Defeasible Logic Programming and Belief Revision

A Tutorial for the 20th ICLP

Guillermo R. Simari

Artificial Intelligence Research and Development Lab.
Dept. of Computer Science and Engineering
UNIVERSIDAD NACIONAL DEL SUR
ARGENTINA
http://cs.uns.edu.ar/~grs

Introduction

Research in Logic Programming, Nonmonotonic Reasoning, and Argumentation has obtained important results, providing powerful tools for knowledge representation and Common Sense reasoning.

We will introduce Defeasible Logic Programming (DeLP), a formalism that combines results of Logic Programming and Defeasible Argumentation.

Agenda

- Introduction
- Defeasible Logic Programming
- Brief Introduction to Belief Revision
- Explanations, Belief Revision and Defeasible Reasoning
- Brief List of References
Introduction

- DeLP adds the possibility of representing information in the form of weak rules in a declarative manner and a defeasible argumentation inference mechanism for warranting the conclusions that are entailed.
- Weak rules represent a key element for introducing defeasibility and they are used to represent a defeasible relationship between pieces of knowledge.
- This connection could be defeated after all things are considered.

DeLP’s Language

- DeLP considers two kinds of program rules: defeasible rules to represent tentative information such as
  \[
  \sim \text{flies}(\textit{dumbo}) \leftarrow \text{elephant}(\textit{dumbo})
  \]
  and strict rules used to represent strict knowledge such as
  \[
  \text{mammal}(\textit{idéfix}) \leftarrow \text{dog}(\textit{idéfix})
  \]
- Syntactically, the symbol “←” is all that distinguishes a defeasible rule from a strict one.
- Pragmatically, a defeasible rule is used to represent knowledge that could be used when nothing can be posed against it.
### Facts and Strict Rules

- A **Fact** is a ground literal: `innocent(joe)`
- A **Strict Rule** is denoted:
  
  \[ L_0 \leftarrow L_1, L_2, \ldots, L_n \]

  where \( L_0 \) is a ground literal called the **Head** of the rule and \( L_1, L_2, \ldots, L_n \) are ground literals which form its **Body**.

  This kind of rule is used to represent a relation between the head and the body which is not defeasible.

  Examples:

  \[ \sim \text{guilty(joe)} \leftarrow \text{innocent(joe)} \]
  \[ \text{mammal(garfield)} \leftarrow \text{cat(garfield)} \]

### Defeasible Rules

- Defeasible rules are not default rules.
- In a default rule such as \( \varphi : \psi_1, \psi_2, \ldots, \psi_n / \chi \) the justification part, \( \psi_1, \psi_2, \ldots, \psi_n \), is a consistency check that contributes in the control of the applicability of this rule.
- The effect of a defeasible rule comes from a dialectical analysis made by the inference mechanism.
- Therefore, in a defeasible rule there is no need to encode any particular check, even though could be done if necessary.
- Change in the knowledge represented using DeLP’s language is reflected with the sole addition of new knowledge to the representation, thus leading to better elaboration tolerance.

### Defeasible Rules

- A **Defeasible Rule** is denoted:
  
  \[ L_0 \leftarrow L_1, L_2, \ldots, L_n \]

  where \( L_0 \) is a ground literal called the **Head** of the rule and \( L_1, L_2, \ldots, L_n \) are ground literals which form its **Body**.

  This kind of rule is used to represent a relation between the head and the body of the rule which is tentative and its intuitive interpretation is:

  “**Reasons to believe in** \( L_1, L_2, \ldots, L_n \) **are reasons to believe in** \( L_0 \)”

  Examples:

  \[ \sim \text{good_weather(today)} \leftarrow \text{low_pressure(today)}, \text{wind(south)} \]
  \[ \text{flies(tweety)} \leftarrow \text{bird(tweety)} \]

### Defeasible Logic Program

- A **Defeasible Logic Program** (delp) is a set of facts, strict rules and defeasible rules denoted \( P = (\Pi, \Delta) \) where
  
  - \( \Pi \) is a set of facts and strict rules, and
  - \( \Delta \) is a set of defeasible rules.

  Facts, strict, and defeasible rules are ground.
  However, we will use “schematic rules” containing variables.
  If \( R \) is a schematic rule, \( \text{Ground}(R) \) stands for the set of all ground instances of \( R \) and

  \[ \text{Ground}(P) = \bigcup_{R \in P} \text{Ground}(R) \]

  in all cases the set of individual constants in the language of \( P \) will be used (see V. Lifschitz, *Foundations of Logic Programming*, in Principles of Knowledge Representation, G. Brewka, Ed., 1996, folli)
Here is an example of a **Defeasible Logic Program** denoted $P = (\Pi, \Delta)$

**\Pi (Strict Rules)**

- $\text{bird}(X) \leftarrow \text{chicken}(X) \quad \text{chicken}(tina)$
- $\text{bird}(X) \leftarrow \text{penguin}(X) \quad \text{penguin}(opus)$

**\Delta (Defeasible Rules)**

- $\text{flies}(X) \leftarrow \text{bird}(X)$
- $\sim \text{flies}(X) \leftarrow \text{chicken}(X)$
- $\text{flies}(X) \leftarrow \text{chicken}(X), \text{scared}(tina)$

