Combining a Causal Language with Argumentation: A First Approach

María Laura Cobo* **Guillermo Ricardo Simari**
Artificial Intelligence Research and Development Laboratory (LIDIA)
Universidad Nacional del Sur
Bahía Blanca, B8000CPB, Argentina
[mlc,grs]@cs.uns.edu.ar

Abstract

The development of systems with the ability to reason about change notion and actions has been of great importance for the artificial intelligence community. The definition and implementation of systems capable of managing defeasible, incomplete, unreliable, or uncertain information has been also an area of much interest. With a few exceptions research on these two ways of reasoning was independently pursued. Nevertheless, they are complementary and closely related, since many applications that deal with defeasible information also depends on the occurrence of events and time.

$DeLP$ is an argumentative system appropriate for commonsense reasoning. It is interesting to extend this system adding mechanisms to manage events and time. Here we analyze some of the consequences of changing $DeLP$ representation language introducing *Event Calculus* syntax, particularly the role played by the *commonsense rules of inertia*.

Keywords: Argumentative Systems, Knowledge Representation, Defeasible Reasoning, Commonsense Reasoning, Temporal Reasoning, Reasoning about change notion and actions.

1 INTRODUCTION

The development of systems with the ability to reason about causal information (information inferred from events occurrence), has been of great importance for the artificial intelligence community. Research in this area has been interested on this issue as a way to find a solution for a wider variety of problems where the occurrence of actions and the moment they occur makes a difference [5].

Argumentation Systems [2, 16], were developed in order to deal with incomplete or unreliable information. In real scenarios this situation is quite common, specially when we deal with dynamic systems, *i.e.*, systems where the knowledge available to reason with changes frequently (new information become available or information that used to be available became unavailable or invalid). Usually, since it is very difficult to represent all the information related to the objects under consideration, the information that appears as supporting our reasoning is incomplete. As a matter of fact, there are formalisms such as the *Situation Calculus* [19] where this problem is quite relevant. When new information about an entity becomes available, *i.e.*, knowledge changes, we must revise all the representation.

In order to improve on this problem we will follow an *argumentative* approach. $DeLP$ [4] is an argumentation system, which combines results from Logic Programming and Defeasible reasoning

*Partially supported by Agencia de Promoción Científica y Tecnológica (PICT 13096, PICT 15043, PAV 2003 Nro. 076) and the Universidad Nacional del Sur

providing tools for knowledge representation and commonsense reasoning. We are interested in the development of an argumentative system based on $DeLP$ that can deal with causal information. At the same time we intend to define an enhanced system by adding to $DeLP$ more capabilities. Following [15], a system that performs commonsense reasoning should include a way of dealing with fundamental concepts such as:

- *Representation:* The language must be able to build a representation of world scenarios and also must represent commonsense knowledge about that world. The representation can be made through some data structure and should facilitate automated reasoning.

  Commonsense knowledge is usually represented through rules known as *"commonsense laws of inertia"*, with the purpose of, among other things, solving the *frame problem, i.e.*, represent the knowledge that remains unchanged after the execution of an action.

- *Commonsense Entities:* The system must represent objects, agents, time-varying properties, events, and time. These last two entities are crucial because they establish *"how"* the information changes.

- *Commonsense Domains:* The system must represent and reason about time, space, and mental states. It must also deal with object identity, *i.e.*, it should be able to determine references to a specific object.

- *Commonsense Phenomena:* The system must address the common sense law of inertia, the release from the commonsense law of inertia, concurrent events with cumulative and canceling effects, context-sensitive effects, continuous change, delayed effects, indirect effects, nondeterministic effects, preconditions and triggered events.

- *Reasoning:* The system must specify a process for reasoning by using representation of scenarios and representations of commonsense knowledge. The system must support some kind of default reasoning, temporal projection, abduction, and postdiction (or explanation).

We will extend $DeLP$ taking these concepts on consideration, introducing the possibility of including causal information, depending on *events* and *time*. Some representative languages such as the *Event Calculus* and the *Situation Calculus* deal with incomplete information through the use of *circumscription* to minimize the extension of certain predicates used in the representation. The advantage of the argumentative approach is that it makes possible to reach an answer after a dialectical analysis.

This work analyzes some of the consequences of changing $DeLP$ representation language by introducing part of the *Event Calculus* primitives, focusing on the role played by the *commonsense rules of inertia*. The analysis performed here will lead us to the basic definition of the new system Causal $DeLP$, denoted $\mathcal{C}DeLP$.

he paper is structured as follows, on Section 2 the basics on defeasible reasoning are presented, particularly the argumentative system $DeLP$. In this section, the justification of $DeLP$'s extension is also presented. Causal languages, such as *event calculus* are presented on Section 3. The first step in order to get causal argumentation is provided on Section 4, particularly how inertial rules affect arguments construction. The last section presents the conclusions.

2 ARGUMENTATION AS DEFAULT REASONING

Non-monotonic reasoning provides a way to reason extracting conclusions over an incomplete information scenario. If we obtain new information, then previous conclusions may be retracted. Com-

monsense reasoning must have always a non-monotonic component, since most commonsense inferences could become unsupported and therefore should be retracted. One of the first attempts to a formalization of non-monotonic reasoning was made by John McCarthy [7] in the 1970's. In early 1980's several non-monotonic formalisms were developed: Circumscription [8], Default Logic [18], and a modal approach to non-monotonic logics [12]. Argumentation formalisms were developed during the 1980's (see [16, 2].)

