent's reasoning. We start by defining the concept of *hypothetical argument*, which is the building block of ODeLP's precompiled knowledge.

Definition 3.1. (*Hypothetical argument - Sub-argument*)
Let Δ be a set of defeasible rules. A subset $\mathcal{A}$ of Δ is said to be a *hypothetical argument* for a literal q, also denoted $\langle \mathcal{A}, q \rangle_h$ or simply $\mathcal{A}$ when no confusion may arise, if there is a consistent subset Φ of the literals present in the rules of Δ, such that $\langle \mathcal{A}, q \rangle_h$ is an argument with respect to $\mathcal{P} = \langle \Phi, \Delta \rangle$. $\langle \mathcal{A}_1, q_1 \rangle_h$ is a *hypothetical sub-argument* (or simply a sub-argument) of $\langle \mathcal{A}_2, q_2 \rangle_h$ if $\mathcal{A}_1 \subseteq \mathcal{A}_2$. ■

In the definition above, the set Φ represents a state of the world in which the hypothetical argument can be constructed. Note that these structures depend only on the set of defeasible rules of the program.

To build the precompiled knowledge for a program $P = \langle \Psi, \Delta \rangle$ we need to record all of the hypothetical arguments that can be build from Δ. Unfortunately, definition 3.1 does not help to address this task. Besides, being a subset of the defeasible rules in $\mathcal{P}$ is not a sufficient condition for being a hypothetical argument. As shown in example 3.1, some additional restrictions must be satisfied.

Example 3.1. Continuing with program 2.1, the set of defeasible rules

$\mathcal{A} = \{$`aloof(grace)` $-\!\prec$ `cat(grace)`,
`has-tail(grace)` $-\!\prec$ `cat(grace)`$\}$

is not a hypothetical argument for any conclusion. Note that for any subset Φ of the literals in Δ such that $\Phi \cup \mathcal{A}$ is non-contradictory, and for any conclusion q such that q can be (defeasible) derived from $\Phi \cup \mathcal{A}$, there exists some $\mathcal{B} \subset \mathcal{A}$ that also fulfills these conditions. Then the set $\mathcal{A}$ is not minimal. ■

To identify the hypothetical arguments of a given program, we present a constructive definition for this concept. It can be shown that both formalizations are equivalent.

Definition 3.2. (*Hypothetical argument*)
Let Δ be a set of defeasible rules. A subset $\mathcal{A}$ of Δ is said to be a *hypothetical argument* for a literal q, also denoted $\langle \mathcal{A}, q \rangle_h$, if and only if $\mathcal{G}(\mathcal{A}) \cup \mathcal{A} \nvdash q$, $literals(\mathcal{A})$ is consistent, and $\mathcal{A}$ is minimal with respect to set inclusion in fulfilling the previous conditions. ■

Lets see how precompiled knowledge can be used to optimize reasoning. Consider a rational agent that uses a ODeLP program $\mathcal{P} = \langle \Psi, \Delta \rangle$ to represent its epistemic state. To build the precompiled knowledge of this agent we need to record every hypothetical argument that can be constructed using the rules in Δ. When reasoning from the program $\mathcal{P} = \langle \Psi, \Delta \rangle$, the agent uses the precompiled arguments that are valid in this situation, *i.e.*, those which can be constructed from the current set of observations Ψ. This prevents the construction of the arguments and the search for their defeaters that would take place when solving a query in the ODeLP formalism. Even though recording the hypothetical arguments of $\mathcal{P}$ is an onerous task, it is performed only once (after codifying the knowledge of the agent). Besides, hypothetical arguments are independent from the current set of perceptions, and they don't have to be rebuilt or modified every time the set of observations Ψ changes. As a result, the construction of hypothetical arguments does not complicate the interaction of the agent with its environment, and significatively speeds up the reasoning process.

To complete the definition of ODeLP's precompiled knowledge we need to describe how to store the defeat relation among hypothetical arguments. To do this, we generalize the notions of *counterargument* (definition 2.6) and *defeat* (definition 2.7), in order to consider hypothetical arguments. A hypothetical argument $\langle \mathcal{A}_1, q_1 \rangle_h$ *counter-argues* another hypothetical argument $\langle \mathcal{A}_2, q_2 \rangle_h$ at a literal q if and only if there is a sub-argument $\langle \mathcal{A}, q \rangle_h$ of $\langle \mathcal{A}_2, q_2 \rangle_h$ such that $\{q_1, q\}$ is contradictory. This kind of counter-arguments represents only a potential conflict between the hypothetical arguments in contest. It might be the case that these arguments cannot co-exist in any scenario (*e.g.*, consider two hypothetical arguments based on contradictory observations).

