Introducing Generalized Specificity in Logic Programming

Frieder Stolzenburg, stolzen@uni-koblenz.de
Alejandro J. García, ajg@cs.uns.edu.ar
Carlos I. Chesñevar, cic@cs.uns.edu.ar
Guillermo R. Simari, grs@cs.uns.edu.ar

1 Universität Koblenz-Landau, Rheinau 1, 56075 Koblenz, GERMANY
2 Universidad Nacional del Sur, Av. Alem 1253, (8000) Bahía Blanca, ARGENTINA

Abstract. Most formalisms for representing common-sense knowledge allow incomplete and potentially inconsistent information. When strong negation is also allowed, contradictory conclusions can arise. A criterion for deciding between them is needed. The aim of this paper is to investigate an inherent and autonomous comparison criterion, based on specificity as defined in [19, 22]. In contrast to other approaches, we consider not only defeasible, but also strict knowledge. Our criterion is context-sensitive, i.e. preference among defeasible rules is determined dynamically during the dialectical analysis.

We show how specificity can be defined in terms of two different approaches: activation sets and derivation trees. This allows us to get a more syntactic criterion that can be implemented in a computationally attractive way. The resulting definitions may be applied in general rule-based formalisms. We present theorems linking both characterizations.

Finally, we discuss other frameworks for defeasible reasoning in which preference handling is considered explicitly.

Key words: defeasible reasoning; knowledge representation; logic programming; non-monotonic reasoning.

1 Introduction

1.1 Background

Formalisms for representing common-sense knowledge usually handle incomplete and potentially inconsistent information. In such formalisms, contradictory conclusions can arise, which prompts for a criterion for deciding between them. Several extensions of logic programming (LP), default reasoning systems, defeasible logics, and defeasible argumentation formalisms consider priorities over competing rules [2, 4, 5, 11, 13, 26], in order to decide between contradictory conclusions. However, these priorities must be supplied by the programmer, establishing explicitly relations between rules.

Another problem, pointed out by Dung and Son in [7], is that several formalisms “enforce” the principle of reasoning with specificity by first determining a set of priority orders between defaults rules of a set $D$, using the information given by a domain knowledge $K$. The problem is that the resulting semantics is rather weak, in the sense that priorities are defined independent of the set $E$ of evidence. Therefore, if the set $E$ changes, the previous fixed priorities could not behave as expected.

On the contrary, evidence-sensitivity can be naturally captured in argumentation-theoretic approaches as shown in [7, 9, 22] and also here.

In [7], a transformation from the proposed underlying language into extended logic programming [10] is given. However, this transformation encodes the specificity criterion with program rules, forcing re-encoding in the presence of changes in the program. In our approach specificity will be inferred directly from the program rules without any intermediate step. Our approach also takes into consideration the background knowledge $B$ that was assumed empty in [7]. Dealing with background knowledge (strict rules) is not a trivial matter, because taking it into account may lead to more logical consequence for a given argument than before, which have to be considered in the dialectical process (see also Section 3.3).

* This research has been supported by the German-Argentinian program on scientific and technological cooperation, funded by the Bundesministerium für Bildung und Forschung in Germany and the Secretaría de Ciencia y Tecnología in Argentina (see also [6]).
1.2 Motivation

The aim of this paper is to investigate beyond explicit comparison between rules, looking forward for a more autonomous comparison criterion, based on specificity as defined in [19, 22]. In contrast to other approaches, we consider not only defeasible, but also strict knowledge. In our setting, arguments will be basically defeasible proofs involving both defeasible and strict knowledge, which may support contradictory conclusions, so that a comparison criterion is needed to decide between them. Our criterion for comparing arguments, namely specificity, is context-sensitive. This means that preference among defeasible rules is determined dynamically during the dialectical analysis (see also the examples in Section 4.1).

We show how this criterion can be redefined in terms of two different approaches: activation sets and derivation trees. This allows us to get a more syntactic criterion that can be implemented in a computationally attractive way. The resulting definitions may be applied in arbitrary generic rule-based formalisms. As a basis of our presentation we will use defeasible logic programming (DeLP) [8, 9], where a comparison for arguments based on specificity is given. In DeLP (as in most defeasible logics and defeasible argumentation formalisms), there is a distinction between strict rules and defeasible rules. Specificity in DeLP takes into consideration both kind of rules.

Originally, this research has been motivated by the programming of autonomous agents for the RoboCup [24]. Since agents must be able to cope with contradictory knowledge, defeasible reasoning should be employed for agent programming. Defeasible logic programming is able to extend the logic-based approach for multi-agent systems as presented in [24].

This paper is organized as follows. First, in Section 2 we introduce the fundamentals of DeLP. In Section 3, a definition of generalized specificity will be given, and two computationally attractive ways of comparing arguments by means of specificity in a logic programming framework will be developed. Finally, in Section 4, we discuss other frameworks for defeasible reasoning in which preference handling is considered explicitly, contrasting them with our approach. We will end with concluding remarks in Section 5.

2 Defeasible Logic Programming

2.1 Defeasible Programs

The DeLP language [8, 9] is defined in terms of two disjoint sets of rules: a set of strict rules for representing strict (sound) knowledge, and a set of defeasible rules for representing tentative information. Rules will be defined using literals. A literal $L$ is an atom $p$ or a negated atom $\sim p$, where the symbol $\sim$ represents strong negation. We define this formally as follows:

**Definition 2.1** (strict and defeasible rules). A strict rule (defeasible rule) is an ordered pair, conveniently denoted by Head $\leftarrow$ Body (Head $\prec$ Body), whose first member, Head, is a literal, and whose second member, Body, is a finite set of literals. A strict rule (defeasible rule) with the head $L_0$ and body $\{L_1, \ldots, L_n\}$ can also be written as $L_0 \leftarrow L_1, \ldots, L_n (L_0 \prec L_1, \ldots, L_n)$. If the body is empty, it is written $L \leftarrow true (L \prec true)$, and it is called a fact (presumption). Facts may also be written as $L$.