In general, an argumentative system deals with five elements, at least in the abstract layer: *an underlying logical language* (for our purpose we need a temporal-logic language, we will use the *Event Calculus* syntaxs), *a definition of argument, a conflict and rebuttal among arguments definition*, *a form of argument evaluation*, and *a notion of defeasible logic consequence* (again in this case it must be *defeasible temporal-logic consequence*.)

In many cases, the five above mentioned elements are not explicitly defined because they are clearly not independent. In fact, dependencies among them allow the identification of four fundamental layers in argumentative systems [16]: *The Logical Layer* that comprises the language definition, the inference rules, and the argument construction; *The Dialectic Layer* that both involves the definition of conflict between arguments and formalizes the way of solving those possible conflicts; *the Procedural Layer* which defines the interchange of arguments; and the *Strategic Layer* that presents heuristics for argument comparison during a debate based on maximizing success possibilities. Since our work is based on $DeLP$'s language we will introduce brief account below. For a complete presentation see [4].

2.1 Defeasible Logic Programming ($DeLP$)

Defeasible Logic Programming ($DeLP$) is an argumentation formalism that implements the layers expressed above. Information can be represented in the form of weak rules in a declarative manner, and a defeasible argumentation inference mechanism is used for finding the entailed conclusions. In $DeLP$ an argumentation formalism will be used for deciding between contradictory goals. Queries will be supported by arguments that could be defeated by other arguments. A query Q will succeed when there is an argument $\mathcal{A}$ for Q that is *warranted*, *i.e.*, the argument $\mathcal{A}$ that supports Q is found undefeated by the procedure that implements a dialectical analysis. The defeasible argumentation basis of $DeLP$ allows to build applications that deal with incomplete and contradictory information in dynamic domains. Thus, the resulting approach is suitable for representing agent's knowledge and for providing an argumentation based reasoning mechanism for that agent (see for example [3, 1].)

In $DeLP$ weak rules represent a key element for introducing defeasibility and they are used to represent a defeasible relationship between pieces of knowledge. This connection among knowledge could be defeated after all things are considered. General common sense reasoning should be defeasible in a way that is not explicitly programmed. Rejection should be the result of the global consideration of the available knowledge that the agent, performing such reasoning, has at his disposal, Defeasible Argumentation provides a way of doing that.

$DeLP$ language is defined in terms of three disjoint sets: a set of facts, a set of strict rules and a set of defeasible rules. In $DeLP$'s language a literal "L" is a ground atom "A" or a negated ground atom "$\sim A$", where "$\sim$" represents the strong negation.

DEFINITION **1** ($DeLP$ [4])

A $DeLP$ program is defined in term of two disjoint sets: a set of *facts* and *strict rules*(Π) and finally one of *defeasible rules* (Δ), where

- A *fact* is a literal, *i.e.*, a ground atom, o a negated ground atom.

- A *strict rule* is a rule denoted as "$Head \leftarrow Body$", where $Head$ is a literal and, $Body$, is finite set of literals. A strict rule can also be written as:

$$L_0 \leftarrow L_1, \ldots, L_n (n > 0)$$

  where L_0 is rule's $Head$ and each $L_i, i \geq 0$ is a literal.

- A *defeasible rule* is a rule noted as $L_0 \prec L_1, \ldots, L_n$. Again L_i is a literal and $i \geq 0$

Notice that strict negation may affect any literal, in particular may affect L_0. At first sight the only difference between strict and defeasible rules is the way they are denoted, although their meaning is clearly different. In the first kind of rule there are no doubts about the conclusion expressed on the rule, while in the other we only assure that *"reasons to believe in the body of the rule are reasons to believe in the consequent"*. We require the definitions of *Defeasible derivation* and *Argument Structure*.

DEFINITION **2** *Defeasible Derivation* A defeasible derivation for a ground literal, L, is a finite sequence of ground literals, where each literal is in the sequence because L is a fact in set Π of the program, or there is a rule (strict or defeasible) in the program, whose head is literal L and all the literals in the *Body* appears in the sequence before.

DEFINITION **3** *Argument Structure* Let L be a literal and $(P) = (\Pi, \Delta)$ a $DeLP$ program. We say that $\langle \mathcal{A}, L \rangle$ is an argument structure for L, if $\mathcal{A}$ is a set of defeasible rules from Δ, such that:

1. there is a defeasible derivation for L from $\Pi \cup \mathcal{A}$,
2. the set $\Pi \cup \mathcal{A}$ is non-contradictory, and
3. $\mathcal{A}$ is minimal (there is not a subset of $\mathcal{A}$ such that satisfies the previous conditions)

Let see a classic example of the non-monotonic reasoning literature.