**Facts**

$\text{Ground}(\text{flies}(X) \leftarrow \text{bird}(X)) = \{ \text{flies}(tina) \leftarrow \text{bird}(tina), \text{flies}(opus) \leftarrow \text{bird}(opus) \}$

Another example of a $P = (\Pi, \Delta)$

**\Delta (Defeasible Rules)**

- $\text{has\_a\_gun}(X) \leftarrow \text{lives\_in\_chicago}(X)$
- $\sim \text{has\_a\_gun}(X) \leftarrow \text{lives\_in\_chicago}(X), \text{pacificist}(X)$
- $\text{pacificist}(X) \leftarrow \text{quaker}(X)$
- $\sim \text{pacificist}(X) \leftarrow \text{republican}(X)$

**\Pi (Facts)**

- $\text{lives\_in\_chicago}(\text{nixon})$
- $\text{quaker}(\text{nixon})$
- $\text{republican}(\text{nixon})$

Adapted from Prakken and Vreeswijk (2000)

**Defeasible Derivation**

**Def:** Let $P = (\Pi, \Delta)$ be a delp and $L$ a ground literal. A **defeasible derivation** of $L$ from $P$, denoted $P \Downarrow L$, is a finite sequence of ground literals $L_1, L_2, \ldots, L_n = L$, such that each literal $L_k$ in the sequence is there because:

- $L_k$ is a fact in $\Pi$, or
- there is a rule (strict or defeasible) in $P$ with head $L_k$ and body $B_1, B_2, \ldots, B_n$ where every literal $B_i$ in the body is some $L_i$ appearing previously in the sequence ($i < k$).
Defeasible Derivation

Notice that defeasible derivation differs from standard logical or strict derivation only in the use of defeasible, or weak, rules.

Given a Defeasible Logic Program, a derivation for a literal \( L \) is called defeasible because there may exist information in contradiction with \( L \), or the way that \( L \) is derived, that will prevent the acceptance of \( L \) as a valid conclusion.

A few examples of defeasible derivation follow.

Defeasible Derivation

From the program:

\[
\begin{align*}
\text{bird}(X) &\leftarrow \text{chicken}(X) & \text{chicken}(tina) \\
\text{bird}(X) &\leftarrow \text{penguin}(X) & \text{penguin}(opus) \\
\neg \text{flies}(X) &\leftarrow \text{penguin}(X) & \text{scares}(tina) \\
\text{flies}(X) &\leftarrow \text{bird}(X) \\
\neg \text{flies}(X) &\leftarrow \text{chicken}(X) \\
\text{flies}(X) &\leftarrow \text{chicken}(X), \text{scares}(X)
\end{align*}
\]

The following derivations could be obtained:

\[
\begin{align*}
\text{chicken}(tina), \text{bird}(tina), \text{flies}(tina) \\
\text{chicken}(tina), \neg \text{flies}(tina) \\
\text{chicken}(tina), \text{scares}(tina), \text{flies}(tina) \\
\text{penguin}(opus), \text{bird}(opus), \text{flies}(opus) \\
\text{penguin}(opus), \neg \text{flies}(opus)
\end{align*}
\]

Defeasible Derivation

From the program:

\[
\begin{align*}
\text{buy_shares}(X) &\leftarrow \text{good_price}(X) \\
\neg \text{buy_shares}(X) &\leftarrow \text{good_price}(X), \text{risky}(X) \\
\text{risky}(X) &\leftarrow \text{in_fusion}(X, Y) \\
\text{risky}(X) &\leftarrow \text{in_debt}(X) \\
\neg \text{risky}(X) &\leftarrow \text{in_fusion}(X, Y), \text{strong}(Y) \\
\text{good_price}(acme) \\
\text{in_fusion}(acme, estron) \\
\text{strong}(estrон)
\end{align*}
\]

The following derivations could be obtained:

\[
\begin{align*}
\text{good_price}(acme), \text{buy_shares}(acme) \\
\text{in_fusion}(acme, estron), \text{risky}(acme), \text{good_price}(acme), \neg \text{buy_shares}(acme) \\
\text{in_fusion}(acme, estron), \text{risky}(acme) \\
\text{in_fusion}(acme, estron), \text{strong}(estrон), \neg \text{risky}(acme)
\end{align*}
\]

Programs and Derivations

A program \( P = (\Pi, \Delta) \) is contradictory if it is possible to derive from that program a pair of complementary literals.

Note that from the programs given as examples it is possible to derive pairs of complementary literals, such as \( \text{flies}(tina), \neg \text{flies}(tina) \) and \( \text{flies}(opus), \neg \text{flies}(opus) \) from the first one, and \( \text{risky}(acme), \neg \text{risky}(acme) \) and \( \text{buy_shares}(acme), \neg \text{buy_shares}(acme) \) from the second.

Contradictory programs are useful for representing knowledge that is potentially contradictory.

On the other hand, as a design restriction, the set \( \Pi \) should not be contradictory, because in that case the represented knowledge would be inconsistent.
Defeasible Logic Programming and Belief Revision – ICLP 2004

Defeasible Argumentation

**Def.** Let \( L \) be a literal and \( \mathcal{P} = (\Pi, \Delta) \) be a program. We say that \( \mathcal{A} \) is an *argument* for \( L \), denoted \( (\mathcal{A}, L) \), if \( \mathcal{A} \) is a set of rules in \( \Delta \) such that:

1) There exists a defeasible derivation of \( L \) from \( \Pi \cup \mathcal{A} \); and

2) The set \( \Pi \cup \mathcal{A} \) is non-contradictory; and

3) There is no proper subset \( \mathcal{A}' \) of \( \mathcal{A} \) such that \( \mathcal{A}' \) satisfies 1) and 2), that is, \( \mathcal{A} \) is minimal as the defeasible part of the derivation mentioned in 1).