The syntax of strict rules correspond to basic rules in logic programming [14], but we call them “strict” in order to emphasize the difference to the “defeasible” ones (see below). There is no contraposition for rules. Defeasible rules add a new representational capability for expressing a weaker link between the head and the body in a rule [22]. Syntactically, the symbol $\prec$ is all that distinguishes a defeasible rule from a strict one. Pragmatically, a defeasible rule is used to represent defeasible knowledge, i.e. tentative information that may be used if nothing could be posed against it.
**Definition 2.2 (defeasible logic program).** A defeasible logic program (DLP) is a finite set of strict and defeasible rules where literals may have variable or constant parameters. We do not consider the case with general function symbols here. If \( \mathcal{P} \) is a DLP, we will distinguish the subset \( \Pi \) of strict rules in \( \mathcal{P} \), and the subset \( \Delta \) of defeasible rules in \( \mathcal{P} \). When required, we will denote \( \mathcal{P} \) as \( (\Pi, \Delta) \).

**Example 2.3.** The following is a DLP where strict and defeasible rules have been separated for the convenience of presentation. It models a fragment of the soccer domain.

\[
\begin{array}{ll}
\Delta \\
\text{kick}(X) \leftarrow \text{player}(X) & \text{player}(X) \leftarrow \text{libero}(X) \\
\neg \text{kick}(X) \leftarrow \text{libero}(X) & \text{player}(X) \leftarrow \text{goalie}(X) \\
\text{kick}(X) \leftarrow \text{libero}(X), \text{eager}(X) & \neg \text{kick}(X) \leftarrow \text{goalie}(X) \\
& \text{eager}(<\text{diego}) \\
& \text{libero}(<\text{diego}) \\
& \text{goalie}(\text{oli})
\end{array}
\]

Nute’s defeasible logic [4, 17], recent extensions of defeasible logic [2, 16] and some defeasible argumentation formalisms [12, 20, 25] also make use of defeasible and strict rules for representing knowledge. However, in most of these formalisms a priority relation among rules must be explicitly given with the program in order to handle contradictory information. In DeLP, an argumentation formalism for deciding between contradictory goals is used.

### 2.2 Defeasible Derivations

Given a program \( \mathcal{P} \), a defeasible derivation for a literal \( h \) (abbreviated \( \mathcal{P} \vdash h \)) will be a finite set of strict and defeasible rules. A defeasible derivation is obtained like a SLD-derivation, as defined e.g. in [15], but considering the negation symbol \( \neg \) as part of the predicate name and not taking into consideration the type of the rule.

**Definition 2.4 (defeasible derivation tree).** Let a DLP \( \mathcal{P} \) and a literal \( h \) be given. We say that a query \( h \) holds in \( \mathcal{P} \) (abbreviated \( \mathcal{P} \vdash_T h \), or simply \( \mathcal{P} \vdash h \)) iff there is a defeasible derivation tree \( T \) for \( h \) from \( \mathcal{P} \). \( T \) is a finite, rooted tree (strictly speaking, an and-tree), where all nodes are labelled with literals, satisfying the following conditions:

1. The root node of \( T \) is labelled with \( h \).
2. For each node \( N \) in \( T \) that is labelled with the literal \( L \), there is a strict or defeasible rule with head \( L_0 \) and body \( \{L_1, \ldots, L_k\} \) in \( \mathcal{P} \), such that \( L = L_0\sigma \) for some ground variable substitution \( \sigma \), and the node \( N \) has exactly \( k \) children nodes, which are labeled with \( L_1\sigma, \ldots, L_k\sigma \).

**Example 2.5.** Revisiting Example 2.3, the query \( \neg \text{kick}(\text{oli}) \) has the following derivation \( \{\neg \text{kick}(\text{oli}) \leftarrow \text{goalie}(\text{oli}), \text{goalie}(\text{oli})\} \), the query \( \text{kick}(\text{oli}) \) has the following derivation \( \{\text{kick}(\text{oli}) \leftarrow \text{player}(\text{oli}), \text{player}(\text{oli}) \leftarrow \text{goalie}(\text{oli}), \text{goalie}(\text{oli})\} \). A derivation for \( \text{kick}(\text{diego}) \) is \( \{\text{kick}(\text{diego}) \leftarrow \text{player}(\text{diego}), \text{player}(\text{diego}) \leftarrow \text{libero}(\text{diego}), \text{libero}(\text{diego})\} \) (Figure 1a shows the corresponding derivation tree), whereas \( \neg \text{kick}(\text{diego}) \) has the derivation \( \{\neg \text{kick}(\text{diego}) \leftarrow \text{libero}(\text{diego}), \text{libero}(\text{diego})\} \) (see also Figure 1b).

Given the DLP of Example 2.3, in Example 2.5 we have just shown that it is possible to have defeasible derivations for two contradictory literals. Thus, a DLP may represent contradictory information. A defeasible logic program \( \mathcal{P} \) is contradictory iff it is possible to defeasibly derive from \( \mathcal{P} \) a pair of complementary literals wrt. strong negation. We will assume that in every DLP \( \mathcal{P} \) the set \( \Pi \) is non-contradictory. Otherwise problems as in extended logic programming will happen, and the corresponding analysis of the consequences has been done elsewhere [1, 10].
2.3 Arguments

The central notion of the DeLP formalism is the notion of an argument. Informally, an argument is a minimal and non-contradictory set of rules used to derive a conclusion. In DeLP, answers to queries will be supported by an argument. The formal definition follows.

**Definition 2.6 (argument).** Let $h$ be a literal and $\mathcal{P} = (\Pi, \Delta)$ be a DLP. An argument $\mathcal{A}$ for a literal $h$, also denoted $\langle \mathcal{A}, h \rangle$, is a subset of ground instances of defeasible rules of $\Delta$, such that:

1. There exists a defeasible derivation for $h$ from $\Pi \cup \mathcal{A}$,
2. $\Pi \cup \mathcal{A}$ is non-contradictory, and
3. $\mathcal{A}$ is minimal wrt. set inclusion (i.e. there is no $\mathcal{A}' \subset \mathcal{A}$ such that $\mathcal{A}'$ satisfies condition 1).

