Declarative Debugging of Missing Answers in Constraint Functional-Logic Programming

Rafael Caballero, Mario Rodríguez Artalejo, and
Rafael del Vado Vírseda [*]

Dep. Sistemas Informáticos y Computación, Univ. Complutense de Madrid
{rafa,mario,rdelvado}@sip.ucm.es

It is well known that constraint logic and functional-logic programming languages have many advantages, and there is a growing trend to develop and incorporate effective tools to this class of declarative languages. In particular, *debugging tools* are a practical need for diagnosing the causes of erroneous computations. Recently [1], we have presented a prototype tool for the declarative diagnosis of wrong computed answers in $CFLP(\mathcal{D})$, a new generic scheme for lazy Constraint Functional-Logic Programming which can be instantiated by any constraint domain $\mathcal{D}$ given as parameter [2]. The declarative diagnosis of *missing answers* is another well-known debugging problem in constraint logic programming [4]. This poster summarizes an approach to this problem in $CFLP(\mathcal{D})$. From a programmer's viewpoint, a tool for diagnosing missing answers can be used to experiment wether the program rules for certain functions are sufficient or not for computing certain expected answers. For example, consider a $CFLP(\mathcal{H})$-program fragment written in $\mathcal{TOY}$ [3], where strict equality and disequality constraints are used for generating family relationships based on the basic family facts shown in Fig. 1.

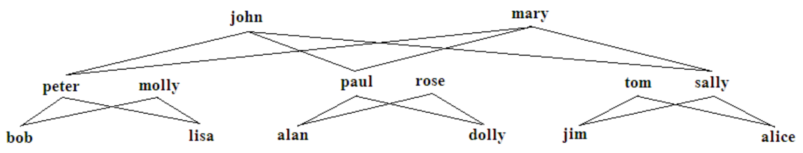

Fig. 1. Missing family relationships

```
type person = [char]
motherOf, fatherOf, sonOf, daughterOf, brotherOf, sisterOf :: person -> person
motherOf  X = Y  <== maleChildOf Z Y == X // femaleChildOf Z Y == X
brotherOf X = Y  <== maleChildOf (fatherOf X) (motherOf X) == Y, Y /= X
sonOf     X = maleChildOf X Y // maleChildOf Y X
% Analogously for the other basic family relationships
basicFamilyRelation :: person -> person
basicFamilyRelation = motherOf // fatherOf // sonOf // ... // brotherOf // sisterOf
familyRelation :: person -> person
familyRelation = basicFamilyRelation // basicFamilyRelation . basicFamilyRelation
(//) :: A -> A -> A          (.) :: (B -> C) -> (A -> B) -> (A -> C)
X // Y = X                   (F . G) X = F (G X)
X // Y = Y
```

[*] The authors have been partially supported by the Spanish National Projects MERIT-FORMS (TIN2005-09207-C03-03) and PROMESAS-CAM (S-0505/TIC/0407).

In order to find a family relationship between *alice* and *alan*, a user can inspect the computed answers for the goal `familyRelation == R, R "alice" == "alan"`, involving higher-order patterns. Different diagnosis for missing answers are possible. For instance, the answer `R -> sonOf . brotherOf . motherOf` (i.e., *alan* is son of a brother of the mother of *alice*) may be missed due to an incomplete definition of `familyRelations`, which could be extended by adding the new rule `familyRelation = familyRelation . basicFamilyRelation`; while other answers such as `R -> cousinOf` or `R -> sonOf . uncleOf` may be missed due to an incomplete definition of `basicFamilyRelation`, which could be extended like this: `basicFamilyRelation = ... // cousinOf // uncleOf`.

Declarative programming paradigms such as the $CFLP(\mathcal{D})$ scheme involve complex operational details such as constraint solving, lazy evaluation of possibly higher-order and non-deterministic functions, logical variables etc. Therefore, *declarative debugging* is a better option than debugging techniques which rely on the inspection of low-level computation traces. Declarative debugging requires suitable trees to represent computations. Inspired by [1] and [4], we propose so-called *Negative Proof Trees* (shortly, NPTs) as computation trees for declarative debugging of missing answers in $CFLP(\mathcal{D})$. NPTs represent logical proofs in a *Constrained Negative Proof Calculus*. The root of a NPT has attached an *answer collection statement* of the form $G \Rightarrow \bigvee_{i\in I} S_i$, where G is an initial goal and S_i are all the observed computed answers. Internal nodes have attached answer collection statements $f\bar{t}_n \rightarrow t \,\Box\, S \Rightarrow \bigvee_{i\in I} S_i$; these correspond to all the computed answers S_i for an intermediate goal $f\bar{t}_n \rightarrow t \,\Box\, S$, asking for results of the function call $f\bar{t}_n$ which match t and satisfy the constraints within S. NPTs are built in such a way that the validity of the collection statement at each node follows from the collection statements at their children, under the assumption that the function definition relating the parent node to the children nodes is complete. As usual, declarative diagnosis proceeds by finding a *buggy node* which is invalid but such that all its children are valid. Every such buggy node points to an incomplete function definition. The search for a buggy node can be implemented with the help of an external *oracle* (usually the user with some semiautomatic support) who has a reliable declarative knowledge of the valid collection statements, the so-called *intended interpretation*. A prototype under development is available at `http://gpd.sip.ucm.es/rafa/missing`.

References

1. R. Caballero, M. Rodríguez-Artalejo and R. del Vado-Vírseda. *Declarative Diagnosis of Wrong Answers in Constraint Functional-Logic Programming*. In Proc. of ICLP'06, volume 4079 of Springer LNCS, pp. 421–422, 2006.
2. F.J. López-Fraguas, M. Rodríguez-Artalejo and R. del Vado-Vírseda. *A New Generic Scheme for Functional Logic Programming with Constraints*. Journal of *Higher-Order and Symbolic Computation*, Volume 20, Issue 1/2, pp. 73–122, 2007.
3. F.J. López-Fraguas, J. Sánchez-Hernández. *$\mathcal{TOY}$: A Multiparadigm Declarative System*. In Proc. of RTA'99, volume 1631 of Springer LNCS, pp 244–247, 1999. System and documentation available at `http://toy.sourceforge.net`.
4. A. Tessier and G. Ferrand. *Declarative Diagnosis in the CLP Scheme*. Volume 1870 of Springer LNCS, Chapter 5, pp. 151–174, 2000.