That is to say, an argument \( (\mathcal{A}, L) \), or an argument \( \mathcal{A} \) for \( L \), is a minimal, non-contradictory set that could be obtained from a defeasible derivation of \( L \).

Stricts rules are not part of the argument.

Note that for any \( L \) which is derivable from \( \Pi \) alone, the empty set \( \emptyset \) is an argument for \( L \) (i.e. \( (\emptyset, L) \)).

In this case, there is no other argument for \( L \).

---

Example arguments:

- An argument for \( \neg \text{suspend}(john) \)
  built from the program above

  \[ \{ \neg \text{suspend}(john), \text{responsible}(john) \} \]

- An argument for \( \text{suspend}(peter) \)
  built from the program above

  \[ \{ \text{poor_perf}(peter), \text{sick}(peter) \} \]
Rebuttal and Defeat

Rebuttals or Counter-Arguments

- In DeLP, answers are supported by arguments but an argument could be defeated by other arguments.
- Informally, a query \( L \) will succeed if the supporting argument for it is not defeated.
- In order to study this situation, rebuttals or counter-arguments are considered.
- Counter-arguments are also arguments, and therefore this analysis must be extended to those arguments, and so on.
- This analysis is dialectical in nature.

Def: Let \( P = (\Pi, \Delta) \) be a program. We will say that two literals \( L_1 \) and \( L_2 \) disagree if the set \( \Pi \cup \{L_1, L_2\} \) is contradictory.

For example, given \( \Pi = \{ \sim L_1 \leftarrow L_2, L_1 \leftarrow L_3 \} \) the set \( \{L_2, L_3\} \) is contradictory.

Def: Let \( P = (\Pi, \Delta) \) be a program. We say that \( \langle A_1, L \rangle \) counter-argues, rebuts or attacks \( \langle A_2, L_2 \rangle \) at literal \( L \), if and only if there exists a sub-argument \( \langle A, L \rangle \) of \( \langle A_2, L_2 \rangle \) such that \( L \) and \( L_1 \) disagree.
Rebuttals or Counter-Arguments

- Given $P = (\Pi, \Delta)$, any literal $P$ such that $\Pi \vdash P$, has the support of the empty argument $\langle \emptyset, P \rangle$.
- Clearly, there is no possible counter-argument for any of those $P$ since there is no way of constructing an argument which would mention a literal in disagreement with $P$.
- On the other hand, any argument $\langle \emptyset, P \rangle$ cannot be a counter-argument for any argument $\langle A, L \rangle$ because of the same reasons.
- It is interesting to note that given an argument $\langle A, L \rangle$, that argument could contain multiple points where it could be attacked.
- Also, it would be very useful to have some preference criteria to decide between arguments in conflict.

Counter-argument

$\Pi \cup \{ \text{risky}(\text{acme}), \sim \text{risky}(\text{acme}) \}$

is a contradictory set

Defeaters

An argument $\langle B, P' \rangle$ is a proper defeater for $\langle A, L \rangle$ if $\langle B, P' \rangle$ is a counter-argument of $\langle A, L \rangle$ that attacks a subargument $\langle S, Q \rangle$ of $\langle A, L \rangle$ and $\langle B, P' \rangle$ is better than $\langle S, Q \rangle$ (by the chosen comparison criterion).
Defeasers

An argument \((B, P)\) is a proper defeater for \((A, L)\) if \((B, P)\) is a counter-argument of \((A, L)\) that attacks a subargument \((S, Q)\) of \((A, L)\) and \((B, P)\) is not comparable to \((S, Q)\) (by the chosen comparison criterion).

Blocking Defeater:
- ~suspend(peter)
- responsible(peter)
- good_perf(peter)
- suspend(peter)
- unrule(peter)

Defeaters

Argument Comparison: Generalized Specificity

Def: Let \(\mathcal{P} = (\Pi, \Delta)\) be a program. Let \(\Pi_G\) be the set of strict rules in \(\Pi\) and let \(\mathcal{F}\) be the set of all literals that can be defeasibly derived from \(\mathcal{P}\). Let \((A_i, L_i)\) and \((A_2, L_2)\) be two arguments built from \(\mathcal{P}\), where \(L_1, L_2 \in \mathcal{F}\).

Then \((A_1, L_1)\) is strictly more specific than \((A_2, L_2)\) if:

1. For all \(H \subseteq \mathcal{F}\), if there exists a defeasible derivation \(\Pi_G \cup H \cup A_1 \sim L_1\), while \(\Pi_G \cup H \not\sim L_1\) then \(\Pi_G \cup H \cup A_2 \sim L_2\), and

2. There is no pair of rules \(r \in A_1, r' \in A_2\) such that \(\Pi_G \cup H' \cup A_1 \not\sim L_1\), and \(\Pi_G \cup H' \cup A_2 \sim L_2\) but \(\Pi_G \cup H' \cup A_1 \not\sim L_1\).

(Poole, David L. (1985). On the Comparison of Theories: Preferring the Most Specific Explanation. pages 144—147 Proceedings of 9th IJCAI.)