The literal $h$ will also be called the conclusion supported by $\mathcal{A}$. An argument $\langle \mathcal{B}, q \rangle$ is a sub-argument of $\langle \mathcal{A}, h \rangle$ iff $\mathcal{B} \subseteq \mathcal{A}$. Note that strict rules are not part of an argument. Observe also that condition 2 of the previous definition prevents the occurrence of “self-defeating” arguments [18].

**Example 2.7.** Using the DLP of Example 2.3, the literal $\neg k(diego)$ has the argument $\mathcal{A}_1 = \{\neg k(diego) \leftarrow l(diego)\}$ and the literal $k(diego)$ has two arguments, namely $\mathcal{A}_2 = \{k(diego) \leftarrow p(diego)\}$ and $\mathcal{A}_3 = \{(k(diego) \leftarrow l(diego), e(diego))\}$ whose derivation tree is shown in Figure 1c. The literal $\neg k(oli)$ has a derivation formed only by strict rules, so $\mathcal{A} = \emptyset$ is an argument for it. Note that $\mathcal{B} = \{k(oli) \leftarrow p(oli), g(oli)\}$ for $k(oli)$ (Example 2.5) is a defeasible derivation, although it is not an argument for this literal because $\Pi \cup \mathcal{B}$ is contradictory.

Given an argument $\mathcal{A}$ for a literal $q$, other arguments that contradict $\mathcal{A}$ (called rebuttals or counter-arguments) could exist. We say that $\langle \mathcal{A}_1, h_1 \rangle$ counter-argues $\langle \mathcal{A}_2, h_2 \rangle$ at a literal $h$ iff there exists a sub-argument $\langle \mathcal{A}, h \rangle$ of $\langle \mathcal{A}_2, h_2 \rangle$ such that the set $\Pi \cup \{h_1, h\}$ is contradictory. Therefore, a comparison criterion among arguments is needed (one will be introduced in the next section). Based on that criterion, the following notion can be introduced

**Definition 2.8 (defeater).** An argument $\langle \mathcal{A}_1, h_1 \rangle$ defeats $\langle \mathcal{A}_2, h_2 \rangle$ at literal $h$ iff there exists a sub-argument $\langle \mathcal{A}, h \rangle$ of $\langle \mathcal{A}_2, h_2 \rangle$ such that $\langle \mathcal{A}_1, h_1 \rangle$ counter-argues $\langle \mathcal{A}_2, h_2 \rangle$ at $h$, and one of the following conditions hold:

1. $\langle \mathcal{A}_1, h_1 \rangle$ is “better” (wrt. some preference criterion) than $\langle \mathcal{A}, h \rangle$; then $\langle \mathcal{A}_1, h_1 \rangle$ is a proper defeater of $\langle \mathcal{A}_2, h_2 \rangle$; or
2. $\langle \mathcal{A}_1, h_1 \rangle$ is unrelated by the given preference order to $\langle \mathcal{A}, h \rangle$; then $\langle \mathcal{A}_1, h_1 \rangle$ is a blocking defeater of $\langle \mathcal{A}_2, h_2 \rangle$.

It is interesting to note that an argument that does not involve defeasible rules cannot be defeated. To understand why, assume that $\mathcal{A}_2 = \emptyset$ is an argument for $h_2$ defeated by $\mathcal{A}_1$ for $h_1$. Since $\mathcal{A}_2 = \emptyset$,

---

1 We use parentheses for improving the readability of the set of rules here.
the only existing subarguments of $\mathcal{A}_2$ are also empty arguments $\langle \emptyset, q \rangle$ (i.e. $\Pi \vdash q$). Therefore $\mathcal{A}_1$ counterargues $\mathcal{A}_2$ at some $q$, i.e. $\Pi \cup \{h_1, q\} \vdash \bot$. Since $\Pi \vdash q$, it follows that $\Pi \cup \{h_1\} \vdash \bot$. But this implies that $\mathcal{A}_1$ does not satisfy condition 2 of definition 2.6 (contradiction).

In DeLP, a query $q$ will succeed if the supporting argument for it is ultimately not defeated. In order to answer the question whether $\mathcal{A}$ is a non-defeated argument, counter-arguments that could be defeaters for $\mathcal{A}$ are considered. Since defeaters are arguments, there may exist defeaters for the defeaters, and so on. In DeLP a complete dialectical analysis is performed constructing a tree of arguments called dialectical tree. We refer the interested reader to [9] for details on the dialectical process.

3 An Inherent Criterion for Comparing Arguments

3.1 Specificity

We will formally define a particular criterion called generalized specificity which allows to discriminate between two conflicting arguments. The next definition characterizes the specificity criterion, defined first in [19] and extended later to be used in the defeasible argumentation formalism of [21, 22]. Here, it is adapted to fit in the DeLP framework. Intuitively, this notion of specificity favors two aspects in an argument: it prefers an argument (a) with greater information content or (b) with less use of rules.

Continuing with Example 2.7, argument $\mathcal{A}_1$ will be considered more specific than $\mathcal{A}_2$, because $\mathcal{A}_1$ does not use the strict rule $\text{player}(X) \leftarrow \text{libero}(X)$ and hence is more direct. On the other hand, the argument $\mathcal{A}_2$ will be regarded as strictly more specific than $\mathcal{A}_1$, because $\mathcal{A}_2$ is based in more information (libero and eager). In other words, an argument will be deemed better than another if it is more precise or more concise.