To check whether a hypothetical argument defeats one of its counter-arguments, the criterion used to compare pairs of arguments must be adapted to pairs of hypothetical arguments. In particular, we have redefined specificity to compare arguments independently from the set of observations.[6] The definition of defeat between hypothetical arguments can be stated analogously to definition 2.7, since it is parameterized with respect to the defeat criterion.

Having described the ground ideas, we can now introduce the notion of *dialectical base* to formally define the precompiled knowledge of the agent according to the previous discussion.

Definition 3.3. (*Dialectical base*)
Let $\mathcal{P} = \langle \Psi, \Delta \rangle$ be a ODeLP program. The tuple $(\mathbb{A}, D_p, D_b)$ is said to be the *dialectical base*

[6] See [2] for the definition and analysis of this criterion.

Algorithm 3.1. Inference process

input: $\mathcal{P} = \langle \Psi, \Delta \rangle$, q
output: $\langle \mathcal{A}_1, q \rangle_h$ (a warrant for q, if any)
For every $\langle \mathcal{A}_1, q \rangle_h$ in $\mathbb{A}$ such that acceptable($\mathcal{A}_1$,$\mathcal{P}$)
    state := undefeated
    For every $\mathcal{A}_2$ in $\mathbb{A}$ such that $(\mathcal{A}_1, \mathcal{A}_2) \in D_p$ or $(\mathcal{A}_1, \mathcal{A}_2) \in D_p$ and
    acceptable($\mathcal{A}_2$,$\mathcal{P}$)
        if state($\mathcal{A}_2$, $\mathcal{P}$, $\emptyset$, $\{\mathcal{A}_1\}$) = undefeated
            then state := defeated
    if state = undefeated
        then return($\langle \mathcal{A}_1, q \rangle_h$)

of $\mathcal{P}$ with respect to Δ, denoted as $\mathcal{DB}_\Delta$, if and only if: (1) $\mathbb{A}$ is a set of hypothetical arguments such that a hypothetical argument $\mathcal{A} \in \mathbb{A}$ if and only if $\mathcal{A}$ is based on Δ; and (2) D_p and D_b are relations over the elements of $\mathbb{A}$ such that for every pair $(\mathcal{A}_1, \mathcal{A}_2)$ it holds that $\mathcal{A}_2$ is a proper (respectively blocking) defeater of $\mathcal{A}_1$ if and only if $(\mathcal{A}_1, \mathcal{A}_2)$ belongs to D_p (respectively D_b). ■

To build the dialectical base of a ODeLP program $\mathcal{P} = \langle \Psi, \Delta \rangle$, the set $\mathbb{A}$ is generated including in it every subset of the defeasible rules in Δ that satisfies the properties enumerated in definition 3.2. Then the defeat relation between elements of $\mathbb{A}$ can be computed, using the chosen defeat criterion. For every pair $(\mathcal{A}_1, \mathcal{A}_2)$, such that $\mathcal{A}_1, \mathcal{A}_2 \in \mathbb{A}$ and $\mathcal{A}_2$ is a proper (respectively blocking) defeater for $\mathcal{A}_1$, we add $(\mathcal{A}_1, \mathcal{A}_2)$ to D_p (respectively D_b). It is important to choose a set of appropriate data structures in order to optimize the construction and use of the dialectical base.