Argument Comparison: Rule’s Priorities

Def: Let \(\mathcal{P} = (\Pi, \Delta)\) be a program, and let “\(>\)” be a partial order defined on the defeasible rules in \(\Delta\). Let \((A_i, L_i)\) and \((A_2, L_2)\) be two arguments obtained from \(\mathcal{P}\). We will say that \((A_i, L_i)\) is preferred to \((A_2, L_2)\) if the following conditions are verified:

1. If there exists at least a rule \(r_a \in A_1\) and a rule \(r_b \in A_2\) such that \(r_a > r_b\) and

2. There is no pair of rules \(r_a' \in A_1\) and \(r_b' \in A_2\) such that \(r_b' > r_a\).
**Argument Comparison: Rule’s Priorities**

From the program:

\[
\begin{align*}
\text{buy\_shares}(X) &\leftarrow \text{good\_price}(X) \\
\sim\text{buy\_shares}(X) &\leftarrow \text{risky}(X) \\
\text{risky}(X) &\leftarrow \text{in\_fusion}(X, Y)
\end{align*}
\]

with rule preference:

\[
\sim\text{buy\_shares}(X) \leftarrow \text{risky}(X) > \text{buy\_shares}(X) \leftarrow \text{good\_price}(X)
\]

argument \( \langle A, \sim\text{buy\_shares}(\text{acme}) \rangle \) where

\[
A = \{ \sim\text{buy\_shares}(\text{acme}) \leftarrow \text{risky}(\text{acme}), \\
\text{risky}(\text{acme}) \leftarrow \text{in\_fusion}(\text{acme}, \text{estron}) \}
\]

will be preferred to argument

\[
\langle B, \text{buy\_shares}(\text{acme}) \rangle \text{ where}
B = \{ \text{buy\_shares}(\text{acme}) \leftarrow \text{good\_price}(\text{acme}) \}
\]

**Deinters**

An argument \( \langle B, P \rangle \) is a defeater for \( \langle A, L \rangle \) if \( \langle B, P \rangle \) is a counter-argument for \( \langle A, L \rangle \) that attacks a subargument \( \langle S, Q \rangle \) de \( \langle A, L \rangle \) and one of the following conditions holds:

(a) \( \langle B, P \rangle \) is better than \( \langle S, Q \rangle \) (proper defeater), or

(b) \( \langle B, P \rangle \) is not comparable to \( \langle S, Q \rangle \) (blocking defeater)

**Deelters: Example**

From the program:

\[
\begin{align*}
\text{buy\_shares}(X) &\leftarrow \text{good\_price}(X) \\
\sim\text{buy\_shares}(X) &\leftarrow \text{risky}(X) \\
\text{risky}(X) &\leftarrow \text{in\_fusion}(X, Y)
\end{align*}
\]

With preference:

\[
\sim\text{buy\_shares}(X) \leftarrow \text{risky}(X) > \text{buy\_shares}(X) \leftarrow \text{good\_price}(X)
\]

The argument \( \langle A, \sim\text{buy\_shares}(\text{acme}) \rangle \) where

\[
A = \{ \sim\text{buy\_shares}(\text{acme}) \leftarrow \text{risky}(\text{acme}), \\
\text{risky}(\text{acme}) \leftarrow \text{in\_fusion}(\text{acme}, \text{estron}) \}
\]

is counter-argument of

\[
\langle B, \text{buy\_shares}(\text{acme}) \rangle \text{ where}
B = \{ \text{buy\_shares}(\text{acme}) \leftarrow \text{good\_price}(\text{acme}) \}
\]

that is a proper defeater of it.
Argumentation Lines

Given \( P = (\Pi, \Delta) \), and \( \langle A_0, L_0 \rangle \) an argument obtained from \( P \). An argumentation line for \( \langle A_0, L_0 \rangle \) is a sequence of arguments obtained from \( P \), denoted \( \Lambda = \langle \langle A_0, L_0 \rangle, \langle A_1, L_1 \rangle, \ldots \rangle \) where each element in the sequence \( \langle A_i, L_i \rangle, i > 0 \) is a defeater for \( \langle A_{i-1}, L_{i-1} \rangle \).

Argumentation Line

Given an argumentation line \( \Lambda = \langle \langle A_0, L_0 \rangle, \langle A_1, L_1 \rangle, \ldots \rangle \), the subsequence \( \Lambda_S = \langle \langle A_0, L_0 \rangle, \langle A_2, L_2 \rangle, \ldots \rangle \) contains supporting arguments and \( \Lambda_I = \langle \langle A_1, L_1 \rangle, \langle A_3, L_3 \rangle, \ldots \rangle \) are interfering arguments.
Let's consider a program $P$ where:

\[
\langle A_1, L_1 \rangle \text{ defeats } \langle A_0, L_0 \rangle
\]
\[
\langle A_2, L_2 \rangle \text{ defeats } \langle A_0, L_0 \rangle
\]
\[
\langle A_3, L_3 \rangle \text{ defeats } \langle A_1, L_1 \rangle
\]
\[
\langle A_4, L_4 \rangle \text{ defeats } \langle A_2, L_2 \rangle
\]
\[
\langle A_5, L_5 \rangle \text{ defeats } \langle A_2, L_2 \rangle
\]