Definition 3.1 (specificity). Let $\mathcal{P} = (\Pi, \Delta)$ be a program, $\Pi_G$ be the set of all strict rules in $\Pi$ which are not facts, $\mathcal{F}$ be the set of all literals that have a defeasible derivation from $\mathcal{P}$, and $\langle \mathcal{A}_1, h_1 \rangle$ and $\langle \mathcal{A}_2, h_2 \rangle$ be arguments, such that $\Pi \cup \{h_1, h_2\}$ is contradictory. Then, $\langle \mathcal{A}_1, h_1 \rangle$ is more specific than an argument $\langle \mathcal{A}_2, h_2 \rangle$ (denoted $\langle \mathcal{A}_1, h_1 \rangle \succeq \langle \mathcal{A}_2, h_2 \rangle$) iff for all $H \subseteq \mathcal{F}$ it holds: $\Pi_G \cup H \cup \mathcal{A}_1 \vdash h_1$ and $\Pi_G \cup H \not\vdash h_1$ implies $\Pi_G \cup H \cup \mathcal{A}_2 \vdash h_2$. According to [19], we define: $\langle \mathcal{A}_1, h_1 \rangle$ is strictly more specific than $\langle \mathcal{A}_2, h_2 \rangle$ (written $\langle \mathcal{A}_1, h_1 \rangle > \langle \mathcal{A}_2, h_2 \rangle$) iff $\langle \mathcal{A}_1, h_1 \rangle \succeq \langle \mathcal{A}_2, h_2 \rangle$ and $\langle \mathcal{A}_2, h_2 \rangle \not> \langle \mathcal{A}_1, h_1 \rangle$.

To understand Definition 3.1 better, observe that the set $\Pi_G$ does not contain facts, so the condition $\Pi_G \cup H \cup \mathcal{A}_1 \vdash h_1$ will hold usually only with some particular non-empty set $H$. We say that $H$ activates $\mathcal{A}_1$. The expression $\Pi_G \cup H \not\vdash h_1$ is called the non-triviality condition, because it is forcing the use of the set $\mathcal{A}_1$ for deriving $h_1$. Hence, Definition 3.1 may be read as: $\langle \mathcal{A}_1, h_1 \rangle$ is more specific than $\langle \mathcal{A}_2, h_2 \rangle$, if for each set $H$ that non-trivially activates $\mathcal{A}_1$, the same set $H$ activates $\mathcal{A}_2$.

Continuing with Example 2.7, argument $\mathcal{A}_1$ is strictly more specific than $\mathcal{A}_2$ (see below), because $\mathcal{A}_1$ does not use the strict rule $\text{player}(X) \leftarrow \text{libero}(X)$ and hence is more direct. Observe that the condition from Definition 3.1 holds: every set $H$ that activates $\mathcal{A}_1$ also activates $\mathcal{A}_2$. However, the set $H' = \{\text{player(diego)}\}$ activates $\mathcal{A}_2$, but does not activate $\mathcal{A}_1$, where $\mathcal{A}_1 = \{\neg\text{kick(diego)} \leftarrow \text{libero(diego)}\}$ and $\mathcal{A}_2 = \{\text{kick(diego)} \leftarrow \text{player(diego)}\}$. On the other hand, the argument $\mathcal{A}_3 = \{\text{kick(diego)} \leftarrow \text{libero(diego)}, \text{eager(diego)}\}$ is strictly more specific than $\mathcal{A}_1$, because it uses more information. Again, the condition holds and the set $H'' = \{\text{libero(diego)}\}$ is enough to activate $\mathcal{A}_1$ but does not activate $\mathcal{A}_3$.

The following example shows why comparing only rules may sometimes be unsatisfactory. Consider the following program:

$$\{(s(X) \leftarrow q(X)), q(a), (p(X) \leftarrow q(X)), (\neg p(X) \leftarrow q(X), s(X))\}$$
The rule $r_1 \vdash \neg p(X) \leftarrow q(X), s(X)$ is using more information than $r_2 \vdash p(X) \leftarrow q(X)$. Thus, in a system with priorities over rules such as [4, 17] (because there “longer” rules are preferred to “shorter” ones), it is expected to have that $r_1$ is preferred to $r_2$ ($r_1 > r_2$). Therefore, in such a system the conclusion $\neg p(a)$ will be preferred over $p(a)$. But in this case, it is not true that $\neg p(a)$ is using more information, because the rule $s(X) \leftarrow q(X)$ establishes a strict connection between $s(X)$ and $q(X)$ (every $q(X)$ is an $s(X)$). In DeLP, the argument $\mathcal{A} = \{ \neg p(a) \leftarrow q(a), s(a) \}$ for $\neg p(a)$ is not more specific than the argument $\mathcal{B} = \{ p(a) \leftarrow q(a) \}$, and vice versa.

In some systems [2, 16] the rules $r_1$ and $r_2$ can be left unrelated wrt. superiority, and then achieve the desired result. But note that, if the rule $s(X) \leftarrow q(X)$ is replaced with the fact $s(a)$ (i.e., there is no longer a connection between $s(X)$ and $q(X)$), then in DeLP the argument $\mathcal{A}$ will be strictly more specific than $\mathcal{B}$. However, in a system with fixed priorities over rules this automatic change in the behavior of the system is not possible. The priority (or superiority) relation has to be changed to produce the expected result.

### 3.2 Computing Specificity in DeLP

Definition 3.1 suggests to test all subsets $\mathcal{H} \subseteq \mathcal{F}$. If $\mathcal{F}$ contains $n$ elements, there are $2^n$ sets to be considered. Besides the exponential explosion problem, this definition might be considering sets of literals that are unrelated to the arguments being compared. In this section, we introduce a way of avoiding these problems, which will be continued in the next section.

**Definition 3.2 (pruned trees and argument completion).** Let $\mathcal{A}$ be an argument for a ground literal $h$ wrt. a program $\mathcal{P} = \Pi \cup \Delta$. We call the literal tree $T(\mathcal{A}, h)$ derivation tree pruned wrt. the argument $\langle \mathcal{A}, h \rangle$ iff it is the tree obtained from a derivation tree $T$ with $\Pi \cup \Delta \vdash_T h$, by deleting (a) all nodes in $T$ which occur below nodes labeled with head literals of defeasible rules $r \notin \mathcal{A}$ and (b) all nodes which dominate only leaf nodes labeled with presumptions $r' \notin \mathcal{A}$ (or instances thereof). A completion of an argument $\mathcal{A}$ for $h$, denoted $\mathcal{A}^c$, is then the set of defeasible and strict rules (without facts), that are used in $T(\mathcal{A}, h)$.

