

UNIVERSITÄT  
KOBLENZ · LANDAU



## **Introducing Generalized Specificity in Logic Programming**

Frieder Stolzenburg, Alejandro J. García,  
Carlos I. Chesñevar, Guillermo R. Simari

4/2000



Fachberichte  
**INFORMATIK**

---

Universität Koblenz-Landau  
Institut für Informatik, Rheinau 1, D-56075 Koblenz

E-mail: [researchreports@infko.uni-koblenz.de](mailto:researchreports@infko.uni-koblenz.de),

WWW: <http://www.uni-koblenz.de/fb4/>



# Introducing Generalized Specificity in Logic Programming<sup>\*</sup>

Frieder Stolzenburg, stolzen@uni-koblenz.de<sup>1</sup>

Alejandro J. García, ajg@cs.uns.edu.ar<sup>2</sup>

Carlos I. Chesñevar, cic@cs.uns.edu.ar<sup>2</sup>

Guillermo R. Simari, grs@cs.uns.edu.ar<sup>2</sup>

<sup>1</sup> Universität Koblenz-Landau, Rheinau 1, 56075 Koblenz, GERMANY

<sup>2</sup> 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. Therefore, a criterion for deciding between them is needed. Several extensions of logic programming consider priorities over program (default) rules. However, these priorities must be supplied by the programmer in a more or less arbitrary manner, establishing explicitly relations between rules.

The aim of this paper is to investigate beyond explicit comparison between rules, looking for an inherent and autonomous comparison criterion, based on specificity as defined in [22, 25]. In contrast to other approaches, we consider not only defeasible, but also strict knowledge. Our criterion for comparing arguments, namely *specificity*, is context-sensitive. This means that preference among defeasible rules is determined dynamically during the dialectical analysis.

We show how such a specificity criterion 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 a generic rule-based formalism. We present a theorem which links both characterizations, showing their equivalence. Finally we discuss other frameworks for defeasible reasoning in which preference handling is considered explicitly, contrasting them with our approach.

**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,

---

<sup>\*</sup> 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 [8]).

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, 6, 7, 13, 15, 28], 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 [9], is that several formalisms “enforce” the principle of reasoning with specificity by first determining a set of priority orders between default 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, this evidence-sensitivity can be naturally captured in argumentation-theoretic approaches as shown in [9, 11, 25] and also here.

In [9], a transformation from the proposed underlying language into extended logic programming [12] 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 [9]. 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).

## 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 [22, 25]. 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) [10, 11], 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 [26]. 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 [26].

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 [10, 11] 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  $\{L_1, \dots, L_n\}$  can also be written as  $L_0 \leftarrow L_1, \dots, L_n$  ( $L_0 \prec L_1, \dots, L_n$ ). 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 [17], 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 [25]. 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  $P$  is a DLP, we will distinguish the subset  $\Pi$  of strict rules in  $P$ , and the subset  $\Delta$  of defeasible rules in  $P$ . When required, we will denote  $P$  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.

$\Delta$	$\Pi$
$kick(X) \prec player(X)$	$player(X) \leftarrow libero(X)$
$\sim kick(X) \prec libero(X)$	$player(X) \leftarrow goalie(X)$
$kick(X) \prec libero(X), eager(X)$	$\sim kick(X) \leftarrow goalie(X)$
	$eager(diego)$
	$libero(diego)$
	$goalie(oli)$

Nute’s defeasible logic [6, 20], recent extensions of defeasible logic [2, 19], and some defeasible argumentation formalisms [14, 23, 27],

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  $P$ , a *defeasible derivation* for a literal  $h$  (abbreviated  $P \vdash h$ ) will be a finite set of strict and defeasible rules. A defeasible derivation is obtained like a SLD-derivation, as defined e. g. in [18], but considering the negation symbol  $\sim$  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  $P$  and a literal  $h$  be given. We say that a query  $h$  holds in  $P$  (abbreviated  $P \vdash_T h$ , or simply  $P \vdash h$ ) iff there is a defeasible derivation tree  $T$  for  $h$  from  $P$ .  $T$  is a finite, rooted tree (strictly speaking, an and-tree), where all nodes are labelled with literals, satisfying the following conditions:*

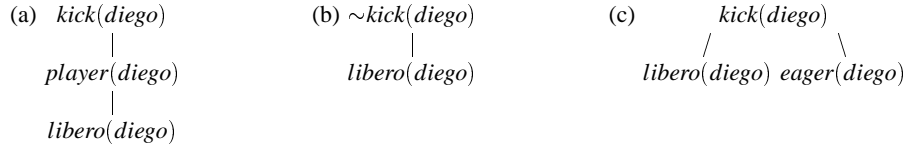
1. *The root node of  $T$  is labelled with  $h$ .*
2. *For each node  $N$  in  $T$  that is labelled with the literal  $L$ , there is a strict or defeasible rule with head  $L_0$  and body  $\{L_1, \dots, L_k\}$  in  $P$ , such that  $L = L_0\sigma$  for some ground variable substitution  $\sigma$ ,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

*Example 2.5.* Revisiting Example 2.3, the query  $\sim kick(oli)$  has the defeasible derivation  $\{\sim kick(oli) \leftarrow goalie(oli), goalie(oli)\}$ , the query  $kick(oli)$  has the following defeasible derivation:  $\{kick(oli) \prec player(oli), player(oli) \leftarrow goalie(oli), goalie(oli)\}$ .