EXAMPLE **1** (extracted from [4]) Let P_1 be a $DeLP$ program defined as: $P_1 = (\Pi_1, \Delta_2)$, where:

$$\Pi_1 = \left\{ \begin{array}{l} bird(X) \leftarrow chicken(X), \\ bird(X) \leftarrow penguin(X), \\ \sim flies(X) \leftarrow penguin(X), \\ chicken(tina), \\ penguin(tweety), \\ scared(tina) \end{array} \right\} \qquad \Delta_1 = \left\{ \begin{array}{l} flies(X) \prec bird(X), \\ \sim flies(X) \prec chicken(X), \\ flies(X) \prec chicken(X), scared(X), \\ nestsintrees(X) \prec flies(X) \end{array} \right\}$$

From this program we can infer that, the answer for $flies(tina)$ will be `yes`, whereas the answer for $\sim flies(tina)$ will be `no`. The answer for $flies(tweety)$ will be `no`, whereas the answer for $\sim flies(tweety)$ will be `yes`. We can also get the sequence

$$\{chicken(tina), bird(tina), flies(tina)\}$$

is a defeasible derivation for $flies(tina)$, obtained for the following set of rules:

$$\{bird(tina) \leftarrow chicken(tina), flies(tina) \prec bird(tina)\}$$

Note that there is also a derivation for $\sim flies(tina)$ from the sequence $chicken(tina), \sim flies(tina)$ obtained form this other set of rules: $\{\sim flies(tina) \prec chicken(tina)\}$. Then there is an argument $\langle \mathcal{A}_1, flies(tina) \rangle$ and there is an argument $\langle \mathcal{A}_2, \sim flies(tina) \rangle$ where: $\mathcal{A}_1 = \{flies(tina) \prec bird(tina)\}$, and $\mathcal{A}_2 = \{\sim flies(tina) \prec chicken(tina)\}$

2.2 Shortcomings in $DeLP$'s Knowledge Representation Language

A fact that is not considered in argumentative systems is the dynamics of knowledge. On one hand there is some knowledge that never changes with time, this knowledge is usually about *"what things are"*. On the other hand, information that express *"how things are"* generally changes with time. As an example lets take a look to the $DeLP$ program presented on example 1. We can see that all the literals defined in P_1 are time independent. If something is a $bird$ it will always be one, since there is no way of changing the nature things. But the definition of $flies$ may depend on information that changes over time. As an example, we can consider the fact that squabs (young pigeons) do not fly. If we add this information in program P_1 above we face a temporal problem since it is based on the fact that some bird that is a squab just for some period of time will no longer be one because it will grow up. This kind of change in the truth-value of a literal, like $squab$, cannot be represented in $DeLP$'s language. Let see a program, $P_2 = (\Pi_2, \Delta_2)$, that considers the predicate $squab$ in the representation. Program P_2 is an extension of P_1 from example 1, where the new rules added are boxed. Let see the definition of the sets Π_2 and Δ_2:

$$\Pi_2 = \left\{ \begin{array}{l} bird(X) \leftarrow chicken(X), \\ bird(X) \leftarrow penguin(X), \\ \boxed{bird(X) \leftarrow squab(X)}, \\ \sim flies(X) \leftarrow penguin(X), \\ chicken(tina), \\ \boxed{squab(sylvester)}, \\ penguin(tweety), \\ scared(tina) \end{array} \right\} \qquad \Delta_2 = \left\{ \begin{array}{l} flies(X) \prec bird(X), \\ \sim flies(X) \prec chicken(X), \\ flies(X) \prec chicken(X), scared(X), \\ nestsintrees(X) \prec flies(X), \\ \boxed{\sim flies(X) \prec squab(X)} \end{array} \right\}$$

In P_2 the truth value of the literal $squab(X)$ should depend on time, a bird that is a squab now in a month will be a full grown pigeon, but in the above representation this fact is not modeled. The lacks of temporal expressivity in the language lead us to an inadequate representation, or, better said, to a representation that is a static snapshot of the world. We cannot represent in the rule $\sim flies(X) \prec squab(X)$ the moment when the query is made, although this information is extremely important for obtaining the right conclusion. If *sylvester* is a squab now it is correct to infer that *sylvester* can not fly, but to assume that *sylvester* is still a squab after one month of its birth is not right, and therefore it is incorrect to obtain that *sylvester* can not fly after that period of time, since *sylvester* is no longer a squab in the present moment. This change on the knowledge is not represented in the previous program. To achieve an argumentation system capable of an appropriate manipulation of this type of knowledge, as a first step, we should change the representation language.

3 CAUSAL LANGUAGES

During the last decades research in Logic Programming has developed in a significant manner. Logic programming is closely related with temporal reasoning [11] but the classical logical programs, based on Horn clauses [22], is not good enough at modeling change. An extended language is required and as a consequence a new computational approach, suitable for the new language it is also required. To overcome these limitations of traditional logic programming some non-logic constructors, annotations or especial predicates, were introduced. These languages and their implementations represent significant advances in the area. In this category there are very well known languages such as *Event Calculus* [10] and *Situation Calculus* [9, 19]. These languages are very efficient but they are based on a non-standard logic, which means that a program could not be interpreted only by its specification. Another way to avoid the limitations of traditional logic programming is the use of temporal logics,

and for this purpose modal and intentional logics are used. As a result many languages appeared, some are purely declarative while other have an associated operational semantics.