In order to illustrate Definition 3.2, let us revisit Example 2.3 and Figure 1c again. The derivation tree $T$ in Figure 1c makes only use of the rule $r = \langle \text{kick(diego)} \leftarrow \text{libero(diego)}, \text{eager(diego)} \rangle$. Thus, $T$ is already pruned wrt. the argument $\mathcal{A}_1 = \{ r \}$. If we prune $T$ wrt. $\mathcal{A}_2 = \{ \langle \text{kick(diego)} \leftarrow \text{player(diego)} \rangle \}$, which is also an argument for $h \equiv \text{kick(diego)}$, then all nodes below the root have to be deleted, because $r \notin \mathcal{A}_2$. Hence, $T(\mathcal{A}_2, h)$ simply consists of one node which is labeled with $h$. This pruned tree will not be considered when comparing arguments, because it contains no use of any defeasible rule. But things are not always that simple (see Example 3.7).

Note that $\mathcal{A}^c$ does not contain the facts used in the construction of the defeasible tree. The reason for that will become clear later. The set of ground literals of $\mathcal{A}^c$, denoted $\text{Lit}(\mathcal{A}^c)$, will be the set of all ground literals that occur in the antecedent or consequent of every rule in $\mathcal{A}^c$. For example, if $\mathcal{A}^c = \{ (h(t) \leftarrow a(t)), (a(t) \leftarrow b(t), c(t)), (b(t) \leftarrow d(t)) \}$ is an argument for $h(t)$ then $\text{Lit}(\mathcal{A}^c) = \{ a(t), b(t), c(t), d(t), h(t) \}$.

**Definition 3.3 (activation set).** Let $\mathcal{A}^c$ be a completed argument, and $\text{Lit}(\mathcal{A}^c)$ the corresponding set of literals. A set $U \subseteq \text{Lit}(\mathcal{A}^c)$ is an activation set wrt. $\mathcal{A}^c$, if $U \cup \mathcal{A}^c \vdash h$, and $U$ is minimal wrt. set inclusion (i.e., $\exists U' \subseteq U$ such that $U' \cup \mathcal{A}^c \vdash h$). We will call Act-sets($\mathcal{A}^c$) the set of all activation sets wrt. $\mathcal{A}^c$. The set $U$ is called a non-trivial activation set for $\mathcal{A}^c$ iff $U$ is an activation set for $\mathcal{A}^c$ and $U \cup \Pi_\mathcal{G} \not\vdash h$. We will call NTAct-sets($\mathcal{A}^c$) the set of all the non-trivial activation sets wrt. $\mathcal{A}^c$. 
Figure 2 shows an algorithm to compute all non-trivial activation sets wrt. \((\mathcal{A}, h)\). In order to avoid trivial activation sets we only check whether a defeasible rule has been used. Note that the first activation set for \(\mathcal{A}\) is \(h\) itself, and it is also a trivial one. As can be seen from the algorithm, the set of activation sets of an argument is easy to compute, just parsing the completed argument once. Specificity can thus be defined in a form such that we only need to consider the non-trivial activation sets.

**Input:** a completed argument \(\mathcal{A}'\) for \(h\).  
**Output:** NTAct-sets(\(\mathcal{A}'\))

1. A stack \(S\) is initialized with the pair \((\{h\}, \text{trivial})\).
2. NTAct-sets(\(\mathcal{A}'\)) is initially empty.
3. Repeat until \(S\) is empty
   (a) Select the first pair \((N, \text{type})\) in \(S\) and remove it from the stack.
   (b) If type is non-trivial then add \(N\) to NTAct-sets(\(\mathcal{A}'\)).
   (c) The element \(N\) will be formed by a set of literals \(l_1, \ldots, l_n\). For each literal \(l_i \in N\) that is a consequent of a rule \(r\) in \(\mathcal{A}'\), with no empty body, create a new activation set \(\mathcal{N}_j\) replacing \(l_i\) with the literals of the antecedent of \(r\). The type of \(\mathcal{N}_j\) is trivial only if the type of \(N\) is trivial and \(r\) is a strict rule. Otherwise the type of \(\mathcal{N}_j\) is non-trivial. Thus, for every literal \(l_i\) in \(N\) a new activation set can be created.
   (d) The new activation sets \(\mathcal{N}_j\) that were not previously expanded, are added to the top of \(S\).
4. Return NTAct-sets(\(\mathcal{A}'\))

Fig. 2. Computing non-trivial activation sets.

**Definition 3.4 (specificity revisited, preliminary version).** Let \(\mathcal{A}\) for \(h_1\) and \(\mathcal{B}\) for \(h_2\) be two arguments with contradictory conclusions, and \(\mathcal{A}'\) and \(\mathcal{B}'\) the respectively completed arguments. We say that \(\mathcal{A}\) is strictly more specific than \(\mathcal{B}\) iff

1. \(\forall U \in \text{NTAct-sets}(\mathcal{A}')\), \(U \cup \Pi_G \cup \mathcal{B} \vdash h_2\), and
2. \(\exists U' \in \text{NTAct-sets}(\mathcal{B}')\) such that \(U' \cup \Pi_G \cup \mathcal{A}' \not\vdash h_1\).

Given an argument \(\mathcal{A}\) for a literal \(h\) there is no unique \(\mathcal{A}'\) because there could be different rules in \(\Pi_G\) that can prove an antecedent of a defeasible rule in \(\mathcal{A}\). Nevertheless, the difference between two argument completions \(\mathcal{A}'\) and \(\mathcal{B}'\) lies only in the use of strict rules. Note that Definition 3.4 is equivalent to Definition 3.1 only if there is a unique completion for each argument. However, Definition 3.4 can be reformulated in terms of the following sets, that consider every possible completion for an argument:

\[
\text{Act-sets}(\mathcal{A}) = \bigcup_{i=1}^n \text{Act-sets}(\mathcal{A}'_i)\\
\text{NTAct-sets}(\mathcal{A}) = \bigcup_{i=1}^n \text{NTAct-sets}(\mathcal{A}'_i)
\]

**Definition 3.5 (specificity revisited, final version).** Let \(\mathcal{A}\) for \(h_1\) and \(\mathcal{B}\) for \(h_2\) be two arguments. We say that \(\mathcal{A}\) is strictly more specific than \(\mathcal{B}\) iff

1. \(\forall U \in \text{NTAct-sets}(\mathcal{A})\), \(U \cup \Pi_G \cup \mathcal{B} \vdash h_2\), and
2. \(\exists U' \in \text{NTAct-sets}(\mathcal{B})\) such that \(U' \cup \Pi_G \cup \mathcal{A} \not\vdash h_1\).

