This chapter describes a visual tool for debugging and analysis of the search-trees generated by finite domain constraint programs. The tool allows to navigate in the search-tree in a flexible way and gives, for any node of the search-tree, a clear view of the current state of the program execution. The tool provides graphical representations of the form of the search-tree, of constraints and variables of the program and of the propagation steps performed after each decision in the tree. The debugger is used via a set of meta-predicates which annotate the search routine given by the user, which allows great flexibility in adapting the program to the needs of different users. The tool is now part of the CHIP constraint programming environment and covers important aspects both of correctness and performance debugging.

7.1 Introduction

In recent years, a significant number of applications have been developed using constraint programming (CP) technology [7.13] [7.15] [7.16]. The complexity of problems handled is increasing and improvement of the debugging facilities becomes an urgent task. Currently, the CP technology is largely lacking debugging tools and a debugging methodology to support users. This methodology is a key point because CP programs are, different from conventional programs, data-driven computation rather than program-driven. Typical finite domain programs are structured into three parts: variable definition, constraint statement and finally the search procedure. Debugging concerns all three parts, but special emphasis lies on the search procedure, as most real-life optimisation problems encountered in industry have a very large search space. To understand the effect of the search procedure on the search space there is a requirement for a novel visual tool which allows to perform an abstraction of the constraints, and which shows different views of the variables, constraints and the search space. At the same time, its use should be simple and intuitive and should not require major changes in the program under analysis. According to requirements collected from different users of CHIP and a study inside the DiSCiPl project [7.8], debugging tools should be usable at different levels of expertise, from a novice constraint programmer trying to understand how constraints work, to the expert programmer developing and debugging large applications, but also by the tool developer.
to understand existing and to help find new or improved propagation mechanisms. It is important not only to cover the aspect of correctness debugging, finding errors in the logical meaning of the program, but also to help performance debugging, improving the speed of a correct, but slow application. The concept of global constraints introduced in CHIP [7.2] [7.4] has drastically reduced the number of constraints needed to express a problem, and allows the programmer to focus more on heuristics for the search procedure. At the same time, new debugging requirements for these powerful abstractions have arisen. This chapter discusses the search-tree visualisation tool for CHIP [7.14] developed at COSYTEC in the DiSCIPl project. The chapter is structured as follows: We first review existing work on debugging tools for finite domain programs in section 7.1. We then give in section 7.2 a motivation why search-tree visualisation is an important aspect in the development of large scale constraint applications and present the overall working principle of the search-tree tool. This is followed in section 7.3 by a description of the programmer’s interface used. In section 7.4 we present the different views the tool offers. In the last section 7.5, we describe the current state of development and further features, which are currently under development. Visualisation tools for the global constraints in CHIP are described in chapter 12, while chapter 13 gives examples and an analysis of the use of the visualisation tools presented here.

7.2 Related Work

Given the recognised difficulty of developing correct and efficient constraint programs, there is a surprising lack of work on debugging aspects in constraint programming. In most systems, debugging tools are based on a trace of the execution. This makes it very hard to extract general information about the search procedure, and also leads to much time consuming navigation through the trace in order to find the current point of interest. The paper of Meier [7.11] describes GRACE, a tool to visualise domains in the context of a normal box-model trace. It can also be used to follow (in a textual form) individual propagation steps in the trace. The tool can be extensively reconfigured by the user to execute user-written code at each step of the trace process. It takes advantage of the fact that the propagation engine is written in Prolog, so that modifications can be performed in the kernel. The paper of Schulte [7.12], describing the Oz Explorer, is the main influence on our work. The Oz Explorer provides a graphical interface to display and control the search. It is possible to collapse or expand parts of the search-tree in order to concentrate on interesting sub-parts. A major advantage over our system is the possibility to program new search methods with a small set of primitives of the concurrent language. On the other hand, it does not contain at the moment views on constraints or on propagation steps so that its capability to