How to teach difference lists?

Ulrich Geske

Fraunhofer FIRST
D-12489 Berlin, Kekuléstr. 7
email: Ulrich.Geske@first.fraunhofer.de

Abstract

Lists are, on the one hand, a useful modeling construct in logic programming because of their not predefined length, on the other hand the concatenation of lists by append(L1,L2,L3) is rather inefficient because it copies the list L1. To avoid the invocation of the append/3-procedure an alternative possibility is the use of incomplete lists of the form [el_1, ... el_k | Var], in which Var describes a remainder of the list not specified completely. If there is an assignment of a concrete list L for this variable in the program, it will results an efficient (physical) concatenation of the first list elements with L without copying the elements el_1,...el_k. This physical concatenation does not consist in an (extra-logically) replacing of a pointer (a memory address) but is a purely logical operation since the reference to the list L was already created by the specification in the program.

From the mathematical point of view, the difference of the two lists [el_1, ...el_k | Var] and Var denotes the initial piece [el_1, ..., el_k] of the complete list. E.g., the difference [1,2,3] arises from the lists [1,2,3| X] and X or from [1,2,3,4,5] and [4.5] or [1,2,3,a] from [1,2,3,a] and [a]. The first-mentioned possibility is the most general representation of the list difference [1,2,3]. Every list may be represented as a difference list. The empty list can be expressed as a difference of the two lists L and L, the list List is the difference of the list List and the empty list.

The combination of the two list components [el_1, ...el_k | Var] and Var in a structure with the semantics of the list difference will be denoted as a "difference list". Since such a combination is based on the possibility of specifying incomplete lists, the Prolog standard does not provide any standard notation for this. A specification of a difference list from the two lists L and R may be given, by a list notation [L, R] or by the use of a separator, e.g. L−R or L\R (the used separator must be defined in the concrete Prolog system) or as two separate arguments separated by commas in the argument list.

Unfortunately, in programs the concatenation is very frequently expressed using a call of the append/3 procedure. The reason for it may be the inadequate explanation of the use of incomplete lists and the difference list technique using such incomplete lists. Very different types of the representations of difference lists and very different attitudes to them can befound in well–known manuals and textbooks about Prolog.

On the one hand, Clocksin has described this technique in a tutorial [4] based on his book "Clause and Effect" [3] as *"probably one of the most ingenious programming techniques ever invented yet neglected by mainstream computer science"*. On the other hand, Dodds [10] opinion concerning a pair of variables for the result of a list operation and the used accumulator, like (All, Temp), is: *"some people find it helpful to think of All and Temp as a special data structure called difference list. We prefer to rest our account of the matter on the logical reading of the variable as referring to any possible tail, rather than to speak of lists with variable tails "*, which can be read in his textbook "Prolog. A Logical Approach".

In this paper, a detailed summary of the presentations to difference lists and related topics (accumulator technique) in the literature is given. One frequent problem with the list concatenation by append/3 consists in invocations of append/3 after the recursive procedure call. We will demonstrate, how such left–recursive procedures can be transformed into right–recursion using the folding–/unfolding technique. In the sequel, we will consider different ways for realization of append-free concatenations of programs. The different techniques which are essential for the list

generation can be reduced to two methods which are essential for list processing: top-down- and bottom-up generation of structures. For the present, we can show that the difference-list notation is only syntactic sugar since it can be derived from the previous techniques by a simple syntactic transformation. But difference lists are not only syntactic sugar after all, since a modeling technique using difference lists instead of additional arguments or accumulators can offer advantages. A knowledge of the background can make it easier to use difference lists and promote their use in programming.

References

1. OKeefe, Richard A.: The Craft of Prolog. The MIT Press. 1990.
2. Clark, K.L.; McCabe, F.G.: micro-Prolog: Programming in Logic. Prentice Hall International, 1984.
3. Clocksin, W.F.: Clause and Effect. Prolog Programming for the Working Programmer. Springer-Verlag. 1997.
4. Clocksin, W.F.: Prolog Programming. Lecture “Prolog for Artificial Intelligence 2001-02” at University of Cambridge, Computer Laboratory. 2002.
5. Clocksin, W.F.; Mellish, C.S.: Programming in Prolog. Springer-Verlag, 1981, 1984, 1987.
6. Colhoe, Helder; Cotta, José C.: Prolog by Example. Springer-Verlag, 1988.
7. Marriott, K.; Søndergaard, H.: Prolog Transformation by Introduction of Difference-Lists. TR 88/14. Dept. CS, The University of Melbourne, 1988.
8. Sterling, L; Shapiro, E.: The Art of Prolog. The MIT Press, 1986. Seventh printing, 1991.
9. Maier, D.; Warren, D.S.: Computing with Logic. The Benjamin/Cummings Publisher Company, Inc., 1988.
10. Dodd, Tony: Prolog. A Logical Approach.Oxford University Press, 1990