Making Exhaustive Search Programs Deterministic*

Kazunori UEDA
ICOT Research Center,
Institute for New Generation Computer Technology,
4-28, Mita 1-chome, Minato-ku, Tokyo 108, Japan.

Received 12 December 1986
Revised manuscript received 23 February 1987

Abstract This paper presents a technique for compiling a Horn-clause program intended for exhaustive search into a GHC (Guarded Horn Clauses) program. The technique can be viewed also as a transformation technique for Prolog programs which compiles away the 'bagof' primitive and non-determinate bindings. The class of programs to which our technique is applicable is shown with a static checking algorithm; it is nontrivial and could be extended. An experiment on a compiler-based Prolog system showed that our technique improved the efficiency of exhaustive search by 6 times for a permutation generator program. This compilation technique is important also in that it exploits the AND-parallelism of GHC for parallel search.

Keywords: Exhaustive Search, Multiple Binding Environments, Compilation, Program Transformation, Continuation, Mode Analysis, Parallelism, Guarded Horn Clauses

§1 Introduction

We often use Horn-clause logic, or more specifically the language Prolog, to obtain all solutions of some problem, that is, to obtain all answer substitutions for a goal to be solved. In this framework, however, it is difficult to collect the obtained solutions into a single environment for further processing such as counting the number of the solutions, comparing them, classifying them, and so on. This is because these solutions correspond to different, independent paths of a search tree. For this reason, many of Prolog implementations support system predicates for creating a list of all the solutions of a goal given as an argument;

* This paper is a revised version of the paper presented at the Third International Conference on Logic Programming, 1986, London.
examples are 'setof' and 'bagof' of DEC-10 Prolog. Naish made a survey of all-solutions predicates in various Prolog systems. These system predicates, however, internally use some extralogical features to record the obtained solutions. So it should be an interesting question whether exhaustive search can be done without such primitives.

Another motivation is that we may sometimes wish to do exhaustive search in GHC or other parallel logic programming languages which do not directly support exhaustive search. In this case, parallelism inherent in GHC should be effectively used for the search.

One possible way to achieve the above requirements is to directly write down a first-order relation which states, for example, that "\( S \) is a list of all the solutions of the N-queens problem." It is almost evident that such a relation can be described within the framework of Horn-clause logic. However, in practice, it is much harder to write it manually than to write a program that finds only one solution at a time. A programming tool which automatically generates an exhaustive search program may resolve this situation, and this is the way we will pursue in this paper.

§2 Outlines of the Method

Our method is to compile a Horn-clause program intended for exhaustive search using backtracking or OR-parallelism into a GHC program or a deterministic Prolog program which returns the same (multi-)set of solutions in the form of a single list. The word 'deterministic' means that all bindings given to variables are determinate and never undone. Prolog programs in this subclass are interesting from the viewpoint of implementation, since they never call for a trail stack. Furthermore, determinism in this sense has a similarity with the semantical restriction which GHC imposed to a proof procedure for Horn-clause logic in order to make all activities done in a single environment. This similarity is reflected by the fact that a transformed program can be interpreted both as a GHC program and as a Prolog program by the slight change between the '!' (commitment) operator and the '!' (cut) operator.

There are two possible views of this transformation technique. One is to regard this as compilation from a Horn-clause program to a guarded-Horn-clause program. By compiling OR-parallelism into AND-parallelism, we eliminate a multiple environment mechanism for managing different binding environments created simultaneously by the paths of a search tree. The other view is to regard it as transformation of a Prolog program. This transformation serves as simplification in the sense that all-solutions predicates and the unbinding mechanism can be eliminated. Moreover, this transformation may remarkably improve the efficiency of a search program, as we will see in Section 7.

Our technique has another important meaning. By making search performed in a single environment, it becomes possible to introduce a mechanism for controlling the search. That is, our technique may provide a starting