Interactive Model Generation

5.1 Introduction

In this chapter, we describe the design and implementation of our Common Interactive Domain-level Reasoning Engine (CIDRE). It is used as a tool to “breathe life” into our logic-based domain modelling discussed in Chapter 4 in order to take advantage of these models for concrete instances of domain tasks.

We start by outlining the related component architecture of its usage environment. CIDRE is conceived as one module in a spoken language dialogue system. In particular, the reasoning engine is to be used as an interactive component that mediates between the dialogue manager (acting on behalf of the user) and the domain applications that provide domain-specific functionalities and services. Building on that we discuss some of the advantages and potential drawbacks of using finite model generation as a reasoning procedure. Next, the actual implementation of the reasoning engine and the algorithms used are described. Several issues concerning our implementation, such as the data structures used, are discussed. A distinctive feature of our engine is the interactive protocol it provides to enable the use of the engine as an interactive component, instead of an autonomous black box. We discuss the design and implementation of that protocol. Finally, in Section 5.5 we describe our approach to testing the reasoning engine as part of the development cycle and present related results. These provide more insights into the workings of the engine when applied in a concrete domain-specific setting.

For our purposes, finite model generation [78] is a deductive reasoning procedure that, given a theory, or set of first-order clauses, as input on return provides consistent finite models, or, alternatively, a proof of the inconsistency of the theory. We distinguish between axioms on the one hand which are first-order logic formulas containing variables and, on the other hand, ground units, called facts. A model returned by the model generation procedure can be viewed as an extension of the input theory with a set of facts such that each axiom of the theory is met.
CIDRE implements a tableau-based deduction algorithm enhanced with novel features for an interactive usage. In contrast to some existing (especially non-tableau-based) implementations of finite model generation \cite{77} which focus on enumerating small models fast, our procedure attempts to cope with larger models in an interactive fashion. It is designed to work as a module that may be synchronised with a dialogue manager’s information state and contributes information, and thus is not required to find complete solutions autonomously. One of the most important features for this kind of interactive functionality is the reasoner’s capability to store a trace for each proof step that is taken. This enables, firstly, to trace inferences back to their original hypotheses, which is necessary for explanation and conflict resolution in dialogue. In addition, it enables the reasoner to keep track of inconsistent sets of hypotheses, so-called nogoos. Nogoos are reused in the search process and may potentially cut off a large subtrees of the search space.

In terms of architecture of the usage environment (cf. Figure 1.1), the crucial difference to a conventional dialogue systems is that the dialogue manager does not directly communicate with different domain applications. Instead, the reasoning engine can be seen as a mediator between the dialogue manager and the different domain applications. All components in the architecture operate asynchronously, such that the dialogue manager may be occupied with producing output or with interpreting the next user utterance, while the reasoner is still processing input based on the previous dialogue state. In addition, the processing performed by the reasoning engine is incremental in the sense that it is able to incorporate new information from the dialogue manager without abandoning all of its previous work. Both the dialogue manager and the domain applications access the proof database that is created by the reasoning engine to store the proofs it generates. The components can query the proof database in order to obtain new inferences from the reasoning engine. The new information can also be filtered as needed by the querying component. The components can create new tasks to be processed by the engine. Within a reasoning task, the domain applications can create proof structures in order to implement domain-specific reasoning functionality. For instance, a travel planning application may create a proof concerning the duration of a trip as soon as the origin and destination are given in a form understood by the application. The domain applications have to make sure that the proofs they create are sound, i.e. all preconditions that are used in the proof (which are available as separate proof structures) have to be referenced correctly. This will be described in more detail in Section 5.4.

5.2 Model Generation as a Reasoning Procedure

Advantages. It is worthwhile to discuss some of the advantages and potential disadvantages of using finite model generation as an inference procedure: First of all, model generation reasoning is based on the notion of satisfiability