Then, from $\langle A_0, L_0 \rangle$ there exist several argumentation lines such as:

\[
\Lambda_1 = [\langle A_0, L_0 \rangle, \langle A_1, L_1 \rangle, \langle A_3, L_3 \rangle]
\]
\[
\Lambda_2 = [\langle A_0, L_0 \rangle, \langle A_2, L_2 \rangle, \langle A_1, L_1 \rangle]
\]
\[
\Lambda_3 = [\langle A_0, L_0 \rangle, \langle A_2, L_2 \rangle, \langle A_3, L_3 \rangle]
\]

Argumentation Lines: Problems

- There are several undesired situations that could appear in argumentation lines.
- Let's see an example:

\[
\{ (d \sim b, c), (b \sim d, a), (\sim b \sim a), (\sim d \sim c), (a), (c) \}
\]

$\langle A_1, b \rangle = \{ (b \sim d, a), (\sim d \sim c) \}$ is a proper defeater of $\langle A_2, d \rangle = \{ (d \sim b, c), (\sim b \sim a) \}$ and reciprocally.

Note $\langle A_1, b \rangle$ is strictly more specific than the sub-argument $\langle B, \sim b \rangle = \{ (\sim b \sim a) \}$ of $A_2$ and $\langle A_2, d \rangle$ is strictly more specific than the sub-argument $\langle C, \sim d \rangle = \{ (\sim d \sim c) \}$ of $A_3$.

This will not be allowed since only defeaters could be introduced.

Nevertheless, in a more subtle way, it is possible to introduce a sub-argument of an argument that is already introduced.

When $\langle W, p \rangle$ is introduced, that action allows to reintroduce $\langle B, \sim p \rangle$ and that leads to circular argumentation.

The problem came from the introduction of argument $\langle W, p \rangle$. 
In the picture below, the argumentation line shows the problem created by reintroducing an argument. This argument started as a supporting argument and it is reintroduced as an interference argument. The problem appears when argument \(<C, \sim q>\) is introduced as a supporting argument, but it contains a counter-argument for the original argument. This leads to the notion of *concordance* in a line.

Given a program \(P = (\Pi, \Delta)\), we will say that \(<A_1, L_1>\) is concordant with \(<A_2, L_2>\) if and only if \(\Pi \cup A_1 \cup A_2\) is non-contradictory.

Given a program \(P = (\Pi, \Delta)\), we will say that an argumentation line \(\Lambda = [\langle A_0, L_0 \rangle, \langle A_1, L_1 \rangle, \ldots]\) will be acceptable if:

1. \(\Lambda\) is a finite sequence (no circularity).
2. The set \(\Lambda^S\) of supporting arguments is concordant, and the set \(\Lambda^I\) of interfering arguments is concordant.
3. There is no argument \(<A_i, L_i>\) in \(\Lambda\) that is a subargument of a preceding argument \(<A_i, L_i>\), \(i < k\).
4. For all \(i\), such that \(<A_i, L_i>\) is a blocking defeater for \(<A_{i+1}, L_{i+1}>\), if there exists \(<A_{i+1}, L_{i+1}>\) then \(<A_{i+1}, L_{i+1}>\) is a proper defeater for \(<A_i, L_i>\) (i.e., \(<A_i, L_i>\) could not be blocked).
Dialectical Tree

A Dialectical Tree is the conjoint representation of all the acceptable argumentation lines.

Given an argument \( A \) for a literal \( L \), the dialectical tree contains all acceptable argumentation lines that start with that argument.

In that manner, the analysis of the defeat status for a given argument could be carried out on the dialectical tree.

As every argumentation line is admissible, and therefore finite, every dialectical tree is also finite.
Marking of a Dialectical Tree

**Marking Procedure:** Let $T_{(A, L)}$ be a dialectical tree for $(A, L)$. The corresponding marked dialectical tree, $T^*_{(A, L)}$, will be obtained marking every node in $T_{(A, L)}$ as follows:

1. All leaves in $T_{(A, L)}$ are marked as $U$'s in $T^*_{(A, L)}$.
2. Let $(B, Q)$ be an inner node of $T_{(A, L)}$. Then $(B, Q)$ will be marked as $U$ in $T^*_{(A, L)}$ if and only if every child of $(B, Q)$ is marked as $D$ and the node $(B, Q)$ will be marked as $D$ if and only if it has at least a child marked as $U$.
Warranted Literals

Let $P = (\Pi, \Delta)$ be a defeasible program. Let $(A, L)$ be an argument and let $T^*_A(L)$ be its associated dialectical tree. A literal $L$ is **warranted** if and only if the root of $T^*_A(L)$ is marked as "U".

That is, the argument $(A, L)$ is an argument such that each possible defeater for it has been defeated.

We will say that $A$ is a warrant for $L$.

Dialectical Tree: Pruning

If the strict part $\Pi$ of a program $P = (\Pi, \Delta)$ is inconsistent, any literal can be derived.

When it is possible to defeasible derive a pair of complementary literals $\{ L, \sim L \}$ it is possible to introduce a way to try to decide whether to accept one of them.

Therefore, there are three different possible answers: accept $L$, accept $\sim L$, or to reject both.

Also, if the program is used as a device to resolve queries, a fourth possibility appears: the literal for which the query is made is unknown to the program.

