Issues of Non-Determinism in PROLOG and the Committed Choice Non-Deterministic Logic Languages

R. Trehan* & P. F. Wilk†
*Department of Artificial Intelligence, and †Artificial Intelligence Applications Institute, University of Edinburgh, Edinburgh, UK

Abstract. Issues of logic programming have always been bound with those of search. Logic programs are generally thought of as a logical specification of the problem (the declarative reading) in terms of clauses. The inference mechanism of the given logic language provides a means by which the clauses can be used to prove a query to the system (procedural interpretation). The problem of search can be related into forms of non-determinism. That is how to select a possible solution path (clause). It is this criteria that is used to consider the issues of search in PROLOG and the Committed Choice Non-Deterministic execution models.

1 Introduction

Logic programming languages offer many modelling capabilities. In particular logic programming languages attempt to model the non-determinism found in logical conjectures, i.e. making the correct logical deduction when faced with several choices. This search capability of logic programming languages has proven to be powerful for defining and implementing algorithms, concisely and efficiently. Programs that exhibit different forms of non-determinism require different forms of search to select the correct solution path at a choice point.

In this paper we compare how search is achieved in a sequential logic programming language, PROLOG (Sterling & Shapiro, 1986), and in the parallel logic programming languages, PARLOG (Gregory, 1985); GUARDED HORN CLAUSES (GHC) (Ueda, 1986); and CONCURRENT PROLOG (CP) (Shapiro, 1983), which belong to the Committed Choice Non-Deterministic (CCND) class of languages. The comparison is based on how the different forms of non-determinism are realized in the different computational models.

The paper is organized as follows. We first give a brief overview of the CCND computation model and the three main CCND languages, PARLOG, GHC and CP, a
knowledge of PROLOG is assumed. We then consider how well the two computation models support search. This comparison is based on how the two paradigms support various forms of non-deterministic algorithms, namely: don't care non-determinism; don't know non-determinism; and generate and test non-determinism. We then consider various approaches, automatic and manual, that address some of the limitations of mapping general search algorithms to the CCND computation model. Finally we present our conclusions and comments.

2 Committed choice non-deterministic languages

2.1 Syntax of GUARDED HORN CLAUSES

A Committed Choice Non-Deterministic (CCND) program is a finite set of guarded horn clauses of the form:

\[ R(a_1, \ldots, a_k) :- G_1, \ldots, G_n : B_1, \ldots, B_m \quad (n,m \geq 0) \]

The different CCND languages adopt various names for the various components of the guarded horn clause. We use the following terminology for all the languages:

- \( R(a_1, \ldots, a_k) \) is a head goal;
- \( R \) is its functor, or predicate name;
- \( k \) is the number of arguments (referred to as the predicate arity);
- \( G_1, \ldots, G_n \) form the guarded goals;
- \( ' : ' \) is known as the commit operator;
- \( B_1, \ldots, B_m \) are known as the body goals.

where the \( G_s \) and \( B_s \) are literals.

The commit operator generalizes and cleans the cut of sequential PROLOG; the cut is used to control and reduce the search of OR-branches in PROLOG. The commit operator forms the means of pruning OR-branches in a parallel search.

A general query in the CCND languages has the following form:

\[ :- C_1, C_2, \ldots, C_n \]

2.2 Semantics of GUARDED HORN CLAUSES

2.2.1 Declarative semantics. A guarded horn clause program has a similar declarative reading to Horn clause based programs (Sterling & Codish, 1987).

Each clause:

\[ H :- G_1, \ldots, G_n : B_1, \ldots, B_m \]

is read as:

\( H \) is true if \( G_1 \) and \( \ldots \) and \( G_n \) and \( B_1 \) and \( B_m \) are all true.