The derivation for  $kick(diego)$  is  $\{kick(diego) \prec player(diego), player(diego) \leftarrow libero(diego), libero(diego)\}$  (Figure 1 a shows the corresponding derivation tree), whereas  $\sim kick(diego)$  has the derivation  $\{\sim kick(diego) \prec libero(diego), libero(diego)\}$  (see also Figure 1 b).

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  $P$  is *contradictory* iff it is possible to defeasibly derive from  $P$  a pair of complementary literals wrt. strong negation. We will

assume that in every DLP  $P$  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, 12].



**Fig. 1.** Derivation trees.

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

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

The literal  $h$  will also be called the conclusion supported by  $A$ . An argument  $\langle B, q \rangle$  is a *sub-argument* of  $\langle A, h \rangle$  iff  $B \subseteq A$ . 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 [21].

*Example 2.7.* Using the DLP of Example 2.3, the literal  $\sim kick(diego)$  has the argument  $A_1 = \{\sim kick(diego) \prec libero(diego)\}$  and the literal  $kick(diego)$  has two arguments, namely  $A_2 = \{kick(diego) \prec player(diego)\}$  and  $A_3 = \{(kick(diego) \prec libero(diego), eager(diego))\}$ <sup>1</sup> whose derivation

<sup>1</sup> We use parentheses for improving the readability of the set of rules here.

tree is shown in Figure 1 c. The literal  $\sim\text{kick}(\text{oli})$  has a derivation formed only by strict rules, so  $A = \emptyset$  is an argument for it. Note that  $B = \{\text{kick}(\text{oli}) \prec \text{player}(\text{oli}), \text{goalie}(\text{oli})\}$  for  $\text{kick}(\text{oli})$  (Example 2.5) is a defeasible derivation, although it is not an argument for this literal because  $\Pi \cup B$  is contradictory.

Given an argument  $A$  for a literal  $q$ , other arguments that contradict  $A$  (called rebuttals or counter-arguments) could exist. We say that  $\langle A_1, h_1 \rangle$  *counter-argues*  $\langle A_2, h_2 \rangle$  at a literal  $h$  iff there exists a sub-argument  $\langle A, h \rangle$  of  $\langle A_2, h_2 \rangle$  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  $\langle A_1, h_1 \rangle$  defeats  $\langle A_2, h_2 \rangle$  at literal  $h$  iff there exists a sub-argument  $\langle A, h \rangle$  of  $\langle A_2, h_2 \rangle$  such that  $\langle A_1, h_1 \rangle$  counter-argues  $\langle A_2, h_2 \rangle$  at  $h$ , and one of the following conditions hold:*

1.  $\langle A_1, h_1 \rangle$  is “better” (wrt. some preference criterion) than  $\langle A, h \rangle$ ; then  $\langle A_1, h_1 \rangle$  is a proper defeater of  $\langle A_2, h_2 \rangle$ ; or
2.  $\langle A_1, h_1 \rangle$  is unrelated by the given preference order to  $\langle A, h \rangle$ ; then  $\langle A_1, h_1 \rangle$  is a blocking defeater of  $\langle A_2, h_2 \rangle$ .

In DeLP, a query  $q$  will succeed if the supporting argument for it is not defeated. In order to answer the question whether  $A$  is a non-defeated argument, counter-arguments that could be *defeaters* for  $A$  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 [11] 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 [22] and extended later to be used in the defeasible argumentation formalism of [24, 25]. 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  $A_1$  will be considered more specific than  $A_2$ , because  $A_1$  does not use the strict rule  $player(X) \leftarrow libero(X)$  and hence is more direct. On the other hand, the argument  $A_3$  will be regarded as strictly more specific than  $A_1$ , because  $A_3$  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  $P = (\Pi, \Delta)$  be a program,  $\Pi_G$  be the set of all strict rules in  $\Pi$  which are not facts,  $F$  be the set of all literals that have a defeasible derivation from  $P$ , and  $\langle A_1, h_1 \rangle$  and  $\langle A_2, h_2 \rangle$  be arguments, such that  $\Pi \cup \{h_1, h_2\}$  is contradictory. Then,  $\langle A_1, h_1 \rangle$  is more specific than an argument  $\langle A_2, h_2 \rangle$  (denoted  $\langle A_1, h_1 \rangle \succeq \langle A_2, h_2 \rangle$ ) iff for all  $H \subseteq F$  it holds:  $\Pi_G \cup H \cup A_1 \vdash h_1$  and  $\Pi_G \cup H \not\vdash h_1$  implies  $\Pi_G \cup H \cup A_2 \vdash h_2$ . According to [22], we define:  $\langle A_1, h_1 \rangle$  is strictly more specific than  $\langle A_2, h_2 \rangle$  (written  $\langle A_1, h_1 \rangle \succ \langle A_2, h_2 \rangle$ ) iff  $\langle A_1, h_1 \rangle \succeq \langle A_2, h_2 \rangle$  and  $\langle A_2, h_2 \rangle \not\prec \langle A_1, h_1 \rangle$ .

To understand Definition 3.1 better, observe that the set  $\Pi_G$  does not contain facts, so the condition  $\Pi_G \cup H \cup A_1 \vdash h_1$  will hold usually only with some particular non-empty set  $H$ . We say that  $H$  activates  $A_1$ . 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  $A_1$  for deriving  $h_1$ . Hence, Definition 3.1 may be read as:  $\langle A_1, h_1 \rangle$  is more specific than  $\langle A_2, h_2 \rangle$ , if for each set  $H$  that non-trivially activates  $A_1$ , the same set  $H$  activates  $A_2$ .