Answers in DeLP
Answers in DeLP

Given a program $\mathcal{P} = (\Pi, \Delta)$, and a query for $L$ the possible answers are:

- **YES**, if $L$ is warranted.
- **NO**, if $\neg L$ is warranted.
- **UNDECIDED**, if neither $L$ nor $\neg L$ are warranted.
- **UNKNOWN**, if $L$ is not in the language of the program.

Specification of the Warrant Procedure

\[
\text{warrant}(Q, A) :-
\text{find_argument}(Q, A),
\text{\ plus defeated}(A, [\text{support}(A, Q)]).
\]

\[
\text{defeated}(A, ArgLine) :-
\text{find_defeater}(A, D, ArgLine),
\text{acceptable}(D, ArgLine, NewLine),
\text{\ plus defeated}(D, NewLine).
\]

Extensions and Applications

Adding *not*

- DeLP program rules can contain *not* as in
  - $\neg \text{cross_railway_tracks} \leftarrow \neg \text{train_is_coming}$
  - $\neg \text{cross_railway_tracks} \leftarrow \text{cannot_wait, not } \neg \text{train_is_coming}$

- Is very simple to extend the notions of defeasible derivation, argument and counter-argument.

- If *not* $L$ is a literal used in the body of a rule, there is a new kind of attack on it, *i.e.* if we have an undefeated argument for $L$ then the argument that contains a rule with *not* $L$ will be defeated.
Extending generalized specificity allowing utility values for facts and rules, giving the possibility of introducing pragmatic considerations.

Decision-Theoretic Defeasible Logic Programming will be represented as $P = (\Pi, \Delta, \Phi, B)$, where $\Pi$ and $\Delta$ are as before, $B$ is a Boolean algebra with top $\top$ and bottom $\bot$, and $\Phi$ is defined $\Phi: \Pi \cup \Delta \to B$.


We just got the second place in the Robocup e-league using Prolog (see http://cs.uns.edu.ar/~gis/robocup-TDP.htm). Now we are extending DeLP in a way of controlling the robots.

An action $A$ will be an ordered triple $(X, P, C)$, where $X$ is a consistent set of literals representing consequences of executing $A$, $P$ is a set of literals representing preconditions for $A$, $C$ is a set of constraints of the form $not L$, where $L$ is a literal.

Actions will be denoted:

$$\{X_1, \ldots, X_n\} \leftarrow\rightarrow \{P_1, \ldots, P_m\}, \text{not} \{C_1, \ldots, C_l\}$$

where $\text{not} \{C_1, \ldots, C_l\}$ means $\{\text{not} C_1, \ldots, \text{not} C_l\}$ and $\text{not} C_i$ means $C_i$ is not warranted.

$$\{\text{water}_\text{garden(today)}\} \leftarrow\rightarrow \{\text{not} \text{rain(today)}\}, \text{not} \{\text{rain(X)}\}$$


Implementation issues considering world dynamics.

The set of agent’s beliefs is formed by the warranted literals, i.e., those literals that are supported by an undefeated argument.

As an agent receive new perceptions, beliefs could change.

Because the process of calculating the new warrants is computationally hard we have developed a system to integrate precompiled knowledge in DeLP to address real time constrains for belief change. Our goal is to avoid re-computing arguments.

Belief Revision and Defeasible Reasoning

What is the motivation of belief revision?
To model the Dynamics of Knowledge

How can we do that?
- Classical Logic
- Selection Mechanism
- Non-classical Logic

An Example

From the following beliefs

- *The bird caught in the trap is a swan*
- *The bird caught in the trap comes from Sweden*
- *Sweden is part of Europe*
- *All European swans are white*

It can be inferred that

- *The bird caught in the trap is white*

Now, new information arrives:

- *The bird caught in the trap is black*

What it should be thrown away?

(Example due Peter Gärdenfors and Hans Rott, Belief Revision. Handbook of Logic in Artificial Intelligence and Logic Programming, vol. 4, 1995)
Epistemic Models

- **Belief Sets:**
  Sets of sentences closed under logical consequence.

- **Belief Bases:**
  Arbitrary sets of sentences.

Epistemic Attitudes

Let $K$ be a consistent belief base and let $\alpha$ be a sentence.

- $\alpha$ is **accepted** when $\alpha \in Cn(K)$
- $\alpha$ is **rejected** when $\sim \alpha \in Cn(K)$
- $\alpha$ is **indetermined** when $\alpha \not\in Cn(K)$ and $\sim \alpha \not\in Cn(K)$

If $K$ is inconsistent then every sentence is accepted (and rejected).

Operations

**Expansion ($+$):** Allows to transform indetermained sentences in accepted or rejected:

  a) If is $\alpha$ indetermined in $K$ then $\alpha$ is accepted in $K + \alpha$
  b) If is $\alpha$ indetermined in $K$ then $\alpha$ is rejected in $K + \sim \alpha$

**Contraction ($-$):** Allows to transform accepted or rejected sentences in indetermined:

  a) If is $\alpha$ accepted in $K$ then $\alpha$ is indetermined in $K - \alpha$
  b) If is $\alpha$ rejected in $K$ then $\alpha$ is indetermined in $K - \sim \alpha$

**Revision ($\ast$):** Allows to transform sentences accepted in rejected and to transform rejected sentences in accepted:

  a) If is $\alpha$ accepted in $K$ then $\alpha$ is rejected in $K \ast \sim \alpha$
  b) If is $\alpha$ rejected in $K$ then $\alpha$ is accepted in $K \ast \alpha$

How can they be defined?

