THE IDT SYSTEM AND ITS APPLICATION FOR LEARNING PROLOG PROGRAMS

GABRIELLA KÓKAI
Department of Informatics J¢zsef Attila University Hungary
H-6720 Szeged, Árpád tér 2. e-mail:kokai@inf.u-szeged.hu

ZOLTÁN ALEXIN
Research Group on the Theory of Automata at the Hungarian Academy of Sciences
H-6720 Szeged, Aradi vértanúk tere 1. e-mail: h157ale@ella.hu

FERENC KOCSIS
Research Group on the Theory of Automata at the Hungarian Academy of Sciences
H-6720 Szeged, Aradi vértanúk tere 1. e-mail: h155koc@ella.hu

ABSTRACT

In this paper the IDT (Interactive Diagnosis and Testing) System is presented, which combines the Shapiro's Interactive Diagnosis Algorithm with Category Partition Testing Method. This system can be used for testing, debugging and learning of Prolog programs.

1. Introduction

In this paper we present a system (called IDT) which combines Shapiro's Interactive Diagnosis Algorithm [SHA83] with CPM (Category Partition Method) [OSB88]. This system can be used in the testing, debugging and learning of Prolog programs.

Shapiro's Interactive Diagnosis Algorithm was originally applied to Prolog programs to diagnose the following three types of errors: termination with incorrect output, termination with missing output, and nontermination. The main problem with this diagnosis algorithm is that the user (oracle) has to answer many queries during the diagnosis process.

These questions may be very difficult to answer, therefore the application of this diagnosis method to practical problems is questionable. Shapiro suggests that 'query complexity' of this algorithm can be improved by using test results. However, to our knowledge the IDT system is the first integrated tool for testing, debugging and learning of Prolog programs.

The basic concept of IDT is very similar to GADT (Generalized Algorithmic Debugging and Testing) presented in [FGK91]. In GADT the CPM (Category Partition Testing Method) [OSB88] was combined with an algorithmic debugging method to localize a bug in the program using minimal number of user interactions. In CPM the tester has to define equivalence classes called *test frames* over the input domain. There is an assumption that the behaviour of a test frame from the point of view of the testing process can be represented by an arbitrary element of it (called *test case*).

The IDT method for logic programs is presented in [HGA93]. This method can be used for both debugging and testing of Prolog programs. The main difference between GADT and IDT is that GADT has been applied to imperative languages and IDT to logic programs.

In the debugging process the concrete values of parameters of procedures are given. By determining the corresponding test frame to a given input the test database can be checked against the selected frame. In the case of successful test the debugger skips to the next procedure without a query to the user.

* This work is supported by OTKA-501 and ESPRIT BRA 6020.

A further improvement for bug-localization process is presented in [PGH93]. In that paper a method was introduced which combines the IDT with program slicing. The close relationship between diagnosis algorithm and inductive learning of logic programs has been reported in [SHA83]. Therefore we can apply the IDT system not only in the debugging and testing but in the learning of logic programs as well.

In the paper first we give a description of the IDT system. Section 4 contains the inductive inference algorithm which synthesizes logic programs from positive and negative samples. The algorithm combines the diagnosis algorithm with CPM methods. Finally, in Section 5 some remarks for the future work is given.

2. Description of the IDT system.

In this section first we give a short overview of CPM and the implementation of this method in Prolog environment.

2.1. Category Partition Testing Method

The CPM method has been defined in [OSB88]. A formal description of this method can be found in [HGA93]. During the process of functional testing, the programs (procedures) cannot be tested with all possible properties of the input parameters. Hence, the tester's first task is to define the critical properties of parameters. These critical properties - called *categories* - are investigated in the testing process.

The categories can be divided into classes - called *choices* - presuming that the behavior of the elements of one choice is identical from the point of view of the testing.

If the categories and choices for a program have been defined, then all the possible *test frames* can be generated. A test frame contains exactly one choice from each category.

Example 2.1: The category-partition specification for the predicate *member*:

member(X,[X³_]).
member(X,[Y³Z]) :- member(X,Z).

Test specification: member
Category: number_of_the_elements
empty**:** **property** empty
{The second argument is an empty list}
one**:** **property** one
{The second argument has exactly one element}
more**:** **property** more
{The second argument has more than one element}
Category: position_in_the_list
first**:** **if not** empty
{The first argument and the first element of the second argument are the same.}
last**:** **if** more
{The first argument and the last element of the second argument are the same}
inside**:** **if** more
{The first argument can be found in the second argument but it is neither the first nor the last element of the list}
none**:** **if not** empty
{The first argument can not be found in the second argument}
End of specification

The generated test frames are: {(empty,_), (one,first), (one,none), (more,first), (more,last), (more,inside), (more,none)}

2.2. The implementation of CPM in Prolog

2.2.1. Preparing the initial test database

The test database is generated by a Prolog program. The input of the program is a test specification which is given in a Prolog form. The descriptions are Prolog facts in the following form:

choice_of(category_name_1,choice_name_1).
choice_of(category_name_1,choice_name_2).
...
choice_of(category_name_n,choice_name_n_1).
choice_of(category_name_n,choice_name_n_2).
...

