Search Combinators for MiniZinc

Tom Schrijvers[1], Guido Tack[2], Horst Samulowitz[3], Pieter Wuille[2], and Peter J. Stuckey[4]

[1] Universiteit Gent, Belgium
[2] Katholieke Universiteit Leuven, Belgium
[3] IBM Research, USA
[4] National ICT Australia (NICTA) and University of Melbourne, Victoria, Australia

In previous work [2], Samulowitz et al. have proposed a lightweight search language for MiniZinc. Based on our recent development of *search combinators* [3], we extend and improve upon this standardization proposal by providing a more generic and extensible modeling language that is build up on a practical implementation approach.

For the user, we provide a compositional approach for expressing complex search heuristics based on an (extensible) set of primitive combinators. These combinators combine and modify MiniZinc's base searches (with all the variable and value selection strategies proposed by [3]) into more complex search heuristics.

A small catalog of primitive combinators can be found in [3]. While we emphasize that this catalog is open-ended, the provided combinators are already expressive enough to express a range of well-known search heurstics, including iterative deepening, limited discrepancy search, branch-and-bound, restart optimization, credit-based pruning heuristics, etc. In fact, our approach is also capable of specifying even more refined search strategies such as Dichotomic Search [4].

The following heuristic briefly demonstrates the conciseness of our search combinators. This search heuristic can be used to solve radiotherapy treatment planning problems [1]. The heuristic minimizes a variable obj using branch-and-bound (bab), first searching the variables N, and then verifying the solution by partitioning the problem along the row_i variables, one row at a time. Failure on one row must be caused by the search on the variables in N, and consequently search never backtracks into other rows (exh_once). This is a good example of integrating domain knowledge in search:

$$\mathsf{bab}(k, \mathsf{and}([\mathsf{base\_search}(N, \dots)]{+}{+} \\ [\mathsf{exh\_once}(\mathsf{base\_search}(row_i, \dots)) \mid i \text{ in } 1..n]))$$

Here we assume that a basic search construct like the following is available:

$$s = \mathsf{int\_search}(vars, \textit{var-select}, \textit{value-select})$$

which specifies a systematic search over the variables *vars*, applying *var-select* and *value-select* as variable- and value-selection strategies respectively. Another primitive combinator is and, with the obvious meaning. The remainders, bab and exh_once, are themselves defined in terms of other primitive combinators.

For the system developer, we show how to design and implement modular combinators. Developers do not have to cater explicitly for all possible combinator combinations. Small implementation efforts result in providing the user with a lot of expressive power. Moreover, the cost of adding one more combinator is small, yet the return in terms of additional expressiveness can be quite large.

The tough technical challenge we face here does not lie in designing a high-level syntax; several proposals have already been made (e.g., [2]). Our contribution is to bridge the gap between a conceptually simple search language (high-level, functional and naturally compositional) and an efficient implementation (typically low-level, imperative and highly non-modular). This is where existing approaches fail; they restrict the expressiveness of their search specification language to face up to implementation limitations, or they raise errors when the user strays out of the implemented subset.

We overcome this challenge by implementing the primitives of our search language as *mixin* components. As in Aspect-Oriented Programming, mixin components neatly encapsulate the *cross-cutting behavior* of primitive search concepts, which are highly entangled in conventional approaches. Cross-cutting means that a mixin component can interfere with the behavior of its sub-components (in this case, sub-searches). The combination of encapsulation *and* cross-cutting behavior is essential for systematic reuse of search combinators. Without this degree of modularity, minor modifications require rewriting from scratch.

An added advantage of mixin components is extensibility. We can add new features to the language by adding more mixin components. The cost of adding such a new component is small, because it does not require changes to the existing ones. Moreover, experimental evaluation bears out that this modular approach has no significant overhead compared to the traditional monolithic approach.

Implementation We have developed two diametrically opposed approaches for the Gecode C++ library: *dynamic composition* (interpretation) and *static composition* (compilation). Experimental evaluation shows that both implementation approaches have competitive performance and match the performance of the native implementation of the same search heuristics in Gecode.

Challenges

- The standard set of primitive combinators has to be decided upon. MiniZinc implementations should provide a means to add custom user-defined combinators.
- In order to facilitate the reuse of search heuristics defined in terms of multiple combinators, MiniZinc requires an abstraction mechanism, for instance *macros*. This should allow to name a heuristic, define it in a *library* together with other predefined heuristics, and import it into models as required.
- A library of well-known and frequently used heuristics should be part of MiniZinc's standard search language. After all, we do not expect the user to define such basic heuristics as branch-and-bound over and over again.

References

1. Baatar, D., Boland, N., Brand, S., Stuckey, P.J.: CP and IP approaches to cancer radiotherapy delivery optimization. Constraints 16(2), 173–194 (2011)
2. Samulowitz, H., Tack, G., Fischer, J., Wallace, M., Stuckey, P.: Towards a lightweight standard search language. In: ModRef (2010)
3. Schrijvers, T., Tack, G., Wuille, P., Samulowitz, H., Stuckey, P.: Search combinators. In: Proceedings of the 17th International Conference on Principles and Practice of Constraint Programming (2011)

4. Sellmann, M., Kadioglu, S.: Dichotomic search protocols for constrained optimization. In: CP. LNCS, vol. 5202, pp. 251–265. Springer (2008)