Two possibilities have been introduced:

- **Levi Identity:** $K \ast \alpha = (K - \sim \alpha) + \alpha$
- **Harper Identity:** $K - \sim \alpha = K \cap K \ast \sim \alpha$
Contraction Postulates

Let $K$ be a Belief Set.

Closure: $K \vdash \alpha$ is a belief set.

Inclusion: $K \vdash \alpha \subseteq K$

Vacuity: if $\alpha \notin K$ then, $K \vdash \alpha = K$

Success: if $G \alpha$ then $\alpha \notin K$

Recovery: if $\alpha \in K$ then, $K \subseteq (K \vdash \alpha) + \alpha$

Equivalence: if $\alpha \leftrightarrow \beta$ then $K \vdash \alpha = K \vdash \beta$

Partial Meet Contraction

Construction:

- $K \perp \alpha = \{ H: H \subseteq K, \alpha \notin Cn(H) \text{ and for all } H \subseteq H' \subseteq K \text{ then } \alpha \in Cn(H') \}$
- $K \vdash \alpha = \cap \gamma(K \perp \alpha)$

Selection Function

- If $K \perp \alpha \neq \emptyset$, then $\gamma(K \perp \alpha) \neq \emptyset$
- Otherwise $\gamma(K \perp \alpha) = K$

Example:

- $K = \{ a, b, a \land b \rightarrow c, d \}$
- $K \perp c = \{ K_1, K_2, K_3 \} = \{ \{ a, b, d \}, \{ a, a \land b \rightarrow c, d \}, \{ b, a \land b \rightarrow c, d \} \}$
- Some possible results of $K \vdash c$:
  - $\{ a, b, d \} \Rightarrow \gamma(K \perp c) = \{ K_1 \}$
  - $\{ a, d \} \Rightarrow \gamma(K \perp c) = \{ K_1, K_3 \}$
  - $\{ a \land b \rightarrow c, d \} \Rightarrow \gamma(K \perp c) = \{ K_2, K_3 \}$
  - $\{ d \} \Rightarrow \gamma(K \perp c) = \{ K_1, K_3, K_3 \}$

Kernel Contraction

Kernel mode:

- Let $K$ be a set of sentences and $\alpha$ be a sentence.
- We found all minimal subsets of $K$ implying $\alpha$ (called $\alpha$-kernels).
- We “cut” the $\alpha$-kernels by means of an incision function $\sigma$ and then we eliminate the cut set from $K$. 
**Kernel Contraction**

<table>
<thead>
<tr>
<th>Construction</th>
<th>Incision Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>$K \perp \alpha = { H : H \subseteq K, \alpha \in Cn(H) } \land \text{for all } H \subseteq H \text{ then } \alpha \not\in Cn(H') } $</td>
<td>$\sigma(K \perp \alpha) \subseteq \cup K \perp \alpha $</td>
</tr>
<tr>
<td>$K + \alpha = K \setminus \sigma(K \perp \alpha)$</td>
<td>if $x \in K \perp \alpha$ and $x \not\in \emptyset$, then $x \not\in \sigma(K \perp \alpha) \subseteq \emptyset$</td>
</tr>
</tbody>
</table>

**Example:**

- $K = \{ a, a \rightarrow c, b, b \rightarrow c, d, \sim e \}$
- $K + c = \{ a, a \rightarrow c, \{ b, b \rightarrow c \} \}$

Some possible results of $K \setminus c$:

<table>
<thead>
<tr>
<th>$K \setminus c$</th>
<th>$\sigma(K \perp c)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>${ a \rightarrow c, b \rightarrow c, d, \sim e }$</td>
<td>${ a, b }$</td>
</tr>
<tr>
<td>${ a, b \rightarrow c, d, \sim e }$</td>
<td>${ a \rightarrow c, b }$</td>
</tr>
<tr>
<td>${ b \rightarrow c, d, \sim e }$</td>
<td>${ a, a \rightarrow c, b }$</td>
</tr>
<tr>
<td>${ d, \sim e }$</td>
<td>${ a, a \rightarrow c, b, b \rightarrow c }$</td>
</tr>
</tbody>
</table>

**Controversial Postulates**

- Every construction of a change operator is characterized by postulates.
- In the AGM model, there are some controversial postulates.
- Contraction:
  - **Recovery**: $K \subseteq (K \perp \alpha) + \alpha$
- Revision:
  - **Success**: $\alpha \in K * \alpha$
  - **Consistency**: If $\alpha$ is consistent then $K * \alpha$ is consistent.

**Belief Bases**

There are two kinds of beliefs:

- **Explicit Beliefs**: all the sentences in the belief base.
- **Implicit Beliefs**: all sentences derived from the belief base.

The implicit beliefs are "explained" from more basic beliefs.
Explanations

An *explanans* justifies an *explanandum*.