There are different ways to conceive time, conceptions that are borrowed from philosophy. We can think in linear or branching time, discrete or dense time, etc. Other aspect appears when we combine time an actions. We can think time as an entity were events take place, or we can think on events as a entity, and thus time can be seen only as a collateral phenomenon of the events occurrence. More precise information about different time conceptions can be found at [17, 20]. Taking this aspects on consideration, in this kind of reasoning we can choose different languages according to what time conception we need or how we interpret events and its relation with time.

Causality defines the law of cause an effect. If we reduce this definition to a way of determining when the validity of which fluents end after some other fluents become valid, almost every logical language can be seen as a causal language. Since time an actions are basic concepts on a cause and effect scenario, then any causal language should take into consideration these concepts. Any language considered causal has a way to represent action effects (effect axioms) and action un-effects (frame axioms). In this sense some of the languages mentioned above, such as the *Event Calculus*, are causal.

Effect axioms represent the core of any causal program, they set out what changes are performed as a consequence of an action; but what remains unchanged is not modeled by them. That is why we could say that effect axioms lead only to a partial set of the consequences we intuitively like to have.

We will need other type of axioms to deal with the frame problem. Since frame information is default, then default reasoning becomes an essential part of any causal language. This form of reasoning gives to us the means to reason with this other important type of information.

Languages that implement this kind of axioms can choose among several implementations of default reasoning, the best known are: Default Logic [18], Circumscription [8], Argumentation Systems [16]. However, they usually use circumscription because it is a mechanism closer to logic programming than the others mentioned. Recall that circumscription is a form of non-monotonic reasoning developed by John McCarthy [6, 7], that augments standard first-order predicate calculus with a second-order axiom.

DEFINITION **4** [6, 7] Let ϕ be a formula, in which predicate ρ appears. The *circumscription* of ϕ minimizing ρ, noted as $CIRC[\phi; \rho]$, is the second order formula,

$$\phi \wedge \sim\exists\, q\, [\phi(q) \wedge q < \rho\,]$$

Its basic idea is to minimize the extension of certain predicates. This process is usually referred to as *minimizing* the predicate.

Event Calculus

The *Event Calculus*(*EC*) was introduced in the 1980's by Kowalski and Sergot as a logic programming formalism to represent events and their effects [10]. Many dialects have been developed since then, *e.g.*, see [21, 13]. In the original language events initiates time periods during which properties hold. Once a property or *"fluent"* is initiated, it holds unless it is terminated by the occurrence of an event. In Kowalski and Sergot version, discrete time ontology was chosen to indicate changes. A particular extension of the language is required in order to represent continuous characteristics. Most known extensions of this calculus were developed by Shanahan [21].

In general *EC* is a logical mechanism capable of making inferences to determine *what is true when* from *what happens when* (knowledge about the state of the world) and *what actions do* (effect of an action on the world). The logical machinery includes arithmetic to set a relation between time references. The kind of arithmetic involved depends on the selected temporal ontology. The basic

ontology of the calculus are *actions*, also called *events*, *fluent* and *time points*. A *fluent* is anything whose truth value is subject to change over time. It could be a quantity such as *"temperature in a room"* or *"amount of liquid in a bottle"* whose numerical value is subject to variation, or a proposition such as *"it is sunny"* whose truth value change from time to time. The predicate deals basically with propositional fluents although the other ones are allowed in some dialects. Another important issue in the choice of the ontology is the choice of the predicates. The main predicates used on a simple version of *Event Calculus*, (*SEC*), are:

$happens(E,T)$:	E takes place on T.
$holdsAt(F,T)$:	F holds at T.
$initiates(E,F,T)$:	F starts to hold after E, and is not freed on $T+1$.
$terminates(E,F,T)$:	F ceases to hold after E at T.
$releases(E,F,T)$:	F is not subject to inertia after E at T
$initiallyP(F)$:	F holds form time zero.

where E represents events, T time moments and F fluents. The calculus complete axiomatization depends on the time ontology of choice. For example if we consider a discrete ontology, we can use the ontology presented by Mueller [14], or the more complete ontology from Miller and Shanahan research [13].

The reasoning mechanism uses circumscription to deal with default information and to solve incompleteness. The latest versions of this calculus use as reasoning technique a first-order logic automated theorem proving. Previous versions use propositional satisfiability or abductive logic programming.

4 CAUSAL ARGUMENTATION

We begin the design of a causal argumentative system by choosing a representation language that allows representing causal information and with this purpose we introduce the syntax of *Event Calculus*. The implementation aspects of this calculus will be matter of future research.

In any causal language, there are rules that allow the manipulation of frame information. These rules are known as the *commonsense rules of inertia* and they are, in certain way, considered in non causal argumentation approaches but the role they play there is quite different. The information available is about different moments of time. In this new context they are crucial to the construction of arguments and we must consider how they affect the argument comparison mechanism. This issue is important because when we use an inertial rule we are accepting that none of the actions that took place so far, change this particular fluent status.

