Default Reasoning Implementation in CoGui

Patrice Buche¹, Jérôme Fortin², and Alain Gutierrez³

¹ INRA IATE, LIRMM GraphIK, France
² Université Montpellier II
³ CNRS LIRMM, France

Abstract. This is an application paper in which we propose to present the actual implementation of default reasoning under conceptual graph formalism using CoGui. CoGui is a free graph-based visual tool, developed in Java, for building Conceptual Graph knowledge bases. We present the extension of this application to define and represent default CG rules (a CG-oriented subset of Reiter’s default logics) and how to use these rules in skeptical or credulous reasoning.

1 Introduction

Default conceptual graph rules have been introduced in [1] in order to model expert knowledge, especially in agronomy applications. Default CG rules encode a subset of Reiter’s default logic [2] and deal with knowledge of the form "if an hypothesis is proved true, a conclusion is generally true unless something that we know prevent us to infer this conclusion. Dealing with default is a kind of non monotonic reasoning because adding some new information to a knowledge base may prevent to apply some default.

The contribution of this paper is the presentation of the actual implementation of default reasoning under conceptual graph formalism using CoGui. CoGui is a free graph-based visual tool, developed in Java, for building Conceptual Graph knowledge bases.

The paper is organized as follows : Section 2 introduces classical notions of CG and CG rule. It gives some clues about their implementation in CoGui. Section 3 recalls basic definition of Reiter’s defaults and introduces conceptual default rules. Section 4 is devoted to the presentation of the deduction algorithm using default CG rules.

2 Conceptuals Graphs in Cogui

In this section, we recall main notations and results required for the default CG rules used in this paper. In Section 2.2 we present the simple CGs of [3]. In Section 2.3 the CG rules of [4].

For each of these subsections we present how CoGui is designed to model these different notions. Figure 1 shows the structure of Cogui packages. CoGui is composed of four different layers associated with four different eclipse projects :
– the project `fr.lirmm.graphik.cogui.core` contains the model of knowledge representation, and all algorithms that permit to explore this model. It also contains tools to serialize data objects in xml, and to transpose a model in Datalog+ [5,6];
– the project `fr.lirmm.graphik.cogui.rdf` permits to make import and export from the CoGui internal model to RDF(S) and an OWL fragment [7];
– the project `fr.lirmm.graphik.cogui.edit` contains the user interface;
– the project `fr.lirmm.graphik.cogui.appli` is the upper layer of CoGui containing its entry point in the `CoGuiApplication` Class.

![Fig. 1. Packages Structure of CoGui](image)

2.1 Support

Syntax. With the simple CGs of [3], a knowledge base is structured into two objects: the vocabulary (also called support) encodes hierarchies of types, and the conceptual graphs (CGs) themselves represent entities and relations between them. Simple CGs are extended to handle conjunctive types, as done in [8].

Definition 1 (Vocabulary). We call vocabulary a tuple \( \mathcal{V} = (\mathcal{C}, \mathcal{R} = (\mathcal{R}_1, \ldots, \mathcal{R}_k), \mathcal{M}_I, \mathcal{M}_G) \) where \( \mathcal{C} \) is a partially ordered set of concept types that contains a greatest element \( \top \), each \( \mathcal{R}_i \) is a partially ordered set of relation types of arity \( i \), \( \mathcal{M}_I \) is a set of individual markers, and \( \mathcal{M}_G \) is a set of generic markers. Note that all these sets are pairwise disjoint, and that we denote all the partial orders by \( \leq \).

Definition 2 (Conjunctive types). A conjunctive concept type over a vocabulary \( \mathcal{V} \) is a set \( T = \{t_1, \ldots, t_p\} \) (that we can note \( T = t_1 \cap \ldots \cap t_p \) of arity \( p \). If \( T = \{t_1, \ldots, t_p\} \) and \( T' = \{t'_1, \ldots, t'_q\} \) are two conjunctive concept types, then we also note \( T \leq T' \iff \forall t'_i \in T', \exists t_j \in T \text{ such that } t_j \leq t'_i. \)