Continuing with Example 2.7, argument  $A_1$  is strictly more specific than  $A_2$  (see below), because  $A_1$  does not use the strict rule  $player(X) \leftarrow libero(X)$  and hence is more direct. Observe that the condition from Definition 3.1 holds: every set  $H$  that activates  $A_1$  also activates  $A_2$ . However, the set  $H' = \{player(diego)\}$  activates  $A_2$ , but does not activate  $A_1$ , where  $A_1 = \{\sim kick(diego) \prec libero(diego)\}$  and  $A_2 = \{kick(diego) \prec player(diego)\}$ . On the other hand, the argument  $A_3 = \{(kick(diego) \prec libero(diego), eager(diego))\}$  is strictly more specific than  $A_1$ , because it uses more information. Again, the condition holds and the set  $H'' = \{libero(diego)\}$  is enough to activate  $A_1$  but does not activate  $A_3$ .

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

$$\{(s(X) \leftarrow q(X)), q(a), (p(X) \prec q(X)), (\sim p(X) \prec q(X), s(X))\}$$

The rule  $r_1: \sim p(X) \prec q(X), s(X)$  is using more information than  $r_2: p(X) \prec q(X)$ . Thus, in a system with priorities over rules such as [6, 20] (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  $\sim p(a)$  will be preferred over  $p(a)$ . But in this case, it is not true that  $\sim p(a)$  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  $A = \{\sim p(a) \prec q(a), s(a)\}$  for  $\sim p(a)$  is *not* more specific than the argument  $B = \{p(a) \prec q(a)\}$ , and vice versa.

In some systems [2, 19] 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  $A$  will be strictly more specific than  $B$ . 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 needs to test all subsets  $H \subseteq F$ . If  $F$  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  $A$  be an argument for a ground literal  $h$  wrt. a program  $P = \Pi \cup \Delta$ . We call the literal tree  $T_{\langle A, h \rangle}$  derivation tree pruned wrt. the argument  $\langle A, h \rangle$  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  $r \notin A$  and (b) all nodes which dominate only leaf nodes*

labeled with presumptions  $r' \in A$  (or instances thereof). A completion of an argument  $A$  for  $h$ , denoted  $A^c$ , is then the set of defeasible and strict rules (without facts), that are used in  $T_{\langle A, h \rangle}$ .

In order to illustrate Definition 3.2, let us revisit Example 2.3 and Figure 1 c again. The derivation tree  $T$  in Figure 1 c makes only use of the rule  $r = (\text{kick}(\text{diego}) \prec \text{libero}(\text{diego}), \text{eager}(\text{diego}))$ . Thus,  $T$  is already pruned wrt. the argument  $A_1 = \{r\}$ . If we prune  $T$  wrt.  $A_2 = \{\text{kick}(\text{diego}) \prec \text{player}(\text{diego})\}$ , which is also an argument for  $h = \text{kick}(\text{diego})$ , then all nodes below the root have to be deleted, because  $r \notin A_2$ . Hence,  $T_{\langle A_2, h \rangle}$  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 (look at Example 3.7).

Note that  $A^c$  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  $A^c$ , denoted  $\text{Lit}(A^c)$ , will be the set of all ground literals that occur in the antecedent or consequent of every rule in  $A^c$ . For example, if  $A^c = \{ (h(t) \leftarrow a(t)), (a(t) \prec b(t), c(t)), (b(t) \leftarrow d(t)) \}$  is an argument for  $h(t)$  then  $\text{Lit}(A^c) = \{ a(t), b(t), c(t), d(t), h(t) \}$ .

**Definition 3.3 (activation set).** Let  $A^c$  be a completed argument, and  $\text{Lit}(A^c)$  the corresponding set of literals. A set  $U \subseteq \text{Lit}(A^c)$  is an activation set wrt.  $A^c$ , if  $U \cup A^c \vdash h$ , and  $U$  is minimal wrt. set inclusion (i. e.,  $\nexists U' \subseteq U$  such that  $U' \cup A^c \vdash h$ ). We will call  $\text{Act-sets}(A^c)$  the set of all activation sets wrt.  $A^c$ . The set  $U$  is called a non-trivial activation set for  $A^c$  iff  $U$  is an activation set for  $A^c$  and  $U \cup \Pi_G \not\vdash h$ . We will call  $\text{NTAct-sets}(A^c)$  the set of all the non-trivial activation sets wrt.  $A^c$ .

Figure 2 shows an algorithm to compute all non-trivial activation sets wrt.  $\langle A, h \rangle$ . In order to avoid trivial activation sets we only check whether a defeasible rule has been used. Note that the first activation set for  $A$  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.