Let us consider how precompiled knowledge favors the inference process when the system is faced with a query q with respect to a program $\mathcal{P} = \langle \Psi, \Delta \rangle$. The traditional procedure starts by building an argument $\mathcal{A}_1$ for q from the rules in $\mathcal{P}$, and then looking for defeaters of $\mathcal{A}_1$ that may prevent $\mathcal{A}_1$ from becoming a warrant for q. Next, the state of $\mathcal{A}_1$ is decided based on the condition of its defeaters (as established in the marking process). In contrast, if the dialectical base $\mathcal{DB}_\Delta$ is used, the inference process may be carried out as described in algorithm 3.1. In this case, every possible argument is already recorded in $\mathcal{DB}_\Delta$, and hence there is no need of constructing the arguments for q nor its corresponding defeaters. The system only selects the hypothetical arguments for q that are that are valid for the particular Ψ under consideration. To this purpose, every $\langle \mathcal{A}_1, q \rangle_h \in \mathcal{DB}_\Delta$ is checked using the function `acceptable(`$\mathcal{A}_1$`,`$\mathcal{P}$`)` to verify if the ground of $\mathcal{A}_1$ is included in Ψ, and $\mathcal{A}_1$ is consistent and minimal with respect to $\mathcal{P}$. Next, every hypothetical argument $\langle \mathcal{A}_1, q \rangle_h$ that fulfills these conditions 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 $\mathcal{A}_1$ and the `state` function (see algorithm 3.2) determines the marking of these defeaters (*i.e.*, if they are marked as `U-nodes` or `D-nodes`). Finally, this information is used to compute the state of $\mathcal{A}_1$.

It remains to analyze the task of the `state` algorithm. It takes as input an argument $\mathcal{A}_1$, a ODeLP program $\mathcal{P}$ such that $\mathcal{A}_1$ is based on $\mathcal{P}$, and the interference and support argumentative lines up to this point, `IL` and `SL`. Then the algorithm works in a similar manner to algorithm 3.1, analyzing the defeaters of $\mathcal{A}_1$ to define its state. However, one extra condition must be met: defeaters must also comply with the rules established in definition 2.10 regarding acceptability

Algorithm 3.2. State

input: $\mathcal{A}_1$, $\mathcal{P} = \langle \Psi, \Delta \rangle$, IL, SL
output: state
state := undefeated
For every $\mathcal{A}_2$ in $\mathbb{A}$ such that $((\mathcal{A}_1, \mathcal{A}_2) \in D_p$ or $(\mathcal{A}_1, \mathcal{A}_2) \in D_p)$ and acceptable($\mathcal{A}_2$,$\mathcal{P}$) and valid($\mathcal{A}_2$,IL,SL)
    if $\mathcal{A}_1$ is a supporting argument and state($\mathcal{A}_2$,$\mathcal{P}$,IL,$SL \cup \{\mathcal{A}_1\}$) = undefeated
        then state := defeated
    if $\mathcal{A}_1$ is an interfering argument and state($\mathcal{A}_2$,$\mathcal{P}$,$IL \cup \{\mathcal{A}_1\}$,$SL$) = undefeated
        then state := defeated
return(state)

of the argumentative lines. This test is performed by the function `valid`.

4 Guidelines for an Efficient Implementation

In the previous section we proposed a set of algorithms for optimizing the inference process in ODeLP based on the creation and use of dialectical bases. To function properly, this optimization must be developed using adequate data structures and algorithms. This section presents a fast implementation for dialectical bases using *pattern matching algorithms* [4, 6].

In a pattern matching problem, a collection of objects is compared to a collection of patterns in order to determine all the existing matches. This kind of algorithms has been extensively used in many AI programs, such as production system interpreters, which use it as a component to determine which productions have satisfied condition parts. We intend to adapt the methods used in these interpreters to obtain an efficient implementation of the algorithms presented in section 3.

A production system program consists of a collection of `IfThen` statements called *productions*. The data operated by the productions is held in a global database called *working memory*. By convention, the `If` part of the productions is called its *left hand side* (LHS), and the `Then` part its *right hand side* (RHS). The LHS of a production is composed by a sequence of patterns: that is, a sequence of partial descriptions of working memory elements. When a pattern P describes an element E, P is said to *match* E. The RHS of a production consists of an unconditional sequence of actions and some of these actions may change the contents of the working memory. The interpreter evaluates the LHS of the productions to determine which are satisfied given the current contents of the working memory and performs the actions of the selected productions.

The traditional approach to implement production systems is to combine a process called *indexing* with *direct interpretation* of the LHS. The key idea in the indexing process is to extract one or more features from the working memory elements and use those features to hash into the collection of productions. This renders a set of productions which may have satisfied LHSs. In the direct interpretation step the interpreter examines each LHS in this set to decide whether it is satisfied. If an appropriate representation is chosen, these techniques can be applied to the implementation of dialectical bases. In what follows, we explore this idea.