**Theorem 3.6.** The Definitions 3.1 (for \(\Rightarrow\)) and 3.5 are equivalent.

It is important to notice, that we cannot restrict our attention to derivations which only make use of the defeasible rules in the given argument \(\mathcal{A}\). We must take into consideration also related arguments pruned wrt. \(\mathcal{A}\) in Definition 3.2. Otherwise, Theorem 3.6 would not hold. Look at the following example:
Example 3.7. Let us consider the following program:
\[
\{(x \leftarrow a, f), c, d, e, (x \rightarrow a, b, c), (a \rightarrow d), (b \leftarrow e), (f \leftarrow e), (\neg x \rightarrow a, b)\}
\]

Clearly, \(\langle A, x \rangle\) with \(A = \{(x \rightarrow a, b, c), (a \rightarrow d), (b \leftarrow e)\}\) and \(\langle B, \neg x \rangle\) with \(B = \{(\neg x \rightarrow a, b), (a \rightarrow d), (b \leftarrow e)\}\) are arguments. By Definition 3.1, it holds, that \(A\) is not more specific than \(B\), because the strict rules (especially \(x \leftarrow a, f\)) together with \(A\) and \(\{d, f\}\) non-trivially activate \(x\), but the strict rules together with \(B\) and \(\{d, f\}\) do not activate \(\neg x\).

However, according to Definition 3.5 without taking derivations into account which use defeasible rules \(r \notin A\), we would have that \(\langle A, x \rangle\) is strictly more specific than \(\langle B, \neg x \rangle\), because \(\{d, f\}\) would not be considered as an activation set for \(A\). This means that \(A\) is more specific than \(B\). But \(B\) is not more specific than \(A\), because \(\{a, b\}\) non-trivially activates \(\neg x\), but not \(x\). The problem is that there are two arguments for \(x\), namely \(A\) and \(C = \{(a \rightarrow d), (f \leftarrow e)\}\), which makes the activation set \(\{d, f\}\) possible.

3.3 A (More) Syntactic Criterion for Specificity

In the previous section, we expressed specificity by means of activation sets. In this section, we will go one step further by defining specificity via the comparison of (sets of) derivations. For this, we will identify each defeasible derivation tree with its sets of paths in the tree.

Let \(N\) be a leaf node in a (possibly pruned) derivation tree \(T\). Then, we call the set consisting of the literal labeling \(N\) plus all literals labeling its ancestors except the root node the path in \(T\) through \(N\). Let \(T^P\) be the set of all paths in \(T\) through all leaf nodes \(N\). For example, the path sets for the derivation trees in Figure 1, which are already pruned wrt. the corresponding arguments, are (a) \(\{(\text{player(diego)}, \text{libero(diego)})\}\), (b) \(\{\{\text{libero(diego)}\}\}\) and (c) \(\{\{\text{libero(diego)}\}, \{\text{eager(diego)}\}\}\), respectively. With this notion of paths, we are able to give a (preliminary) syntactic definition of specificity as follows, by introducing the relation \(\trianglerighteq\).

Definition 3.8. Let \(T_1\) and \(T_2\) be two trees. We define \(T_1 \trianglerighteq T_2\) iff for all \(t_2 \in T_2^P\) there exists a \(t_1 \in T_1^P\) such that \(t_1 \subseteq t_2\).

As already observed in the previous section, an argument cannot always be identified with one unique derivation or completed argument, but with a set of those. Therefore, we will take this into account in our next definition.

Definition 3.9 (syntactic criterion). Let \(\langle A_1, h_1 \rangle\) and \(\langle A_2, h_2 \rangle\) be two contradictory arguments wrt. a program \(P = \Pi \cup \Delta\). Then, \(\langle A_1, h_1 \rangle \trianglerighteq \langle A_2, h_2 \rangle\) iff for all derivation trees \(T_1\) for \(h_1\) pruned wrt. \(A_1\) there is a tree \(T_2\) for \(h_2\) pruned wrt. \(A_2\) such that \(T_1 \trianglerighteq T_2\).

Now, we are able to state yet another formulation of specificity by means of the relation \(\geq\) in the subsequent theorem. It gives us a more syntactic characterization of specificity without guessing sets of possible facts. Note that the elements of \(\mathcal{F}\) in Definition 3.1 are also called possible facts.\(^2\)

Theorem 3.10. Let \(\langle A_1, h_1 \rangle\) and \(\langle A_2, h_2 \rangle\) be two arguments. Then: \(\langle A_1, h_1 \rangle \geq \langle A_2, h_2 \rangle\) implies \(\langle A_1, h_1 \rangle \trianglerighteq \langle A_2, h_2 \rangle\). If \(\Pi_G\) is empty, then also the converse holds: \(\langle A_1, h_1 \rangle \trianglerighteq \langle A_2, h_2 \rangle\) implies \(\langle A_1, h_1 \rangle \geq \langle A_2, h_2 \rangle\).

Proof. See [23]. \(\Box\)

\(^2\) This theorem repairs so to speak the (false) conjecture in [22, Lemma 2.24].
Example 3.11. To see the necessity of the restriction in the second part of the theorem, consider the program \( \{ (x \rightarrow a, b), (b \rightarrow c), (\neg x \rightarrow c, d), (d \rightarrow a, a, c, (\neg x \leftarrow b, d) \} \). For \( \mathcal{A} = \{ (x \rightarrow a, b), (b \rightarrow c) \} \) and \( \mathcal{B} = \{ (\neg x \rightarrow c, d), (d \rightarrow a) \} \), it holds \( \langle \mathcal{A}, x \rangle \geq \langle \mathcal{B}, \neg x \rangle \), because \( \{ a, b \} \) and \( \{ a, c \} \) are the only activation sets for \( \langle \mathcal{A}, x \rangle \), which also activate \( \langle \mathcal{B}, \neg x \rangle \). However, \( \langle \mathcal{A}, x \rangle \not\geq \langle \mathcal{B}, \neg x \rangle \), because there is only one derivation \( T_1 \) for \( x \), and there are two derivations \( T_2 \) for \( \neg x \) (pruned wrt. \( \mathcal{B} \)), but for both of them it holds \( T_1 \not\leq T_2 \).