**Definition 3.4 (specificity revisited, preliminary version).** Let  $A$  for  $h_1$  and  $B$  for  $h_2$  be two arguments with contradictory conclusions, and  $A^c$

Input: a completed argument  $A^c$  for  $h$ .  
Output:  $\text{NTAct-sets}(A^c)$

1. A stack  $S$  is initialized with the pair  $(\{h\}, \text{trivial})$ .
2.  $\text{NTAct-sets}(A^c)$  is initially empty.
3. Repeat until  $S$  is empty
  - (a) Select the first pair  $(n, \text{type})$  in  $S$ .
  - (b) For each literal  $l_i \in n$  that is a consequent of a rule  $r$  in  $A^c$ , replace  $l_i$  with the literals of the antecedent of  $r$ , obtaining a new activation set  $n_i$ . The type of each  $n_i$  is *trivial* only if the type of  $n$  is *trivial* and  $r$  is a strict rule. Otherwise the type of  $n_i$  is *non-trivial*.
  - (c) The new activation sets  $n_i$  that are not previously expanded, are added to the top of  $S$ .
  - (d) If the type of  $N$  is *non-trivial* then add  $n$  to  $\text{NTAct-sets}(A^c)$ .
4. Return  $\text{NTAct-sets}(A^c)$

**Fig. 2.** Computing non-trivial activation sets.

and  $B^c$  the respectively completed arguments. We say that  $A$  is strictly more specific than  $B$  iff

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

Given an argument  $A$  for a literal  $h$  there is no unique  $A^c$  because there could be different rules in  $\Pi_G$  that can prove an antecedent of a defeasible rule in  $A$ . Nevertheless, the difference between two argument completions  $A^c$  and  $B^c$  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:

$$\begin{aligned} \text{Act-sets}(A) &= \bigcup_{i=1}^n \text{Act-sets}(A^c_i) \\ \text{NTAct-sets}(A) &= \bigcup_{i=1}^n \text{NTAct-sets}(A^c_i) \end{aligned}$$

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

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

**Theorem 3.6.** The Definitions 3.1 (for  $\succ$ ) 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  $A$ . We must take into consideration also related arguments pruned wrt.  $A$  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:

$$\{(x \leftarrow a, f), c, d, e, (x \prec a, b, c), (a \prec d), (b \prec e), (f \prec e), (\sim x \prec a, b)\}$$

Clearly,  $\langle A, x \rangle$  with  $A = \{(x \prec a, b, c), (a \prec d), (b \prec e)\}$  and  $\langle B, \sim x \rangle$  with  $B = \{(\sim x \prec a, b), (a \prec d), (b \prec e)\}$  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  $\sim x$ .

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  $\langle B, \sim x \rangle$ , because  $\{d, f\}$  would not be considered as an activation set for  $A$ . This means,  $A$  is more specific than  $B$ . But  $B$  is not more specific than  $A$ , because  $\{a, b\}$  non-trivially activates  $\sim x$ , but not  $x$ . The problem is that there are two arguments for  $x$ , namely  $A$  and  $C = \{(a \prec d), (f \prec e)\}$ , 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)  $\{\{player(diego), libero(diego)\}\}$ , (b)  $\{\{libero(diego)\}\}$  and (c)  $\{\{libero(diego)\}, \{eager(diego)\}\}$ , respectively. With this notion of paths, we are able to give a (prelimi-

nary) syntactic definition of specificity as follows, by introducing the relation  $\supseteq$ .

**Definition 3.8.** *Let  $T_1$  and  $T_2$  be two trees. We define  $T_1 \supseteq T_2$  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,  $\langle A_1, h_1 \rangle \geq \langle A_2, h_2 \rangle$  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  $T_1 \supseteq T_2$ .*

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  $F$  in Definition 3.1 are also called *possible facts*.<sup>2</sup>

**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  $\langle A_1, h_1 \rangle \succeq \langle A_2, h_2 \rangle$ . If  $\Pi_G$  is empty, then also the converse holds:  $\langle A_1, h_1 \rangle \succeq \langle A_2, h_2 \rangle$  implies  $\langle A_1, h_1 \rangle \geq \langle A_2, h_2 \rangle$ .*

*Example 3.11.* To see the necessity of the restriction in the second part of the theorem, consider the program  $\{(x \prec a, b), (b \prec c), (\sim x \prec c, d), (d \prec a), a, c, (\sim x \prec b, d)\}$ . For  $A = \{(x \prec a, b), (b \prec c)\}$  and  $B = \{(\sim x \prec c, d), (d \prec a)\}$ , it holds  $\langle A, x \rangle \succeq \langle B, \sim x \rangle$ , because  $\{a, b\}$  and  $\{a, c\}$  are the only activation sets for  $\langle A, x \rangle$ , which also activate  $\langle B, \sim x \rangle$ . However,  $\langle A, x \rangle \not\geq \langle B, \sim x \rangle$ , because there is only one derivation  $T_1$  for  $x$ , and there are two derivations  $T_2$  for  $\sim x$  (pruned wrt.  $B$ ), but for both of them it holds  $T_1 \not\supseteq T_2$ .

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

<sup>2</sup> This theorem repairs so to speak the (false) conjecture in [25, Lemma 2.24].

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 [9] 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 [9]. 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 [9], the authors claim that most priority-based approaches define the semantics of  $T$  wrt. certain partial orders on  $D$ , determined only by  $K$ . Let  $PO_K$  be the set of all partial orders defined in this way. For every partial order  $\alpha \in PO_K$  (where  $(d, d') \in \alpha$  means that  $d$  has lower priority than  $d'$ ), we define  $<_\alpha$  to be a partial order between sets of defaults in  $D$ , where  $S <_\alpha S'$  means that  $S$  is preferred to  $S'$ . Whatever the definition of  $<_\alpha$ , it has to satisfy the following property: *Let  $S$  be a subset of  $D$ , and let  $d, d'$  be two defaults in  $D$  such that  $(d, d') \in \alpha$ . Then  $S \cup \{d'\} <_\alpha S \cup \{d\}$ .*

The partial order  $<_\alpha$  can be extended into a partial order between models in  $B \cup E$ , by defining  $M <_\alpha M'$  iff  $D_M <_\alpha D_{M'}$ , 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  $PO_K$  such that  $M$  is minimal wrt.  $<_\alpha$ . [9] shows that any preferential semantics based on  $<_\alpha$  is not satisfactory enough since the set of evidence  $E$  is not considered.