To implement dialectical bases we must define a set of adequate data structures, which should:

- Provide a fast access to the set of hypothetical arguments holding a given conclusion in order to efficiently find the set of argument supporting the main query.
- Link every hypothetical argument with its corresponding defeaters to speed up the construction of the dialectical tree.
- Help to perform the consistency and minimality checks of the hypothetical argument with respect to the current the set of observations.

To satisfy this requirements, every hypothetical argument $\mathcal{A}$ in $\mathcal{DB}_\Delta$ is modeled by the following set of attributes:

- Argument conclusion.
- Argument rules: the rules of Δ that are present in $\mathcal{A}$
- Argument ground: the ground of the set $\mathcal{A}$. If $\mathcal{A}$ is an argument with respect to $P = \langle \Psi, \Delta \rangle$ (where Ψ is the current observation set of the program) then the ground of $\mathcal{A}$ must be a subset of Ψ
- Consistency literals: this set, denoted as $\texttt{CL}(\mathcal{A})$, is composed by the complement of every literal present in the rules of $\mathcal{A}$. For $\mathcal{A}$ to be consistent with the current observation set Ψ, none of these literals should belong to Ψ.
- Minimality literals: this set, denoted as $\texttt{ML}(\mathcal{A})$ is composed by the literals in $heads(\mathcal{A})$ - $bodies(\mathcal{A})$. If any of these elements belongs to Ψ, then $\mathcal{A}$ is not minimal with respect to Ψ.
- Blocking defeaters: a list of links to the blocking defeaters for $\mathcal{A}$.
- Proper defeaters: a list of links to the proper defeaters for $\mathcal{A}$.

The dialectical base $\mathcal{DB}_\Delta$ is represented by a collection of these structures, using one for each hypothetical argument that can be built from Δ. All the hypothetical arguments should be initialized when the dialectical base is constructed.

Under this setting, we say that a hypothetical argument $\mathcal{A}$ in a dialectical base $\mathcal{DB}_\Delta$ *matches* the current observation set Ψ if and only if the argument base of $\mathcal{A}$ is included in Ψ, $\Psi \cap \texttt{CL}(\mathcal{A}) = \emptyset$, and $\Psi \cap \texttt{ML}(\mathcal{A}) = \emptyset$. It can be easily shown that $\mathcal{A}$ matches the current observation set Ψ if and only if $\mathcal{A}$ can be built from Ψ and the rules in Δ. Thus, the algorithm `Acceptable` can be seamlessly implemented using these conditions. In this way, expensive consistency and minimality checks are replaced by intersections between sets of literals. The dialectical base should also be indexed with respect to the conclusions of the hypothetical arguments. This index can be used to search for the arguments supporting a certain query.

Using the structure proposed above, the construction of the dialectical tree for a query q can be performed combining indexing and direct interpretation, in a similar manner to pattern matching algorithms. First we use the index to look for a set of argument with q as conclusion. Next, the interpreter chooses an argument $\mathcal{A}$ from this set of candidates. The action associated with every hypothetical argument $\mathcal{A}$ consists in adding it to the dialectical tree. Then the links to the defeaters of $\mathcal{A}$ are used to select a set of candidates. A direct interpretation on every element $\mathcal{B}$ of this set decides if $\mathcal{B}$ is an argument with respect to the current set of observations (using the algorithm acceptable described before). If this is the case, the candidate must pass a final test before adding it to the tree: it must be concordant and non-circular with respect to its argumentation line. At this point the consistency literals are used. If the argument $\mathcal{B}$ under consideration plays the role of a support (respectively interference) argument then no literal

in $\mathcal{A}$ should appear in $\texttt{CL}(\mathcal{B}_i)$, where $\mathcal{B}_i$ is any previous support (respectively interference) argument in the argumentation line. To see whether $\mathcal{B}$ is a circular argument, the set of rules in $\mathcal{B}$ must be checked to ensure that it is not a subset of the rules of a previous argument in its argumentative line. Once that $\mathcal{B}$ is deemed as valid, it is added into the dialectical tree, and the links to its defeaters are used to determine which hypothetical arguments must be analyzed in the future. This process continues until there is no more defeaters to consider and the construction of the dialectical tree is finished. Then the marking of the tree is made as usual.