4.1 The Role of the Commonsense Rules of Inertia

One the mayor issues that appear under this new information scenario is the role of the commonsense rules of inertia. These rules are responsible of a great number of conflicts among arguments. In commonsense reasoning this kind of rule are defeasible in nature since the information obtained through them holds only if no change occur. This kind of rule appears as a way of solving the *frame problem*. The meaning attached to these rules is like the following default rule: *"Normally, given an action or event and a fluent, the action does not affect the fluent"*, and at the moment they were introduced as device for representing the situation they could not became more formal. Causal languages, as formal frameworks, made possible their formalization through default reasoning, such as circumscription. They usually look like this:

$$[Holds(fluent, Result(a,s)) \leftrightarrow Holds(f,s)] \leftarrow\sim Affects(a,f,s)$$

but they change according to the syntax of the language and how the defaults are managed on it. Languages as the *Event Calculus* deal with these rules through the use of circumscription (see Section 4.) This is one of the possible solutions to manage default reasoning, although it is a decision that brings advantages and disadvantages.

Let consider an example with this kind of rules and rules with *ageing*. A rule considers ageing if changes the truth-value of a literal only by looking at the truth-value the same literal used to have, at some past time t, and the amount of time span between the present moment and that particular moment t.

EXAMPLE **2** Let us consider a flying bird scenario represented in program $P_3 = (\Pi_3, \Delta_3)$, with the predicate $injured$ conveniently added.

$$\Pi_3 = \left\{ \begin{array}{l} holdsAt(bird(X), T+1) \leftarrow holdsAt(bird(X), T), \\ holdsAt(bird(tina), 0), \\ holdsAt(injured(tina), 0) \end{array} \right\}$$

$$\Delta_3 = \left\{ \begin{array}{l} holdsAt(flies(X), T) \prec holdsAt(bird(X), T), \\ holdsAt(flies(X), T) \prec \sim holdsAt(injured(X), T), holdsAt(bird(X), T) \\ \sim holdsAt(flies(X), T) \prec holdsAt(injured(X), T) \\ holdsAt(injured(X), T+1) \prec holdsAt(injured(X), T), \\ \sim holdsAt(injured(X), T+5) \prec holdsAt(injured(X), T), \\ \sim holdsAt(injured(X), T+1) \prec \sim holdsAt(injured(X), T), \end{array} \right\}$$

From the program above we can infer both $flies(tina)$ and $\sim flies(tina)$ at moment 5, this happens as a consequence of inferring that *tina* is $injured$ and not $injured$ in that moment. Let see all the arguments for and against fluent $holdsAt(flies(tina), 5)$, figure 1 depicts some of these arguments. Supporting arguments are: $\langle \mathcal{A}_3, holdsAt(flies(tina), 5) \rangle$ and $\langle \mathcal{A}_4, holdsAt(flies(tina), 5) \rangle$ while $\langle \mathcal{A}_5, \sim holdsAt(flies(tina), 5) \rangle$ is an argument against, where:

$$\mathcal{A}_3 = \left\{ holdsAt(flies(tina), 5) \prec holdsAt(bird(tina), 5), \right\}$$

$$\mathcal{A}_4 = \left\{ \begin{array}{l} holdsAt(flies(tina), 5) \prec \sim holdsAt(injured(tina), 5), holdsAt(bird(tina), 5) \\ \sim holdsAt(injured(tina), 5) \prec holdsAt(injured(tina), 0) \end{array} \right\}$$

$$\mathcal{A}_5 = \left\{ \begin{array}{l} \sim holdsAt(flies(tina), 5) \prec holdsAt(injured(tina), 5), \\ holdsAt(injured(tina), 5) \prec holdsAt(injured(tina), 4), \\ holdsAt(injured(tina), 4) \prec holdsAt(injured(tina), 3), \\ holdsAt(injured(tina), 3) \prec holdsAt(injured(tina), 2), \\ holdsAt(injured(tina), 2) \prec holdsAt(injured(tina), 1), \\ holdsAt(injured(tina), 1) \prec holdsAt(injured(tina), 0) \end{array} \right\}$$

Argument $\langle \mathcal{A}_3, holdsAt(flies(tina), 5) \rangle$ supports that $tina$ flies because it is a bird. Argument $\langle \mathcal{A}_4, holdsAt(flies(tina), 5) \rangle$ also supports that $tina$ flies but in this case because it is no longer an $injured\ bird$. Finally, argument $\langle \mathcal{A}_5, \sim holdsAt(flies(tina), 5) \rangle$ says that $tina$ cannot fly because it is still $injured$; note that the argument applies unconditionally an inertial rule. We can see that argument $\langle \mathcal{A}_5, \sim holdsAt(flies(tina), 5) \rangle$ seems to be, at first sight, the argument with most knowledge, but this argument built upon a sub-argument that supports $holdsAt(injured(tina), 5)$. This sub-argument is based on two rules, one is a common defeasible rule while the other one is an inertial rule. On the other hand, argument $\langle \mathcal{A}_3, \sim flies(tina) \rangle$ is based on a sub-argument that supports $\sim holdsAt(injured(tina), 5)$, which is constructed by one rule that change the fluent status through a defeasible rule that express information ageing, *i.e.* a rule that changes the truth-value of the literal $injured$ because it has passed a considerable amount of time, during which the bird heals.

