Inference of Inheritance Relationships from Prolog Programs: a System Developed with PrologIII

Christine Solnon - Michel Rueher

I3S, University of Nice Sophia Antipolis - CNRS
06560 Valbonne, FRANCE
e-mail: solnon@mimosa.unice.fr

In order to support the object oriented design of reusable software components, we propose to extract an inheritance hierarchy from a Prolog prototype. The goal is to define a reverse engineering technique for recovering structural design information through the analysis of the Prolog prototype. Inheritance is an essential means in object oriented languages to express inclusion polymorphism (i.e., subtyping). Thus, to infer inheritance relationships from a Prolog program, we define a polymorphic type system for Prolog, and identify subtyping relations between these types.

We associate a type with each predicate and with each argument position in a predicate. The type of a term at position[1] i in a predicate p is noted $Tp(i)$, and is defined by:

$$Tp(i) = \{A_i \,/\, p(\ldots,A_i,\ldots) \in \textit{denotation}(p) \text{ and } A_i \text{ is a variable or a constant}\}$$
$$\cup \{ f(A_1, \ldots, A_n) \,/\, A_1 \in Tp(i.f(1)) \text{ and } \ldots \text{ and } A_n \in Tp(i.f(n)) \}$$

The type of a predicate p of arity $n \geq 1$ is noted Tp, and is defined by:

$$Tp = \{ p(A_1, \ldots, A_n) \,/\, A_1 \in Tp(1) \text{ and } \ldots \text{ and } A_n \in Tp(n) \}$$

In order to get independant of the test cases, we propose in [1] an algorithm that defines the types of a Prolog program by their relations with other types rather than by instances sets. The different kinds of type relations infered by this algorithm are:

- $Tp = p(Tp(1) \times \ldots \times Tp(n))$ that specifies that the type Tp of a predicate or a functional term p/n is defined by the composition of the types of its arguments $Tp(1)$, ..., $Tp(n)$.
- $T = T1 \cup \ldots \cup Tn$ that specifies that the type of an argument of a predicate is defined by the disjunction of the types of this argument in the different clauses that define this predicate.
- $T = T1 \cap \ldots \cap Tn$ that specifies that the type of a variable in the head of a clause is defined by the conjunction of the types associated with the occurences of this variable in the body of the clause.

[1] This position can be indexed in order to take into account functional terms which arity is different from zero: the position of a term X in a functional term $A=p(A1,\ldots,An)$ is computed by using the following rules:
- $Position(X,A) = i$ if $X = Ai$
- $Position(X,A) = i.f(Position(X,Ai))$ if X appears in a functional term $Ai = f(..,X,..)$

- $T \subseteq T1 \cap \ldots \cap Tn$ that specifies that the type of a variable in the head of a clause is a specialization of the conjunction of the types associated with the occurences of this variable in the body of the clause. Indeed, the type of a variable can be restricted by another variable that occurs more than once in the body of the clause, a constant, or a functional term.
- $T = var\_type(X)$ that specifies that the type of a variable X that only occurs in the head of a clause is a parametric type.
- $T = instance(c)$ that specifies that a constant c is instance of a type T.

From type relations, we infer inheritance relationships: $T1$ is an heir (i.e., a specialization) of $T2$ if $T1 \subseteq T2$. We compare types by using the two following rules:

$T1 \cup \ldots \cup TN \subseteq U1 \cup \ldots \cup UM$ if $\forall i \in 1..N, \exists j \in 1..M$ so that $Ti \subseteq Uj$
$T1 \cap \ldots \cap TN \subseteq U1 \cap \ldots \cap UM$ if $\forall i \in 1..M, \exists j \in 1..N$ so that $Tj \subseteq Ui$

The infered inheritance hierarchy is independant from the test cases: we do not consider the intances occuring in the type relations for comparing types (i.e., we do not suppose that the identity — resp. inclusion — of the sets of instances associated with two types implies the identity — resp. inclusion — of these types). We also infer an inheritance hierarchy dependant from test cases. For this, we take into account the instances occuring in the type relations when comparing types.

These two inheritance hierarchies exhibit the program structure at two different levels of abstraction: the first one explicits the relations expressed by "functional clauses" (i.e., clauses that describe the application), where as the second one explicits the relations expressed by "data clauses" (i.e., clauses that describe the data of the application). These inheritance hierarchies can be used to design an object oriented model of the application. Moreover, their comparison can outline an inconsistency between the implementation of the functionalities and the test cases. Thus, we provide a support to improve the Prolog prototype and to clarify the underlying data structure of the application to be developed. Our approach also enables one to take advantage of both logic and object-oriented paradigms and provides an alternative approach to the systems based on the integration of both paradigms.

The whole system has been implemented in PrologIII. Constraints on tuples allowed us to write a reversible parser in a very declarative way, whereas constraints on booleans provide a powerful and well suited mechanism for identifying inheritance relationships from type relations. Despite of some limits (e.g., difficulty for processing the resulting constraint system), PrologIII offered significantly more prototyping facilities than Prolog.

Different extensions of this work are considered. First of all, we plan to extend our system in order to better deal with generic clauses. In some cases, all type information is not present in the Prolog program. Thus, the infered inheritance hierarchy has to be interactively refined by the user, and a graphical editor would be useful. Further works concern also the reuse of the deduced type information in the Prolog prototype (e.g., introduction of types by means of constraints).

References

[1] C. Solnon, M. Rueher: *From a Prolog prototype to an object oriented model: an approach using relationships between types*, RR 92-21, I3S, 4 av. A. Einstein, 06560 Valbonne, France, 15p