- **Set of sentences**
- **A sentence**

Notation: $A \rightarrow \alpha$

Properties:
- **Deduction**: $A \vdash \alpha$
- **Consistency**: It is not the case that $A \perp$ ⊥
- **Minimality**: There is no set $A' \subset A$ such that $A' \vdash \alpha$
- **Informational Content**: It is not the case that $\alpha \vdash A$

Informational Content

- **It is not the case that** $\alpha \vdash A$

This postulate precludes the following cases:

**Self-explanation**:

$$\{ \alpha \} \rightarrow \alpha$$

**Redundancy**:

$$\{ \alpha \lor \beta, \alpha \lor \sim \beta \} \rightarrow \alpha$$

New change operators

- **We will define operators for revision with respect to an explanans (i.e., a set of sentences).**
- **The idea is the following:**
  - Instead of incorporating a sentence $\alpha$ we request an explanans $A$ for $\alpha$.
  - We add $A$ to $K$
  - Then, we restore consistency (Consolidation).

New change operators

- **$K$**
- **$A$** Explanans for $\alpha$

Possibly inconsistent state

$$(K \cup A) \perp \perp$$

$\alpha$ might not be accepted
Two kinds of Constructions

- Partial Meet Revision by a set of sentences:
  \[ K \cup A = (K \cup A) \perp \perp \]

- Kernel Revision by a set of sentences:
  \[ K \cup A = (K \cup A) \perp \perp \]

Different kinds of beliefs

- Particular Beliefs:
  \[ \text{car(ferrari)} \quad \text{bird(opus)} \]

- General Beliefs:
  \[ \forall x(\text{car}(x) \rightarrow \text{vehicle}(x)) \quad \forall x(\text{bird}(x) \rightarrow \text{flies}(x)) \]

The strategy: all beliefs removed in a change process are preserved in a different status.

Transformation of Beliefs

\[ \text{Transf} (\ (\forall x)(p(x) \rightarrow q(x)) \ ) \]

- Defeasible rule in Argumentative Systems:
  \[ p(x) \vdash q(x) \]

- Default rule in Default Theories:
  \[ \frac{p(x) : q(x)}{q(x)} \]

Epistemic Model

A knowledge structure \([K, \Delta]\) where:

- \(K\) is the **undefeasible knowledge**.
- \(\Delta\) is the **defeasible knowledge** represented by:
  - Defeasible conditionals in Argumentative Systems; or
  - Default rules in Default Theories.
Changes

\[ [K, \Delta] \circ A = [K', \Delta'] \]
where:

- \( K' = K \circ A \)
- \( \Delta' = \Delta \cup \{ \text{Transf}(\alpha) : \alpha \in K \setminus K \circ A \} \)

Example

- \( K = \{ \text{bird(tweety)}, \text{bird(opus)}, \forall x(\text{peng}(x) \rightarrow \text{bird}(x)), \forall x(\text{bird}(x) \rightarrow \text{fly}(x)) \} \)
- From \( K \) we may conclude that:
  \( \text{bird(tweety)}, \text{bird(opus)}, \text{fly(tweety)}, \text{fly(opus)} \)
- Then, we receive the next explanans \( A \) for \( \sim \text{fly(opus)} \):
  \( \{ \text{bird(opus)}, \text{peng(opus)}, \forall x(\text{peng}(x) \land \text{bird}(x) \rightarrow \sim \text{fly}(x)) \} \)

Example

- In order to obtain \( K \circ A \) we need to eliminate contradictions from \( K \cup A \).
  \( K \cup A = \{ \text{bird(tweety)}, \text{bird(opus)}, \text{peng(opus)}, \forall x(\text{peng}(x) \rightarrow \text{bird}(x)), \forall x(\text{bird}(x) \rightarrow \text{fly}(x)), \forall x(\text{peng}(x) \land \text{bird}(x) \rightarrow \sim \text{fly}(x)) \} \)
- We could give up particular or general beliefs.
- If we discard general beliefs, we could select the \textit{less specific} beliefs, for instance, \( \forall x(\text{bird}(x) \rightarrow \text{fly}(x)) \).
- Then, we have the following belief base:
  \( K \circ A = \{ \text{bird(tweety)}, \text{bird(opus)}, \forall x(\text{peng}(x) \rightarrow \text{bird}(x)), \text{peng(opus)}, \forall x(\text{peng}(x) \land \text{bird}(x) \rightarrow \sim \text{fly}(x)) \} \)
- From \( K \circ A \) we may conclude that:
  \( \text{bird(tweety)}, \text{bird(opus)}, \text{peng(opus)}, \sim \text{fly(opus)} \)
- We can’t conclude \( \text{fly(tweety)} \) even though it is consistent with \( K \).
- This problem can be solved if we preserve the defeasible conditional \( \text{bird}(x) \rightarrow \text{fly}(x) \) or the default rule \( \text{bird}(x) : \text{fly}(x) / \text{fly}(x) \).
**Example**

- That is, we have the following knowledge:

\[ K \circ A = \{ \text{bird}(\text{tweety}), \text{bird}(\text{opus}), \text{peng}(\text{opus}), \forall x (\text{peng}(x) \land \text{bird}(x) \rightarrow \neg \text{fly}(x)) \} \]

\[ \Delta = \{ \text{bird}(x) \Rightarrow \text{fly}(x) \} \]

- From \([K \circ A, \Delta]\) we can infer that:

\[ \text{bird}(\text{tweety}), \text{bird}(\text{opus}), \text{peng}(\text{opus}), \neg \text{fly}(\text{opus}), \text{fly}(\text{tweety}) \]

- We have a new epistemic model and a new set of epistemic attitudes.

**References for the work presented (Short List)**


---

**Two Interesting Surveys**


**References for the work presented (Short List)**