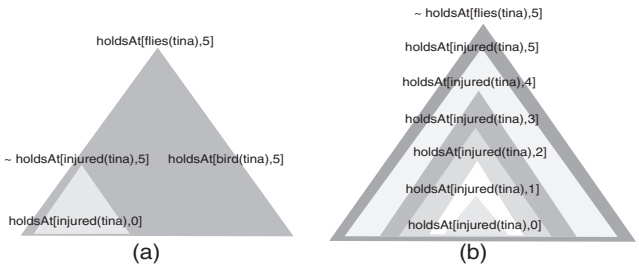

Figure 1: Arguments: (a):$\langle \mathcal{A}_4, holdsAt(flies(tina), 5)\rangle$ and (b): $\langle \mathcal{A}_5, \sim holdsAt(flies(tina), 5)\rangle$

From an informational point of view, argument $\langle \mathcal{A}_5, \sim holdsAt(flies(tina), 5)\rangle$ should be defeated by argument $\langle \mathcal{A}_3, \sim flies(tina)\rangle$. This aspect must be considered in the comparison criteria among arguments. Clearly, in this kind of scenario the specificity criterion is not good enough as comparison criterion among arguments. Multiple comparison criteria can be applied in this case, specificity in the general case and a particular criterion for arguments which involve the use of inertial rules. One of the aspects the criterion should take on consideration is that, in general, is less plausible to accept the inertial value of the fluent than the one obtained through other defeasible rules, for example those that considers ageing.

We will consider now another example. In this case inertia is stopped due to the occurrence of an event, instead of the natural ageing of information.

EXAMPLE **3** We add a literal $fall(X)$ that represents that X has fallen. The corresponding program is $P_4 = (\Pi_4, \Delta_4)$, where:

$$\Pi_4 = \left\{ \begin{array}{l} holdsAt(bird(X), T+1) \leftarrow holdsAt(bird(X), T), \\ holdsAt(bird(tina), 0), \\ \sim holdsAt(injured(tina), 0), \\ happens[fall(tina), 2) \end{array} \right\}$$

$$\Delta_4 = \left\{ \begin{array}{l} holdsAt(flies(X), T) \prec holdsAt(bird(X), T), \\ holdsAt(flies(X), T) \prec \sim holdsAt(injured(X), T), holdsAt(bird(X), T) \\ \sim holdsAt(flies(X), T) \prec holdsAt(injured(X), T) \\ holdsAt(injured(X), T+1) \prec holdsAt(injured(X), T), \\ \sim holdsAt(injured(X), T+5) \prec holdsAt(injured(X), T), \\ \sim holdsAt(injured(X), T+1) \prec \sim holdsAt(injured(X), T), \\ holdsAt(injured(X), T+1) \prec \sim holdsAt(injured(X), T), happens[fall(X), T) \end{array} \right\}$$

Analogously to the situation presented on the example above, (example 2) we can infer that at moment 3 $tina$ can fly and $tina$ cannot fly. We can also infer that $tina$ is $injured$ and $tina$ is not $injured$ at that moment. Lets see all the arguments for and against fluent $holdsAt(flies(tina), 3)$. Figure 2 depicts some of these arguments. Supporting arguments are: $\langle \mathcal{A}_6, holdsAt(flies(tina), 3)\rangle$ and $\langle \mathcal{A}_7, holdsAt(flies(tina), 3)\rangle$ while $\langle \mathcal{A}_8, \sim holdsAt(flies(tina), 3)\rangle$ is an argument against, where:

$$\mathcal{A}_6 = \quad \{ \ holdsAt(flies(tina), 3) \prec holdsAt(bird(tina), 3), \ \}$$

$$\mathcal{A}_7 = \quad \left\{ \begin{array}{l} holdsAt(flies(tina), 3) \prec \sim holdsAt(injured(tina), 3), holdsAt(bird(tina), 3) \\ \sim holdsAt(injured(tina), 3) \prec \sim holdsAt(injured(tina), 2), \\ \sim holdsAt(injured(tina), 2) \prec \sim holdsAt(injured(tina), 1), \\ \sim holdsAt(injured(tina), 1) \prec \sim holdsAt(injured(tina), 0) \end{array} \right\}$$

$$\mathcal{A}_8 = \left\{ \begin{array}{l} \sim holdsAt(flies(tina),3) \prec holdsAt(injured(tina),3), \\ holdsAt(injured(tina),3) \prec \sim holdsAt(injured(tina),2), happens(fall(tina),2) \\ \sim holdsAt(injured(tina),2) \prec \sim holdsAt(injured(tina),1), \\ \sim holdsAt(injured(tina),1) \prec \sim holdsAt(injured(tina),0) \end{array} \right\}$$

Argument $\langle \mathcal{A}_6, holdsAt(flies(tina),3)\rangle$ supports that $tina$ flies because it is a bird, as a matter of fact this argument is similar to $\langle \mathcal{A}_3, holdsAt(flies(tina),5)\rangle$ from the previous example. Argument $\langle \mathcal{A}_7, holdsAt(flies(tina),3)\rangle$ also supports that $tina$ flies but in this case because it is not an $injuredbird$ (through the application, several times, of the same inertial rule.) Note that the argument supports its conclusion applying unconditionally an inertial rule, *i.e.*, the rule is applied without further consideration of the effects of other possible events such as the occurrence of $fall(tina)$. Finally, argument $\langle \mathcal{A}_8, \sim holdsAt(flies(tina),3)\rangle$ supports that $tina$ cannot fly because it is *injured*; *tina* is *injured* because at time 2 has occurred that it *fells down*. Again in this case if we compare argu-

