Preference Revision Via Declarative Debugging

Pierangelo Dell'Acqua[1,2] and Luís Moniz Pereira[2]

[1] Department of Science and Technology - ITN,
Linköping University, 601 74 Norrköping, Sweden
pier@itn.liu.se
[2] Centro de Inteligência Artificial - CENTRIA,
Departamento de Informática, Faculdade de Ciências e Tecnologia,
Universidade Nova de Lisboa, 2829-516 Caparica, Portugal
lmp@di.fct.unl.pt

Abstract. Preference criteria are rarely static. Often they are subject to modification and aggregation. The resulting preference criteria may not satisfy the properties of the original ones and must therefore be revised. This paper investigates the problem of revising such preference criteria by means of declarative debugging techniques.

1 Motivation

Preference criteria are subject to be modified when new information is brought to the knowledge of the individual, or aggregated when we need to represent and reason about the simultaneous preferences of several individuals. As motivating example, suppose you invite three friends Karin, Leif and Osvald to go and see a movie. Karin prefers thrillers to action movies. Leif, on the other hand, prefers action movies to thrillers. Finally, Osvald is like Leif and prefers action movies to thrillers. Suppose you need to buy the tickets. Which movie do you choose?

Preference aggregation is an important problem and potential applications of this work include those where preference reasoning plays a role, e.g., in artificial intelligence, political science, and economics (cf. social choice and multi-criteria decision).

Typically, preference criteria must satisfy certain properties, e.g., those of strict partial order. When aggregating or updating preference criteria such properties might not be preserved, and therefore the need arises for a revision. In this paper, we consider any preference criteria expressible in the language of logic programs (LP), and investigate the problem of revising them by means of declarative debugging techniques for LP. In particular, we employ an adapted version of the contradiction removal method defined for the class of normal logic programs plus integrity constraints proposed in [10]. The resulting framework is flexible and general, and tailored neither to any specific preference criteria nor any specific method for preference aggregation, but rather to any method expressible in LP. The ability to express meta-information on the diagnoses of a revision problem gives us a further level of abstraction permitting to select the best diagnosis for the problem at hand.

C. Bento, A. Cardoso, and G. Dias (Eds.): EPIA 2005, LNAI 3808, pp. 18–28, 2005.

2 Background

In this section we provide some logic programming fundamentals and few basic definitions regarding preference relations.

2.1 Language

Let $\mathcal{L}$ be a first order language. A literal in $\mathcal{L}$ is an atom A in $\mathcal{L}$ or its default negation *not* A. A normal logic program P over $\mathcal{L}$ (sometimes simply called program) is a set of rules and integrity constraints of the form:

$$A \leftarrow L_1, \ldots, L_n \quad (n \geq 0)$$

where A is an atom, $L_1, \ldots, L_n$ are literals in $\mathcal{L}$, where in integrity constraints A is $\bot$ (contradiction). A rule stands for all its ground instances with respect to $\mathcal{L}$. When $n = 0$ we write the rule as A. $\mathcal{L}_P$ denotes the language of P.

For normal logic programs we consider the Well Founded Semantics [7]. We write $P \models L$ whenever a literal L belongs to the well-founded model of a program P. P is contradictory if $P \models \bot$. Programs are liable to be contradictory because of the integrity constraints.

Example 1. Let $P = \{a \leftarrow not\ b; \bot \leftarrow a\}$. Since we have no rules for b, by Closed World Assumption (CWA), it is natural to accept *not* b as true and therefore conclude a. Because of the integrity constraint, we conclude $\bot$ and thus engender a contradiction. ∎

2.2 Preference Relation

Given a set N, a preference relation $\succ$ is any binary relation on N. Given two elements a and b in N, $a \succ b$ means that a is preferred to b. We assume that N contains at least two elements.

We do not assume any property of $\succ$, although in many situations it will satisfy the properties of a strict partial order. Typical properties of $\succ$ include:

- irreflexivity: $\forall x.\ x \not\succ x$
- asymmetry: $\forall x \forall y.\ x \succ y \Rightarrow y \not\succ x$
- transitivity: $\forall x \forall y \forall z.\ (x \succ y \wedge y \succ z) \Rightarrow x \succ z$
- negative transitivity: $\forall x \forall y \forall z.\ (x \not\succ y \wedge y \not\succ z) \Rightarrow x \not\succ z$
- connectivity: $\forall x \forall y.\ x \succ y \vee y \succ x \vee x = y$

The relation $\succ$ is:

- a strict partial order if it is irreflexive and transitive (thus also asymmetric);
- a weak order if it is a negatively transitive strict partial order;
- a total order if it is a connected strict partial order.

Every preference relation $\succ$ induces an indifference relation $\sim$. Two elements a and b in N are indifferent $a \sim b$ if neither is preferred to the other one, that is, $a \not\succ b$ and $b \not\succ a$.