FDBG, a CLP(FD) Debugger for SICStus Prolog

Dávid Hanák and Tamás Szeredi

CLP stands for Constraint Logic Programming. This acronym signifies a group of logic programming (LP) languages which are usually embedded into a host language such as C, Java or Prolog. In these languages the programmer is able to establish correlations between (usually numeric) variables and find the values where these constraints hold. The CLP language family has several branches depending on the variable domains. Thus we can speak of CLP(B), where the variables can have boolean values, CLP(R) and CLP(Q) where the values are real or rational numbers respectively, CLP(FD) where the values are integers, and CHR, a more generic way of handling constraints, where the programmer defines the domain. They have in common a monotonously growing constraint store to keep track of constraints.

In CLP(FD), FD stands for finite domain, because in the constraint store each variable is represented by the finite set of integer 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 labelling process, which enumerates the solutions by enumerating all possible values of 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.

SICStus Prolog includes an implementation of several CLP languages. Prolog as a host language is a very good choice, because the finding of the solutions requires backtracking, which is a fundamental notion in Prolog too. CLP implementations for other (non-logic) host languages on the other hand must explicitly include a backtracker. SICStus also includes a generic and extendible debugger for regular Prolog, but so far a tracing tool for CLP(FD) was missing. And since CLP programs don't run in linear order but behave rather like a set of coroutines, it requires a great effort to trace them with the Prolog debugger.

The main purpose of writing FDBG (which stands for Finite domain DeBuGger) was to enable CLP(FD) programmers to trace the changes of finite domain constraint variables. Our goal was trace the wake-up of constraints and see their effects on variables, as well as labelling events. Because CLP programs run differently than regular Prolog programs do we chose not to implement a traditional step-by-step debugger but to use the wallpaper trace technique instead. This means that every piece of information is printed on the console, to a file, or something similar, and any potential bug may be found by studying this log after the run is complete. Due to the modular and flexible design of FDBG, a graphical front-end may easily be added later, in fact, we already have plans in that respect.

The trace output is a sequence of log entries. Each entry corresponds to a CLP(FD) event, a notion introduced by FDBG. One group of events represent the wake-up and activity of constraints. Such events describe which constraint is active currently and how does it narrow the domains of variables. The other group informs about the proceedings of labelling, containing data on the structure of the search tree, showing its active and failed branches. The appearance of the log entries can be varied freely by the programmer who is given a set of tools to process the events. Filters may also be applied easily to reduce the size of the log. Due to technical reasons only global constraints are handled by FDBG, indexicals are ignored altogether. However, by exploiting a special feature of SICStus CLP(FD), this is already enough to catch every event of a program which doesn't use any self-written indexicals.

It is important to mention that FDBG was written almost entirely in Prolog as user space code, no native routines were used directly. FDBG reached a level of completeness by now where it could be included (with full source) in the official SICStus distribution from version 3.9, which came out in February 2002.