Implementing Global Constraints as Structured Networks of Elementary Constraints

Dávid Hanák

Constraints serve as a basis for Constraint Logic Programming (CLP), a group of logic programming (LP) languages which are usually embedded into a host language such as C, Java or Prolog. A branch of CLP is CLP(FD), a constraint language which operates on variables of integer values. FD here stands for finite domain, because in the constraint store each variable is represented by the finite set of values which it can take. These variables are connected by the constraints, which propagate the change of the domain of one variable to the domains of others. A constraint can be thought of like a sleeping "daemon" which wakes up when the domain of at least one of its variables is changed, propagates the change and falls asleep again. This change can be induced by another constraint or by the labeling process, which enumerates the solutions by gradually substituting all possible values into the variables. There are two major implementation techniques for CLP(FD) constraints: indexicals and global constraints. The former always operate on a fixed number of variables while the latter are more generic and can work with a variable number of variables. These constraints are usually handled very differently for efficiency reasons.

[1] introduces a new aspect of defining and describing global finite domain constraints based on graphs. It gives a description language which enables mathematicians, computer scientists and programmers to share information on global constraints in a way that all of them understands. It also helps to categorize global constraints, and as a most important feature, it makes possible to write programs which, given only this abstract description, can automatically generate parsers, type checkers and pruners (propagators) for specific global constraints.

To define a constraint, an initial graph is generated. The arguments (variables) of the global constraints are assigned to its nodes, while a single elementary constraint is assigned to each of its arcs (elementary constraints are very simple and easy to handle, like A=B). The final graph consists of those arcs of the initial graph for which the elementary constraints hold. The global constraint itself succeeds if a given set of graph properties (i.e. restrictions on the number of arcs, sources, connected components, etc.) holds for this final graph. The description language contains terms to express type and value restrictions on the arguments of the constraint, to determine the graph generator that creates the initial graph, and to specify the elementary constraint and the graph properties that must hold for the final graph.

To put this theory into practice, an interpreter is being written which understands a language very similar to the one defined in [1]. The interpreter is being implemented in SICStus Prolog and its main purpose is to serve as a prototype, therefore the description language was modified slightly in order to suit the syntax of Prolog, thus removing the burden of implementing a parser as well. The interpreter in its current state of development can read the description of a constraint, and given a set of specific values it can check if the values meet the type and value restrictions posed by the constraint and whether the constraint holds for them. Therefore it might be regarded as a complex relation checker. Although it doesn't do any pruning yet, some minor mistakes and inconvenient notations already cropped up by using it.

A design have been created about the extension of the interpreter with pruning capabilities. Here the line of thought is reversed: we assume that the constraint holds, and from the required graph properties we try to deduce conclusions on the domains of its variables. When a constraint wakes up, some of the elementary constraints assigned to the arcs of the graph are sure to hold, some are sure to fail while the state of the rest is yet uncertain. By knowing what graph properties should be achieved and what are the domains of the variables currently, some of these uncertain constraints can be forced into success or into failure. The global constraint finally becomes entailed when there aren't any uncertain arcs left.

The propagator using the descried algorithm will be implemented and fitted into the CLP(FD) library of SICStus Prolog by utilizing the well defined interface of user defined global constraints. This way it will be possible to thoroughly test both the program and the theory itself in a trusted environment. According to my plans, a working prototype will already be available at the conference and I will be able to present it along with new observations and experiences. Finally, if this case study proves the theory to be worthy of further practical investigation, an emphasis could be put on efficiency matters also when implementing new interpreters and perhaps pruner algorithm generators.

References

[1] Beldiceanu, Nicolas: "Global Constraints as Graph Properties on Structured Network of Elementary Constraints of the Same Type", SICS Technical Report T2000/01, ISSN 1100-3154, January 28, 2000