The implementation described above can be further optimized taking into account the following facts. Once a hypothetical argument is matched to a set of observations Ψ and declared valid, it remains valid until Ψ is updated with new perceptions. An argument in $\mathcal{DB}_\Delta$ can be matched to Ψ many different times. Hence, if Ψ is not updated very often, it may be worthwhile to add an extra field (which may be named `acceptable`) in the representation of hypothetical arguments signaling whether it has been matched to Ψ and the result of this operation. When the system is faced with a hypothetical argument $\mathcal{A}$ it acts according to the value of the `acceptable` attribute. If $\texttt{acceptable} = \texttt{true}$, it means that $\mathcal{A}$ has been previously compared to Ψ with satisfying results. Then the system proceeds to check if $\mathcal{A}$ is concordant and non-circular. In contrast, if $\texttt{acceptable} = \texttt{false}$, then $\mathcal{A}$ has been matched to Ψ with negative results, and $\mathcal{A}$ cannot be added to the dialectical tree. Finally, if $\texttt{acceptable} = \texttt{unknown}$, then $\mathcal{A}$ has not been compared with Ψ and the full process must be performed.

In a first attempt, Ψ may be matched to every argument in the dialectical base when it is updated. This is convenient for the acceptable algorithm, since it saves the matching step. Nevertheless, the dialectical base may be composed by many arguments and the proposed mechanism generates an excessive overhead when a new observation is added. To solve this, the system may use a lazy approach in which every argument in $\mathcal{DB}_\Delta$ is marked as `unknown` when Ψ is updated. If the direct interpretation step is performed on an argument $\mathcal{A}$ marked as `unknown`, then the results of the match operation are recorded accordingly in $\mathcal{A}$'s structure. Note that every time the set of observations changes all the hypothetical arguments in the dialectical base should be marked as `unknown`.

5 Conclusions

ODeLP is a complete and adequate framework for representing the epistemic state of rational agents. Our formalism provides mechanisms to update the program from information acquired perceptually, making the agent adaptable to a dynamic world. Incorporating perception in argument-based systems is a new-sprung idea that allows the use of argumentative frameworks in a whole new set of practical applications.

We have also defined the notion of dialectical bases and discussed the main issues in the integration of this component into ODeLP. It has also been addressed how to efficiently implement these concepts using algorithms from production systems interpreters. As a result, it can be shown that the use of precompiled knowledge can improve the performance of argumentative formalisms. Summing up, we can conclude that the incorporation of ODeLP into an agent architecture results in a model with interesting theoretical and practical features.

References

[1] Capobianco, M., and Chesñevar, C. I. Using Logics Programs to Model an Agent's Epistemic State. In *Proceedings of the 7th Workshop on Aspectos Teóricos de la Inteligencia Artificial (ATIA), 2nd Workshop of Investigadores en Ciencias de la Computación (WICC)* (La Plata, May 2000), Universidad Nacional de La Plata.

[2] 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.

[3] Chesñevar, C. I., Maguitman, A., and Loui, R. Logical Models of Argument. *ACM Computing Surveys 32*, 4 (Dec. 2000), 337–383.

[4] Cohen, B. A powerful and efficient structural pattern recognition system. *Artificial Intelligence 9*, 1 (1977), 223–255.

[5] 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.

[6] Forgy, C. Rete: A Fast Algorithm for the Many Patterns/Many Objects Match Problem. *Artificial Intelligence 19*, 1 (1982), 17–37.

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

[8] Katsuno, H., and Mendelzon, A. On the difference between updating a knowledge base and revising it. In *Belief Revision*, P.Gardenfors, Ed. Cambridge University Press, 1992, pp. 183–203.

[9] Martinez, D., and García, A. Significancia de las falacias en los sistemas argumentativos. In *Proceedings of the 4th Congreso Argentino en Ciencias de la Computación (CACIC)* (1999), Universidad Nacional del Centro de la Provincia de Buenos Aires.

[10] Pollock, J. L. Taking Perception Seriously. In *Proceedings of the 1st International Conference on Autonomous Agents* (Feb. 1997), pp. 526–527.

[11] Prakken, H., and Vreeswijk, G. Logical systems for defeasible argumentation. In *Handbook of Philosophical Logic*, D. Gabbay, Ed. Kluwer Academic Publisher, 2001. To appear.

[12] Russell, S. J. Rationality and Intelligence. *Artificial Intelligence 94*, 1–2 (1997), 57–77.

[13] 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.