Programming with $\mathcal{TOY}(\mathcal{FD})$

Antonio J. Fernández[1], Teresa Hortalá-González[2], and Fernando Sáenz-Pérez[2]

[1] Dpto. Lenguajes y Ciencias de la Computación, Univ. of Málaga, Spain
[2] Dpto. Sist. Inf. y Prog. Univ. Complutense of Madrid, Spain
afdez@lcc.uma.es, {teresa, fernan}@sip.ucm.es

In [1] we presented the language $\mathcal{TOY}(\mathcal{FD})$ that integrates the best features of existing functional and logic languages, as well as finite domain ($\mathcal{FD}$) constraint solving. We believe that $\mathcal{TOY}(\mathcal{FD})$ is more flexible and expressive than the existing approaches of constraint logic programming on finite domain ($CLP(\mathcal{FD})$) as it integrates $\mathcal{FD}$ constraint solving, lazy evaluation, higher order applications of functions and constraints, polymorphism, type checking, composition of functions (and, in particular, constraints), combination of relational and functional notation, and a number of other characteristics. These features allow to write more concise programs, therefore increasing the expressivity level.

From an implementation point of view, $\mathcal{TOY}(\mathcal{FD})$ integrates the higher-order lazy functional logic language $\mathcal{TOY}$ and the efficient $\mathcal{FD}$ constraint solver of SICStus Prolog. From a programming point of view, $\mathcal{TOY}(\mathcal{FD})$ is the first constraint functional logic programming system that provides a wide set of $\mathcal{FD}$ constraints comparable to existing $CLP(\mathcal{FD})$ systems and which is competitive with them. $\mathcal{TOY}(\mathcal{FD})$ supports *relational constraints* including equality, disequality, *arithmetical operators on constraints*, a wide set of well-known *global constraints* (e.g., `all_different`/1), *membership constraints* (e.g., `domain`/3), *propositional constraints*, and *enumeration constraints* (e.g., `labeling`/2, with a number of strategies) with optimization.

$\mathcal{TOY}(\mathcal{FD})$ also provides a glass box approach via a set of predefined functions called *reflection constraints* that allow, at runtime, to recover internal information about the constraint solving process. These functions increase the flexibility of the language as they allow the user to construct specific constraint mechanisms such as new search strategies.

Generally speaking, $\mathcal{TOY}(\mathcal{FD})$ is, from its nature, different to all existing $CLP(\mathcal{FD})$ languages as its operational mechanism is the result of combining the operational methods of logic languages (i.e., unification and resolution) and functional languages (i.e., rewriting). Thus, $\mathcal{TOY}(\mathcal{FD})$ is an alternative to $CLP(\mathcal{FD})$ languages and allows a flexible modelling and quick prototyping at a very high level that cannot be reached by most of the existing constraint systems. $\mathcal{TOY}(\mathcal{FD})$ is freely available in `http://toy.sourceforge.net/`.

References

1. Fernández, A.J., Hortalá-González, M.T., Sáenz-Pérez, F.: Solving combinatorial problems with a constraint functional logic language. In Wadler, P., Dahl, V., eds.: PADL'2003. Number 2562 in LNCS, New Orleans, Springer (2003) 320–338