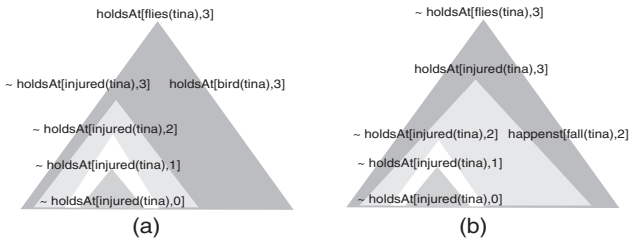

Figure 2: Arguments: (a):$\langle \mathcal{A}_7, holdsAt(flies(tina),3)\rangle$ and (b): $\langle \mathcal{A}_8, \sim holdsAt(flies(tina),3)\rangle$

ments $\langle \mathcal{A}_7, \sim holdsAt(flies(tina),3)\rangle$ and $\langle \mathcal{A}_8, \sim holdsAt(flies(tina),3)\rangle$, the second one should be preferred. The rationale is that the argument $\mathcal{A}_8$ is intuitively closer to reality from the available knowledge. Note that although $\langle \mathcal{A}_8, \sim holdsAt(flies(tina),3)\rangle$ is better we can see that it is using inertial rules as well.

4.2 $\mathcal{C}DeLP$ Definition

In the analysis made on examples 2 and 3 presented in the previous subsection, we argued about a particular argument comparison criteria. It became quite clear that *defeasible inertial rules* must be separated from the set of defeasible rules. This lead us to a change in the basic definitions of $DeLP$ in order to introduce $\mathcal{C}DeLP$ (Clausal $DeLP$).

DEFINITION **5** ($\mathcal{C}DeLP$)

A $\mathcal{C}DeLP$ program is defined in terms of three disjoint sets: a set Π of *facts* and *strict rules*, a set Δ of *defeasible rules*, and a set Υ of *inertial defeasible rules*, where

- A *fact* is a literal, *i.e.*, a ground atom, o a negated ground atom.
- A *strict rule* is a rule denoted as "$Head \leftarrow Body$", where $Head$ is a literal and $Body$, is finite set of literals. A strict rule can also be written as: $L_0 \leftarrow L_1, \ldots, L_n (n > 0)$, where each $L_i, i \geq 0$ is a literal.
- A *defeasible rule* is a rule noted as $L_0 \prec L_1, \ldots, L_n$. Again L_i is a literal and $i \geq 0$
- A *inertial defeasible rule* is a *defeasible rule* that denotes some fluent inertia.

Consequently, this change causes a change on the definition of argument:

DEFINITION **6** *$\mathcal{C}DeLP$ Causal Argument Structure*

Let L be a literal and $(P) = (\Pi, \Delta, \Upsilon)$ a $\mathcal{C}DeLP$ program. We say that $\langle \mathcal{A}, \mathcal{B}, L \rangle$ is a causal argument structure for L, if $\mathcal{A}$ is a set of defeasible rules from Δ and $\mathcal{B}$ is a set of defeasible rules form Υ, such that $\mathcal{A} \cup \mathcal{B}$ verifies $DeLP$ argument structure definition.

Looking at example 2 the program will be defined like this: $P_3 = (\Pi_3, \Delta'_3, \Upsilon_3)$, where Π_3 is the same set and $\Delta_3 = \Delta'_3 \cup \Upsilon_3$:

$$\Delta'_3 = \left\{ \begin{array}{l} holdsAt(flies(X), T) \prec holdsAt(bird(X), T), \\ holdsAt(flies(X), T) \prec \sim holdsAt(injured(X), T), holdsAt(bird(X), T) \\ \sim holdsAt(flies(X), T) \prec holdsAt(injured(X), T) \\ \sim holdsAt(injured(X), T+5) \prec holdsAt(injured(X), T), \end{array} \right\}$$

$$\Upsilon_3 = \left\{ \begin{array}{l} holdsAt(injured(X), T+1) \prec holdsAt(injured(X), T), \\ \sim holdsAt(injured(X), T+1) \prec \sim holdsAt(injured(X), T), \end{array} \right\}$$

and argument $\langle \mathcal{A}_5, \sim holdsAt(flies(tina), 5) \rangle$ will be now defined as $\langle \mathcal{D}, \mathcal{I} \sim holdsAt(flies(tina), 5) \rangle$ where:

$$\mathcal{D} = \left\{ \sim holdsAt(flies(tina), 5) \prec holdsAt(injured(tina), 5), \right\}$$

$$\mathcal{I} = \left\{ \begin{array}{l} holdsAt(injured(tina), 5) \prec holdsAt(injured(tina), 4), \\ holdsAt(injured(tina), 4) \prec holdsAt(injured(tina), 3), \\ holdsAt(injured(tina), 3) \prec holdsAt(injured(tina), 2), \\ holdsAt(injured(tina), 2) \prec holdsAt(injured(tina), 1), \\ holdsAt(injured(tina), 1) \prec holdsAt(injured(tina), 0) \end{array} \right\}$$

The above examples show how difficult the definition of a comparison criteria will result (see discussion below).