*Example 4.1* (taken from [9]). 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  $M = \{\neg a, d, \neg c, b\} <_{\alpha} M' = \{a, d, \neg c, b\}$  since  $D_M = \{c/b, d/\neg a\} <_{\alpha} D'_M = \{c/b, b/a\}$  (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  $d/\neg a$  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  $b \prec a$ . 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  $\Delta = \{(a \prec b), (\sim a \prec d), (c \prec d), (b \prec c)\}$  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  $\{a \prec b\}$ ).

The previous example shows that our approach provides a proof-theoretical way of computing preferred extensions for a given default theory (expressed in terms of DeLP language), and preference among defaults (defeasible rules) is determined dynamically during the dialectical analysis.<sup>3</sup> 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

<sup>3</sup> This implies that our approach is context-sensitive as defined in [9], although this is denied in the same reference.

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 [23]— 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 [23] 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.

In [3], the concept of *argumentative framework* is characterized. Different instances of this abstract argumentation framework (AAF) can be shown to correspond to different non-monotonic formalisms, particularly normal logic programming [16]. If  $P$  is a normal logic program, then the set of hypotheses associated with  $P$  can be defined as  $H(P) = \{not\ p \mid p \text{ is a variable-free atom in the Herbrand base of } P\}$  and the background monotonic logic is the classical logic of (definite) Horn programs. These programs are obtained by replacing each negative literal  $not\ p$  in the program  $P$  by a new positive atom  $not\_p$ . Following [16], we will denote the resulting program using the same symbol  $P$ , and the consequence relation of this logic by  $\models$ . Given a program  $P$ , a conjunction of (variable-free) literals  $G$  is a *non-monotonic consequence* of  $P$  wrt. some semantics iff there exists a set of hypothesis  $\Delta \subseteq H(P)$ , allowed by the semantics, such that  $P \cup \Delta \models G$ . Different semantics for LP allow different sets of hypothesis. Most of these semantics can be formulated by a single attacking relation between sets of (variable-free) negative literals. In [5] it was shown how to recast DeLP into the AAF, using an ELP-like setting. Figure 3 summarizes the main differences between NLP and DeLP under the AAF. It should be noted that DeLP considers preference among conflicting sets of hypotheses instead of attack as in NLP. Besides, dialectical considerations are also introduced in DeLP, characterizing a stricter version of wf-semantics.

Brewka [4] 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 repre-

<b>Normal Logic Programming</b> (under AAF setting)	<b>Defeasible Logic Programming</b> (under ELP, AAF-like setting)
An argument is a set of hypotheses $\Delta$ satisfying derivability	An argument is a set of hypothesis $H(A)$ satisfying consistency, minimality, derivability
Attacks defined on hypotheses ( $p$ attacks <i>not</i> $p$ )	Attack defined on complementary literals $p$ and $\sim p$ , and hypotheses $p$ and <i>not</i> $p$
Attacks involve no preference	Attacks involves preference
Well-founded semantics	Well-founded acceptability restricted by dialectical considerations and preference criterion
Dialectical considerations: - no circularity - consistency among all defense nodes	Dialectical considerations: - no circularity - consistency within defense and attack nodes within argumentation lines
Existence of well-founded pre-tree iff well-founded acceptability	Existence of a dialectical tree which entails pre-tree structures Root marked $U$ iff justification

**Fig. 3.** Normal logic programming and DeLP under AAF settings.

sentation 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. Due to the lack of space, we will not go into more details here.

## 4.2 LP and Defeasible Logic with Superiority Relation

In [15] and later in [7], *logic programming without negation as failure* (LPwNF) was introduced. A LPwNF program consists of a set of basic rules  $L_0 \leftarrow L_1, \dots, L_n$  (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  $r_2 : \sim p \leftarrow r$ , 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:  $\sim p$ . 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 [7] 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 [13], 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, 19], another approach for defeasible reasoning is presented. In this context, defeasible logic programs are (almost) identical to programs  $P$  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  $P$ , we have to consider the pair  $(P, >)$  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 [22, 25]. 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 [10]). 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 [9]). Last but not least, the integration of defeasible reasoning into agent programming should be tackled in greater detail.

