High-Level Nondeterministic Abstractions in C++

Laurent Michel\(^1\), Andrew See\(^1\), and Pascal Van Hentenryck\(^2\)

\(^1\) University of Connecticut, Storrs, CT 06269-2155
\(^2\) Brown University, Box 1910, Providence, RI 02912

Abstract. This paper presents high-level abstractions for nondeterministic search in C++ which provide the counterpart to advanced features found in recent constraint languages. The abstractions have several benefits: they explicitly highlight the nondeterministic nature of the code, provide a natural iterative style, simplify debugging, and are efficiently implementable using macros and continuations. Their efficiency is demonstrated by comparing their performance with the C++ library GECODE, both for programming search procedures and search engines.

1 Introduction

The ability to specify search procedures has been a fundamental asset of constraint programming languages since their inception (e.g., \(^1\)\(^3\)\(^13\)) and a differentiator compared to earlier tools such as Alice \(^7\) and MIP systems where search was hard-coded in the solver. Indeed, by programming the search, users may define problem-specific branching procedures and heuristics, exploit unconventional search strategies, break symmetries dynamically, and specify termination criteria for the problem at hand. The last two decades have also witnessed significant progress in this area (e.g., \(^6\)\(^8\)\(^9\)\(^12\)\(^14\)\(^15\)): Modern constraint programming languages enable programmers to specify both the search tree and the search strategy, provide high-level nondeterministic abstractions with dynamic filtering and ordering, and support hybrid and heuristic search.

The embedding of constraint programming in mainstream languages such as C++ has also been a fundamental step in its acceptance, especially in industry. With constraint programming libraries, practitioners may use familiar languages and environments, which also simplifies the integration of a constraint programming solution within a larger application. ILOG Solver \(^10\) is the pioneering system in this respect: it showed how the nondeterministic abstractions of constraint logic programming (e.g., goals, disjunction, and conjunction) can be naturally mapped into C++ objects. To specify a search procedure, users thus define C++ objects called goals, and combine them with logical connectives such as or and and. In recent years, constraint programming libraries have been enhanced to accommodate search strategies \(^9\)\(^14\) (originally proposed in Oz \(^12\)) and high-level nondeterministic abstractions \(^8\) (originally from OPL \(^14\)).

However these libraries, while widely successful, still have two inconveniences as far as specifying search procedures. On the one hand, they impose a recursive style for search procedures, which contrasts with the more familiar iterative constructs of OPL as indicated in \(^2\). Second, these libraries may obscure
the natural nondeterministic structure of the program and may produce some non-trivial interleaving of C++ code and library functions. This complicates the debugging process which alternates between library and user code.

This paper is an attempt to mirror, in constraint programming libraries, the high-level nondeterministic abstractions of modern constraint programming languages. The paper shows that it is indeed possible and practical to design a search component in C++ that

- promotes an iterative programming style that expresses both sequential composition and nondeterminism naturally;
- simplifies the debugging process, since the C++ stack now reflects the full control flow of the application;
- is as efficient as existing libraries.

The technical idea underlying the paper is to map the nondeterministic abstractions of Comet \cite{15} into C++ using macros and continuations. Obviously, since continuations are not primitive in C++, it is necessary to show how they can be implemented directly in the language itself. The implementation differs significantly from the OPL implementation in which the abstractions are implemented using Ilog Solver facilities.

The rest of the paper is organized as follows. Section 2 presents the nondeterministic abstractions and their benefits. Section 3 shows how to implement continuations in C++. Section 4 shows how to use macros and continuations to implement the nondeterministic abstractions. Section 5 presents the experimental results which shows that the nondeterministic abstractions can be implemented efficiently and compare well with the search implementation of Gecode.

2 The Search Abstractions

This section describes the search abstractions in C++. Section 2.1 starts by describing the nondeterministic abstractions used to define the search tree to explore. These abstractions are parameterized by a search controller that specifies how to explore the search tree. Search controllers are briefly discussed in Section 2.2 and are presented in depth in \cite{15}.

2.1 Nondeterministic Abstractions

The nondeterministic abstractions are mostly modelled after OPL \cite{14}.

Static Choices. The try construct creates a binary search node representing the choice between two alternatives. The snippet

0. TRY(sc)
1. cout << "yes" << endl;
2. OR(sc)
3. cout << "no" << endl;
4. ENDTry(sc)