As stated before in Section 2.3, the comparison of arguments is embedded into a dialectical process, where arguments may be defeated. There may even exist defeaters for the defeaters, and so on. In DeLP a complete dialectical analysis is performed constructing a tree of arguments. The syntactic criterion \( (\geq) \) for specificity defined above, can be used directly by the defeater notion (see Definition 2.8). Thus, the new definition for specificity can be embedded naturally in DeLP in a modular way.

4 Related Work

4.1 Argumentation

Dung and Son in [7] introduce an argumentation-theoretic approach to default reasoning with specificity. Default reasoning in general, and argumentative reasoning in particular, is defined in terms of a set \( E \) of evidence (or facts), and a pair \( K = (D, B) \) which represents the domain knowledge consisting of a set of default rules \( D \), and a first-order theory \( B \) representing background knowledge (\( \Delta \) and \( \Pi \_G \) in our notation). As stated before, our approach also takes into consideration the background knowledge \( B \) that was assumed empty in [7]. It is certainly interesting to consider a generalized setting, where evidence and background knowledge are not restricted to facts and strict rules, respectively. But this is beyond the scope of this paper.

In [7], the authors claim that most priority-based approaches define the semantics of \( T \) wrt. certain partial orders on \( D \), determined only by \( K \). Let \( POK \) be the set of all partial orders defined in this way. For every partial order \( \alpha \in POK \) (where \( (d, d') \in \alpha \) means that \( d \) has lower priority than \( d' \)), we define \( \langle \alpha \rangle \) to be a partial order between sets of defaults in \( D \), where \( S \prec \alpha S' \) means that \( S \) is preferred to \( S' \). Whatever the definition of \( \langle \alpha \rangle \), it has to satisfy the following property: Let \( S \) be a subset of \( D \), and let \( d, d' \) be two defaults in \( S \) such that \( (d, d') \in \alpha \). Then \( S \cup \{ d' \} \prec \alpha S \cup \{ d \} \).

The partial order \( \prec \alpha \) can be extended into a partial order between models in \( B \cup E \), by defining \( M \prec \alpha M' \) iff \( D_M \prec \alpha D_{M'} \), where \( D_M \) is the set of all defaults in \( D \) which are satisfiable in \( M \). A default \( p/q \) is satisfiable in \( M \) iff the implication \( p \rightarrow q \) is satisfiable in \( M \). A model \( M \) of \( B \cup E \) is a preferred model of \( T \) iff there exists a partial order \( \alpha \in POK \) such that \( M \) is minimal wrt. \( \prec \alpha \). [7] shows that any preferential semantics based on \( \prec \alpha \) is not satisfactory enough since the set of evidence \( E \) is not considered.

Example 4.1 (taken from [7]). Consider the default theory \( T = (E, K) \), where \( B = \emptyset \), \( D = \{ d/c, c/b, d/\neg a, b/a \} \), and \( E = \{ d \} \). The desirable semantics here is represented by the model \( M = \{ d, c, b, \neg a \} \). To have this semantics, most priority-based approaches assign the default \( b/a \) a lower priority than the default \( d/\neg a \). Let us consider \( T \) under a new set \( E = \{ d, \neg c, b \} \). Since \( c \) does not hold, the default \( d/\neg a \) cannot be considered more specific than the default \( b/a \), so that it should not be the case that either \( a \) or \( \neg a \) are concluded.

However, in any priority-based approach using the same priorities between defaults wrt. \( E \) and \( E' \), we have \( M = \{ \neg a, d, \neg c, b \} \prec \alpha M' = \{ a, d, \neg c, b \} \) since \( D_M = \{ c/b, d/\neg a \} \prec \alpha D'_{M'} = \{ c/b, b/a \} \) (due to the fact that \( (b/a, d/\neg a) \in \alpha \)). Hence priority-based approaches would conclude \( \neg a \) given
\((E', K)\), which is not the intuitive result, leading to the idea that default \(b/a\) should have a lower priority than \(d/\sim a\) under evidence \(E\), but a different priority under evidence \(E'\).

Example 4.1 can be recast into the DeLP formalism by rewriting a default rule \(a/b\) as a defeasible clause \(b \sim a\). Let us consider the preferred model associated with a DeLP program as defined by those literals supported by arguments ultimately undefeated. It turns out that the intuitively preferred model is computed correctly, since the evidence \(E\) is taken into account.

Example 4.2. Consider the set \(\Delta = \{ (a \sim b), (\sim a \sim d), (c \sim d), (b \sim c) \}\) of defeasible clauses, and let \(\Pi = \{ d \}\). In this case, we have arguments for \(b, c, d, a\) and \(\sim a\). The argument for \(\sim a\) is more specific than the argument for \(a\). However, if \(\Pi = \{ d, \sim c, b \}\), we will have still undefeated arguments for \(d, \sim c\) and \(b\), but \(\sim a\) will no longer hold (since it is blocked by the argument \(\{ a \sim b \}\)).

The previous example shows that in our approach, preference among defaults (defeasible rules) is determined dynamically during the dialectical analysis.\(^3\) A distinctive feature of specificity is that it can be generalized to other common-sense reasoning approaches where the notion of derivation plays a central role. Thus specificity results as a useful comparison criterion for choosing between conflicting extensions in proof-theoretic approaches, whereas the dialectical analysis determines whether a given extension (argument) is ultimately preferred.