## References

1. J. J. Alferes and L. M. Pereira, editors. *Reasoning with Logic Programming*, LNAI 1111, Berlin, 1996. Springer.
2. G. Antoniou, M. J. Maher, and D. Billington. Defeasible logic versus logic programming without negation as failure. *Journal of Logic Programming*, 42:47–57, 2000.
3. A. Bondarenko, P. M. Dung, R. A. Kowalski, and F. Toni. An abstract, argumentation-theoretic approach to default reasoning. *Artificial Intelligence*, 93:63–101, 1997.
4. G. Brewka and T. Eiter. Prioritizing default logic. In *Festschrift for the 60th Annivesary of W. Bibel*. Kluwer, 1998.
5. C. I. Chesñevar and G. R. Simari. Modeling defeasibility into an extended logic programming setting using an abstract argumentation framework. In *Proceedings of the Argentinean Symposium on Artificial Intelligence*, pages 71–89. JAIIO, Buenos Aires, Argentina, Sept. 1999.
6. M. A. Covington, D. Nute, and A. Vellino. *Prolog Programming in Depth*. Scott, Foresman and Company, Glenview, IL, London, 1988.
7. Y. Dimopoulos and A. Kakas. Logic programming without negation as failure. In *Proceedings of 5th. International Symposium on Logic Programming*, pages 369–384, Cambridge, MA, 1995. MIT Press.
8. J. Dix, F. Stolzenburg, G. R. Simari, and P. R. Fillottrani. Automating defeasible reasoning with logic programming (DeReLoP). In S. Jähnichen, editor, *Proceedings of the 2nd German-Argentinian Workshop on Information Technology*, pages 39–46, Königswinter, 1999.
9. P. M. Dung and T. C. Son. An argumentation-theoretic approach to reasoning with specificity. In *Proc. of the Knowledge Representation and Reasoning*, pages 507–519, 1996.
10. A. J. García. Defeasible logic programming: Definition and implementation. Master’s thesis, Dep. de Ciencias de la Computación, Universidad Nacional del Sur, Bahía Blanca, Argentina, July 1997.
11. A. J. García, G. R. Simari, and C. I. Chesñevar. An argumentative framework for reasoning with inconsistent and incomplete information. In *Workshop on Practical Reasoning and Rationality*. 13th biennial European Conference on Artificial Intelligence (ECAI-98), Aug. 1998.
12. M. Gelfond and V. Lifschitz. Logic programs with classical negation. In D. Warren and P. Szeredi, editors, *Proc. ICLP*, pages 579–597. MIT Press, 1990.

13. M. Gelfond and T. C. Son. Reasoning with prioritized defaults. In *Lecture Notes in Artificial Intelligence 1471, Selected Papers from the Workshop on Logic Programming and Knowledge Representation*, pages 164–223, 1997.
14. J. F. Horty. Some direct theories of nonmonotonic inheritance. In D. M. Gabbay, C. J. Hogger, and J. A. Robinson, editors, *Handbook of Logic in Artificial Intelligence and Logic Programming, Vol 3, Nonmonotonic Reasoning and Uncertain Reasoning*, pages 111–187. Oxford University Press, 1994.
15. A. C. Kakas, P. Mancarella, and P. M. Dung. The acceptability semantics for logic programs. In *Proceedings of the 11th. International Conference on Logic Programming*, pages 504–519, Santa Margherita, Italy, 1994. MIT Press.
16. A. C. Kakas and F. Toni. Computing argumentation in logic programming. *Journal of Logic and Computation*, 9(4):515–562, 1999.
17. V. Lifschitz. Foundations of logic programs. In G. Brewka, editor, *Principles of Knowledge Representation*. CSLI Publications, 1996.
18. J. W. Lloyd. *Foundations of Logic Programming*. Springer, Berlin, Heidelberg, New York, 1987.
19. M. J. Maher, G. Antoniou, and D. Billington. A study of provability in defeasible logic. In J. Slaney and G. Antoniou, editors, *Proceedings of the 11th Australian Joint Conference on Artificial Intelligence*, LNAI 1502, pages 215–226. Springer, Berlin, Heidelberg, New York, 1998.
20. D. Nute. Defeasible logic. In D. M. Gabbay, C. J. Hogger, and J. A. Robinson, editors, *Handbook of Logic in Artificial Intelligence and Logic Programming, Vol 3, Nonmonotonic Reasoning and Uncertain Reasoning*, pages 355–395. Oxford University Press, 1994.
21. J. L. Pollock. Self-defeating arguments. *Minds and Machines (Special issue: defeasible reasoning)*, 1(4), Nov. 1991.
22. D. L. Poole. On the comparison of theories: Preferring the most specific explanation. In *Proceedings of the 9th International Joint Conference on Artificial Intelligence*, pages 144–147. IJCAI Inc., San Mateo, CA, Morgan Kaufmann, Los Altos, CA, 1985.
23. H. Prakken and G. Sartor. Argument-based logic programming with defeasible priorities. *Journal of Applied Non-classical Logics*, 7:25–75, 1997.
24. G. R. Simari, C. I. Chesñevar, and A. J. García. The role of dialectics in defeasible argumentation. In *Anales de la XIV Conferencia Internacional de la Sociedad Chilena para Ciencias de la Computación*. Universidad de Concepción, Concepción (Chile), Nov. 1994.
25. G. R. Simari and R. P. Loui. A mathematical treatment of defeasible reasoning and its implementation. *Artificial Intelligence*, 53:125–157, 1992.
26. F. Stolzenburg, O. Obst, J. Murray, and B. Bremer. Spatial agents implemented in a logical expressible language. In M. Veloso, E. Pagello, and H. Kitano, editors, *RoboCup-99: Robot Soccer WorldCup III*, LNAI. Springer, 2000. To appear.
27. G. A. Vreeswijk. Abstract argumentation systems. *Artificial Intelligence*, 90:225–279, 1997.
28. X. Wang, J. You, and L. Yuan. Logic programming without default negation revisited. In *Proceedings of IEEE International Conference on Intelligent Processing Systems*, pages 1169–1174. IEEE, 1997.