5 CONCLUSIONS AND FUTURE WORK

Argumentative systems such as $DeLP$ have been significant for the evolution of commonsense reasoning area. But they fall short of covering today's definition of this kind of reasoning. The main shortcoming is that the selected representation language, does not consider actions and time in an explicit way. If events and time, concepts present on any language considered as causal (based on the cause-effect principle), would behave as any other object in the language then it would be possible to use $DeLP$ unchanged without any further analysis. Unfortunately, these two elements have a completely different semantics. Events and when the events take place create a significant impact on the validity of certain properties.

In this work we changed $DeLP$ representation language introducing *Event Calculus* syntax and obtaining Causal $DeLP$ ($\mathcal{C}DeLP$). We presented the new definition of program and argument structure. These modifications were justified through the analysis made over two examples, which show that all defeasible rules cannot be considered with the same relevance, for example, *commonsense rules of inertia* are defeasible and important for commonsense reasoning but they are not as relevant as rules that express effect. As part of the analysis made we found out that although the definition of argument is similar, the comparison among them must be different. Taking on consideration the advances shown here, a complete causal argumentation system is being investigated.

A complete definition of the comparison criteria is needed. The analysis must take in consideration how the use of inertial rules are used in the construction of arguments and counterarguments. Clearly, an argument that uses inertial rules is weaker than an argument that uses only non-inertial defeasible rules. Nevertheless, some questions remain. What should be the result of comparing two arguments that use inertial rules? Is a longer chain of inertial rules weaker than a short one? What should be the decision when several different propagations of inertial information conflict?

REFERENCES

[1] Marcela Capobianco, Carlos Iván Chesñevar, and Guillermo Ricardo Simari. argumentation and the dynamics of warranted beliefs in changing environments. *Autonomouns Agents and Multi-Agent Systems*, 11(2):127–151, 2005.

[2] Carlos I. Chesñevar, Ana G. Maguitman, and Ron Loui. Logical models of argument. *ACM Computing Surveys*, 4(32):337–383, 2000.

[3] Carlos Iván Chesñevar, Guillermo Ricardo Simari, Lluis Godo, and Teresa Alsinet. Expansion operators for modelling agent reasoning in possiblistic defeasible programming. In *EUMAS*, pages 474–475, 2005.

[4] Alejandro J. Garcia and Guillermo R. Simari. Defeasible logic programming: an argumentative approach. *TPL*, 4:95–138, 2004.

[5] Steve Hanks and Drew Mc Dermott. Nonmonotonic logic and temporal projection. *Artificial Intelligence*, 33:379–412, 1987.

[6] John McCarthy. First order theories of individual concepts and propositions. In B. Meltzer and editors D. Michie, editors, *Machine Intelligence 9*, pages 120–147. Edinburgh Univerity Press, Edinburgh, 1979.

[7] John McCarthy. A Form of Non-Monotonic Reasoning. *Artificial Intelligence*, 13:27–39, 1980.

[8] John McCarthy. Applications of Circunscription to Formalizing commonsense Knowledge. *Artificial Intelligence*, 28:89–116, 1986.

[9] John McCarthy and Patrick J. Hayes. Some philosophical problems from the standpoint of artificial intelliegence. In B. Meltzer and editors D. Michie, editors, *Machine Intelligence 4*, pages 463–502. Edinburgh Univerity Press, Edinburgh, 1969.

[10] Robert Kowalski and Marek Sergot. A logic-based calulus of events. *New Generation Computing*, 4(1):67–895, 1986.

[11] John W. Lloyd. *Foundations of Logic Programming*. Springer-Verlag, New York, third edition edition, 1995.

[12] D. McDermott and J. Doyle. Nonmonotonic logic 1. *Artificial Intelligence*, 13:41–72, 1980.

[13] R. Miller and Murray Shannahan. Some alternative formulations of the event calculus. *Computational Logic: Logic Programming and Beyond*, 14:703–730, 2004.

[14] Erik T. Mueller. Event calculus reasoning through satisfiability. *Journal of Logic and Computation*, pages 452–490, 2002.

[15] Erik T. Mueller. *Commonsense Reasoning*. Morgan Kaufman an imprint of ELsevier, 2006.

[16] Henry Prakken and Gerard Vreeswijk. Logics for defeasible argumentation. In Dov Gabbay (ed.), editor, *Handbook of Philosophical Logic*. Kluwer Academic Publishers, 1998.

[17] Arthur Prior. *Past, Present and Future*. Clarendon Press, 1967.

[18] Ray Reiter. A Logic for Default-Reasoning. *Artificial Intelligence*, 13:81–132, 1980.

[19] Raymond Reiter. *Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems*. MIT Press, 2001.

[20] Nicholas Rescher and Alasdair Urquhart. *Temporal Logic*. Springer-Verlag, 1971.

[21] Murray Shanahan. Representing continuous change in the event calculus. In *Proceedings ECAI 90*, pages 598–603, 1990.

[22] M. van Emdem and Robert Kowalski. The Semantics of Predicate Logic as a Programming Language. *Journal of the Association for Computing Machinery*, 23(4):733–742, 1976.