Other argumentation formalisms—particularly those motivated by legal reasoning, such as [20]—consider priorities as well as defeasible reasoning about priorities. It must be remarked that in these cases criteria for comparing arguments are also debatable, and in many cases they are subordinated to hierarchical and temporal considerations (see [20] for an in-depth discussion). In contrast to these approaches, we concentrate on first finding an acceptable criterion for determining preferred extensions associated with the presence of defeasible information. Incorporating other features (such as hierarchical or temporal preference principles) is intended for further research.

4.2 Prioritizing Default Logic

Brewka [3] has extended default logic in order to handle priorities, developing a preferential default logic (PDL). This approach has many properties which seem relevant for argumentation, such as explicit representation of preferences and reasoning about preferences. Although this approach is not explicitly argument-based, prioritized default theories extend default theories adding a strict partial order on defaults, using this ordering to define preferred extensions.

A prioritized default theory \(\Delta = (D, W, <)\) extends the default theory \((D, W)\) with a strict partial order \(<\) on default rules. A default \(d\) will be considered preferred over default \(d'\) whenever \(d < d'\) holds. \(\Delta\) is called fully prioritized iff \(<\) is a well-ordering.

The following proposition can be established: if \(\Delta = (D, W, <)\) is a fully prioritized ground theory, and \(E\) a classical extension of \(\Delta\), then \(E\) is a preferred extension of \(\Delta\) iff for each default \(d \in D\) such that \(\text{pre}(d) \subseteq E\) and \(\text{cons}(d) \not\subseteq E\) there exists a set of defaults \(K_d \subseteq \{ d' \in GD(D, E) \mid d' < d \}\) such that \(d\) is defeated in \(\text{Th}(W \cup \text{cons}(K_d))\).\(^4\) Here \(GD(D, E)\) denotes the set of all defaults from \(D\) which are generating in \(E\) (a default \(d\) is called generating in a set of wffs \(S\) if \(\text{pre}(d) \subseteq S\) and \(\neg \text{just}(d) \cap S = \emptyset\)).

This proposition basically says that in preferred extensions defaults which are not applied must be defeated by defaults with higher priority.

PDL has a number of properties which seem to be relevant for defeasible argumentation, such as non-monotonicity, explicit representation of preferences and reasoning about preferences. However,

---

\(^3\) This implies that our approach is context-sensitive as defined in [7], although this is denied in the same reference.

\(^4\) Given a default \(d = a : b_1, \ldots, b_n/c\), where \(a, b_1, \ldots, b_n, c\) are first-order formulas, \(a\) is called the prerequisite of \(d\), each \(b_i\) is a justification, and \(c\) is the consequent. This is denoted as \(\text{pre}(d)\), \(\text{just}(d)\) and \(\text{cons}(d)\), respectively.
since an ordering of defaults is enforced, similar problems to those mentioned in section 4.1 are also present.

4.3 LP and Defeasible Logic with Superiority Relation

In [13] and later in [5], logic programming without negation as failure (LPwNF) was introduced. A LPwNF program consists of a set of basic rules $L_0 \leftarrow L_1, \ldots, L_n$ (where $L_i$ are literals that could use strong negation) and a given irreflexive and antisymmetric priority relation among program rules. They claim that default negation can be removed using the following transformation: the rule $r_0 : p \leftarrow q, \text{not } r$ is transformed to two rules, $r_1 : p \leftarrow q$ and $r_2 : \neg p \leftarrow r$, with $r_1 < r_2$. Hence, when $r$ is not derivable the rule $r_2$ cannot be used, and there is a derivation for $p$. On the other hand when $r$ is derivable, rule $r_2$ blocks $r_1$. However, the problem with this approach is that when $r$ is derivable, a new literal (not present in the original program) is derivable: $\neg p$. Contradiction between derivations is based on complementary literals, and the priority relation among rules. The proof procedure of LPwNF is very similar to the one of d-Prolog. Although in [5] there is no comparison with defeasible logic, in [2] a comparison among LPwNF, defeasible logic, and so-called courteous logic programs is given. The main result of [2] is that defeasible logic can prove everything that sceptical LPwNF can. In [11], Gelfond and Son developed a system to “investigate the methodology of reasoning with prioritized defaults in the language of logic programs under the answer set semantics”. Their system allows the representation of defeasible and strict rules, and the representation of an order among those rules. The way in which defeasible inferences are obtained is very similar to [2], although no comparison of these two systems is given.

In [2, 16], another approach for defeasible reasoning is presented. In this context, defeasible logic programs are (almost) identical to programs $\mathcal{P}$ as defined in Definition 2.2. But there, specificity is a relation between program clauses, modeled by the so-called superiority relation $>$, whereas in our framework specificity is an implicit relation between arguments according to Definition 3.1. The main difference is that this approach is not argument-based.

Since the relation $>$ must be explicitly given by the programmer in addition to the program $\mathcal{P}$, we have to consider the pair $(\mathcal{P}, >)$ for this approach. Since the procedure for deriving defeasibly valid literals is quite different from our approach, it is not clear how to express specificity as defined here by means of an appropriately chosen superiority relation. However, the construction of such a relation is a non-trivial issue, and deserves a more detailed analysis.

5 Conclusions

Formalisms for representing common-sense knowledge need to deal with contradictory conclusions, and decide between them with some comparison criterion. To our opinion, this comparison should be performed within the formalism itself by analyzing the pieces of knowledge which lead to contradictory conclusions. Thus, our aim was to look forward for an autonomous comparison criterion that may fit in any rule-based formalism.

As a result we characterized a generalized version of specificity, based on the comparison criterion defined in [19, 22]. We showed that specificity can be redefined in terms of two different approaches: activation sets (Theorem 3.6) and derivation trees (Theorem 3.10). A more syntactic criterion was obtained, which can be implemented in a computationally attractive way. This has been done in the DLP system (described in [8]). These results may be applied to other rule-based formalisms which currently make use of explicit priorities.

Further work will concentrate on investigating even deeper the relationships to other approaches and possible translations from one method of defeasible reasoning into another one. For instance,
it seems to be possible to reformulate defeasible reasoning as done here by means of (extended) logic programs (see also [7]). Last but not least, the integration of defeasible reasoning into agent programming should be tackled in greater detail.

References