## Appendix: Proof of Theorem 3.10

Let us first prove the first part of the statement. Thus, by hypothesis it holds that  $\Pi_G \cup H \cup A_1 \vdash h_1$  for some set of possible facts  $H$  with  $\Pi_G \cup H \not\vdash h_1$ . This means, there is a derivation  $T_1$  with  $\Pi_G \cup H \cup A_1 \vdash_{T_1} h_1$ . Since  $\Pi_G$  does not contain any facts, the leaves in  $T_1$  must be labeled with literals from  $H$  or presumptions in  $A_1$ . Since by hypothesis the activation set  $H$  for  $\langle A_1, h_1 \rangle$  is non-trivial, there must exist a derivation  $T'_1$  for  $h_1$  that is identical with  $T_1$ , but completed with additional sub-derivations below those leaves in  $T_1$  which are neither facts in  $P$  nor presumptions in  $A_1$ , and afterwards pruned wrt.  $A_1$ .

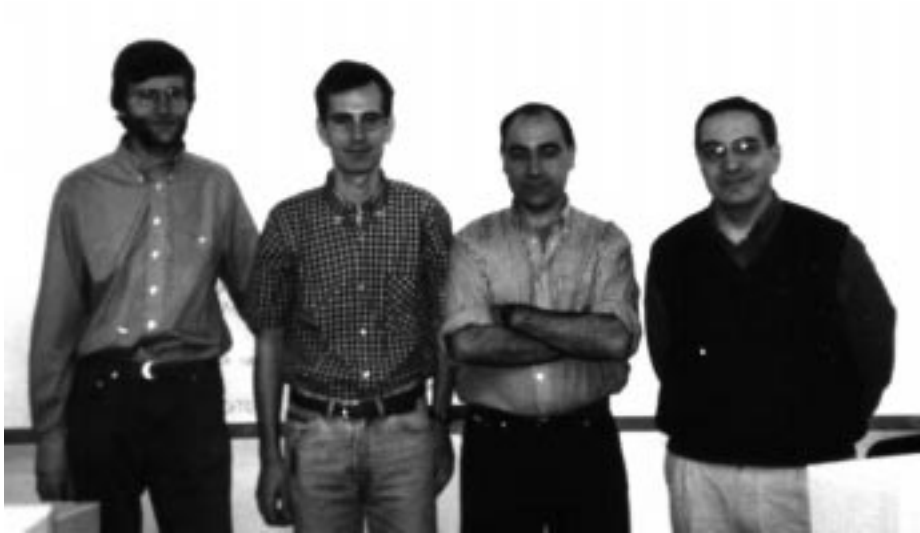
Now, by precondition, there is a derivation  $T'_2$  for  $h_2$  pruned wrt.  $A_2$  such that  $T'_1 \supseteq T'_2$ . For each path  $t \in T_2'^P$ , there must be a literal  $L \in H$ , since otherwise there would be a path  $t^*$  in  $T_2'^P$ , such that no element of  $H$  occurs in  $t^*$ , but this contradicts  $T'_1 \supseteq T'_2$ . Now we delete all sub-derivations below nodes labeled with a literal  $L \in H$ . Obviously, the obtained tree  $T_2$  is a derivation tree, satisfying  $\Pi_G \cup H \cup A_2 \vdash_{T_2} h_2$ . Hence, it holds  $\Pi_G \cup H \cup A_2 \vdash h_2$ . This completes the first part of the proof.

In order to show the second part, we first notice that since an argument  $A$  must be minimal (according to Definition 2.6, condition 3), for each literal  $L$ , there can be at most one defeasible rule with head  $L$  in  $A$ . Furthermore, since  $\Pi_G$  is empty by precondition in this case, it follows, that every argument  $\langle A, h \rangle$  corresponds to exactly one derivation tree  $T$ . This observation will be helpful in the following argument.

Let  $T_1$  be the derivation for  $\langle A_1, h_1 \rangle$  (possibly pruned wrt.  $A_1$ ), which exists by hypothesis. The leafs of this tree clearly are labeled with program facts. Let  $H$  be the set of them. Obviously,  $A_1 \cup H \vdash h_1$  (and  $\Pi_G \cup H \not\vdash h_1$ ). Therefore, we conclude  $A_2 \cup H \vdash h_2$  because of  $\langle A_1, h_1 \rangle \succeq \langle A_2, h_2 \rangle$  (by precondition). Let  $T_2$  be the corresponding derivation (possibly pruned wrt.  $A_2$ ), which is unique as just stated.

Let us assume  $T_1 \not\supseteq T_2$ . Then there must be a path  $t_2 \in T_2^P$  such that for all paths  $t_1 \in T_1^P$  it holds  $t_1 \not\supseteq t_2$ . This means, in each path  $t_1$ , there must be (at least) one literal  $L$  which is not in  $t_2$ . Let  $H'$  the set of all these literals. Clearly,  $H'$  is a non-trivial activation set for  $\langle A_1, h_1 \rangle$ . Thus by precondition, there must be a derivation  $T'_2$  with  $A_2 \cup H' \vdash_{T'_2} h_2$ . Now,  $T'_2$  can be completed with additional sub-derivations and pruned wrt.  $A_2$  such that all leaves are facts. However, the obtained tree is different from  $T_2$ , because it cannot contain the path  $t_2$ . But this contradicts to the fact

that derivation trees are uniquely determined. Hence, it holds  $T_1 \supseteq T_2$ , and finally  $\langle A_1, h_1 \rangle \geq \langle A_2, h_2 \rangle$ . This completes the second part of the proof.  $\square$



