Hoare, C. A. R.; **He, Jifeng**

Unifying theories for logic programming. (English) Zbl 1005.68036

Hoare, Tony (ed.) et al., Engineering theories of software construction. Proceedings of the NATO ASI, Marktoberdorf, Germany, July 25 - August 6, 2000. Amsterdam: IOS Press. NATO Sci. Ser. III, Comput. Syst. Sci. 180, 21-45 (2001).

Summary: A theory of programming is intended to support the practice of programming by relating each program to the specification of what it is intended to achieve. An unifying theory is one that is applicable to a general paradigm of computing, supporting the classification of many programming languages as correct instances of the paradigm. A distinctive characteristic of the logic programming paradigm is that each execution of a program produces not just a single answer but a collection of answers, allowing the later parts of the program to select between them. A specification of such a program is therefore just a predicate describing the size, content, and other properties of the set of options offered. The actual behaviour of a particular program is also described by a predicate, namely the strongest specification that it is certain to satisfy. The program is correct if this behavioural predicate logically implies its specification. The semantics of the programming language is given by a shallow embedding into the predicate calculus. Atomic commands are predicates, and the operators of a programming language are defined as transformations on these predicates. The familiar **and** and **or** of logic programming are specified to combine their offer sets by intersection or union, so as to permit parallel implementation. Explicit mention of the complete set of answers simplifies definition of so-called impure features, such as the negation and the cut of Prolog.

A common objective of many logic and constraint programs is to enumerate the solutions of simultaneous equations in some mathematical space. Particular languages differ in the choice of space, in their solution method, and in their representation of the set of solutions offered. For example, Prolog solves equations in a Herbrand universe (term algebra) by means of unification, and presents the solutions as a sequence in an order determined by SLD scheduling. Such a particular language is related to the general paradigm by the standard method of data abstraction and representation.

To prove total correctness of a program, a specification language must be able to describe all the ways that a component of a program can go wrong. In logic programming, these include finite and infinite failure. Prolog also allows infinite success – producing an infinite sequence of answers. To achieve these effects, we need asymmetric versions of union and intersection, which are necessary more complicated. Nevertheless, the operators often enjoy surprisingly elegant algebraic properties, which aid both programmer comprehension and mechanical optimization. Some of the properties are proved directly from the definitions, and are applicable to specifications and designs as well as programs. Other properties have to be postulated as healthiness conditions, which are satisfied by all the primitives and preserved by all the operators of the programming language. The discovery and classification of general algebraic laws, shared by many programming paradigms and languages, is a primary goal of unification of their theories.

For the entire collection see [Zbl 0972.00060].

MSC:

68N17 Logic programming

Keywords:

SLD scheduling; specification language