**Fig. 4.** The authors. Bahía Blanca, Argentina, October 1999.

## Available Research Reports (since 1998):

### 2000

- 4/2000** *Frieder Stolzenburg, Alejandro J. García, Carlos I. Chesñevar, Guillermo R. Simari.* Introducing Generalized Specificity in Logic Programming.
- 3/2000** *Ingar Uhe, Manfred Rosendahl.* Specification of Symbols and Implementation of Their Constraints in JKogge.
- 2/2000** *Peter Baumgartner, Fabio Massacci.* The Taming of the (X)OR.
- 1/2000** *Richard C. Holt, Andreas Winter, Andy Schürr.* GXL: Towards a Standard Exchange Format.

### 1999

- 10/99** *Jürgen Ebert, Luuk Groenewegen, Roger Süttenbach.* A Formalization of SOCCA.
- 9/99** *Hassan Diab, Ulrich Furbach, Hassan Tabbara.* On the Use of Fuzzy Techniques in Cache Memory Management.
- 8/99** *Jens Woch, Friedbert Widmann.* Implementation of a Schema-TAG-Parser.
- 7/99** *Jürgen Ebert, and Bernt Kullbach, Franz Lehner (Hrsg.).* Workshop Software-Reengineering (Bad Honnef, 27./28. Mai 1999).
- 6/99** *Peter Baumgartner, Michael Kühn.* Abductive Coreference by Model Construction.
- 5/99** *Jürgen Ebert, Bernt Kullbach, Andreas Winter.* GraX – An Interchange Format for Reengineering Tools.
- 4/99** *Frieder Stolzenburg, Oliver Obst, Jan Murray, Björn Bremer.* Spatial Agents Implemented in a Logical Expressible Language.
- 3/99** *Kurt Lautenbach, Carlo Simon.* Erweiterte Zeitstempelnetze zur Modellierung hybrider Systeme.
- 2/99** *Frieder Stolzenburg.* Loop-Detection in Hyper-Tableaux by Powerful Model Generation.
- 1/99** *Peter Baumgartner, J.D. Horton, Bruce Spencer.* Merge Path Improvements for Minimal Model Hyper Tableaux.

### 1998

- 24/98** *Jürgen Ebert, Roger Süttenbach, Ingar Uhe.* Meta-CASE Worldwide.

- 23/98** *Peter Baumgartner, Norbert Eisinger, Ulrich Furbach.* A Confluent Connection Calculus.
- 22/98** *Bernt Kullbach, Andreas Winter.* Querying as an Enabling Technology in Software Reengineering.
- 21/98** *Jürgen Dix, V.S. Subrahmanian, George Pick.* Meta-Agent Programs.
- 20/98** *Jürgen Dix, Ulrich Furbach, Ilkka Niemelä .* Nonmonotonic Reasoning: Towards Efficient Calculi and Implementations.
- 19/98** *Jürgen Dix, Steffen Hölldobler.* Inference Mechanisms in Knowledge-Based Systems: Theory and Applications (Proceedings of WS at KI '98).
- 18/98** *Jose Arrazola, Jürgen Dix, Mauricio Osorio, Claudia Zepeda.* Well-behaved semantics for Logic Programming.
- 17/98** *Stefan Brass, Jürgen Dix, Teodor C. Przymusiński.* Super Logic Programs.
- 16/98** *Jürgen Dix.* The Logic Programming Paradigm.
- 15/98** *Stefan Brass, Jürgen Dix, Burkhard Freitag, Ulrich Zukowski.* Transformation-Based Bottom-Up Computation of the Well-Founded Model.
- 14/98** *Manfred Kamp.* GReQL – Eine Anfragesprache für das GUPRO-Repository – Sprachbeschreibung (Version 1.2).
- 12/98** *Peter Dahm, Jürgen Ebert, Angelika Franzke, Manfred Kamp, Andreas Winter.* TGraphen und EER-Schemata – formale Grundlagen.
- 11/98** *Peter Dahm, Friedbert Widmann.* Das Graphenlabor.
- 10/98** *Jörg Jooss, Thomas Marx.* Workflow Modeling according to WfMC.
- 9/98** *Dieter Zöbel.* Schedulability criteria for age constraint processes in hard real-time systems.
- 8/98** *Wenjin Lu, Ulrich Furbach.* Disjunctive logic program = Horn Program + Control program.
- 7/98** *Andreas Schmid.* Solution for the counting to infinity problem of distance vector routing.
- 6/98** *Ulrich Furbach, Michael Kühn, Frieder Stolzenburg.* Model-Guided Proof Debugging.
- 5/98** *Peter Baumgartner, Dorothea Schäfer.* Model Elimination with Simplification and its Application to Software Verification.

**4/98** *Bernt Kullbach, Andreas Winter, Peter Dahm, Jürgen Ebert.* Program Comprehension in Multi-Language Systems.

**3/98** *Jürgen Dix, Jorge Lobo.* Logic Programming and Nonmonotonic Reasoning.

**2/98** *Hans-Michael Hanisch, Kurt Lautenbach, Carlo*

*Simon, Jan Thieme.* Zeitstempelnetze in technischen Anwendungen.

**1/98** *Manfred Kamp.* Managing a Multi-File, Multi-Language Software Repository for Program Comprehension Tools — A Generic Approach.