From the loaded CPM description the program generates the initial test database. The elements of test database are Prolog facts in the following form: *test* (P, A, F, IO, E), where P is the predicate name, A is the predicate arity, F is the test frame (represented by a list of corresponding choices), IO is the list of input and output values of representing elements, E is the evaluation of the test frame.

2.2.2. The searching functions:

In the IDT system searching functions have been introduced to decide which test frame contains the given input. For every choice name a searching function can be defined. If the searching function is not defined the selection of the proper testframe is perfomed via menu. The searching functions are Prolog predicates.

Example 2.2. The test database for *member*:

Test frames:

```
test(member,2,[empty,_],[],undefined).
test(member,2,[one,first],[([a,[a]],[a,[a]])],true).
test(member,2,[one,none],[([a,[b]],[a,[b]])],false).
test(member,2,[more,first],[],undefined).
test(member,2,[more,last],[([b,[a,b]],[b,[a,b]])],true).
test(member,2,[more,inside],[],undefined).
test(member,2,[more, none],[],undefined).
```

Searching functions

```
empty(_,[]).
one(_,[_]).
more(_,[In|_]).
first(In,[In|_])

last(X,[Y|Ys]):- X \== Y, last(X,Ys),
last(X,[X]).

inside(X,Y):- member(X,Y), \+ first(X,Y), \+ last(X,Y).

none(X,Y):- \+ member(X,Y).
```

2.2.3 Searching in the test database

During the debugging we need the evaluation of some test frames. The *geteval*/5 predicate can return the CPM evaluation of an arbitrary input. First it finds the test frame which the given input belongs to then it returns the evaluation of this test frame.

An input is in a test frame iff all searching functions, associated with the choices of the test frame return true for this input.

Algorithm 1. Searching in the test database

Input: the procedure P, its arity A and the input I.

Output: The test frame Fe which contains this input and the evaluation value E of this test frame.

Algorithm: find all test frames in the test database then selects those test frames for which all searching functions return true.

2.2.4. The modification of the database

If the program finds the test frame which contains the input parameter and its evaluation is *undefined* then it modifies the test database in the following manner.

Algorithm 2. The modification of test database

Input: The fact P and the predicate name F of this fact with its arity, A, the input I and the frame C.

Output: The new evaluation S.

Algorithm: The program solves the fact and asks the user if the output is correct or not. There are two cases depending on the length of input/output list. If it is empty the program retracts the item and asserts a new item where the new input/output list has one element and the evaluation will be the answer of the user. In the other case the new input/output pair will be inserted into the input/output list.

3.Shapiro's PDS (Program Diagnosis System)

3.1. Modified Single Stepping with Category Partition Method

Now we are ready to modify the Shapiro's Single Stepping method. The idea of modification is that the algorithm does not ask the user about the correctness of each resolved goal if it finds the evaluation of this goal in the database for test frames. The original description of the algorithm and the program written in Prolog can be found in [SHA83]. The advantage of our method can be seen if we run it on the predicate *member* described above:

Example 3.3. The advantage of modified single stepping algorithm.

```
| ?- fpx(member(x, [a,a,a,a,a,b] ),X).
Is it ok [member(a,[a,b])] (y/n) y
Is it ok [member(x,[a,a,a,a,a,b])] (y/n) n

X = (member(x,[a,a,a,a,a,b]):-member(a,[a,a,a,a,b])) ?

yes
```

In this example we supposed that the system uses the test database presented in Example 2.2. Therefore the IDT system first gives a question for the input *(a,[a,b])* because the test frame *[more, first]* was *undefined* (see Example 2.2). However after the answer *yes* the evaluation of this test frame changed to *true*. After that the system does not give more questions to this frame. The state of the test database after running the Example 3.3 can be seen below.

| ?- listing(test/5)

test(member, 2,[empty, []], [([a, []], [a, []])], false).
test(member, 2,[one, first], [([a, [a]], [a, [a]])], true).
test(member, 2,[one,none], [([a, [b]], [a, [b]])], false).
test(member, 2,[more, last], [([b, [a,b]], [b, [a,b]])], true).
test(member, 2,[more, inside], [], undefined).
test(member, 2,[more, first], [([a, [a,b]], [a, [a,b]])], true).
test(member, 2,[more,none], [([x, [a,a,a,a,a,b]], [x, [a,a,a,a,a,b]])], false).

yes

4. The improved inductive program synthesis

In this section we apply the CPM method for improving the inductive program synthesis algorithm published in [SHA83]. A detailed description of the concept and implementation (MIS Model Inference System) of this algorithm can be found in [SHA83]. Now we give an informal description which is helps to understand our improvement.

4.1. The inductive inference algorithm of Shapiro

A presentation of a program ***P*** is a (possibly infinite) sequence of input/output samples of ***P*** in which every input in the domain of ***P*** eventually appears. Assume that the algorithm is given an initial program (normally the *empty* program) and a presentation of some target program. The inference algorithm reads the samples one at a time, and performs modifications to the initial program as it is necessary. The inference algorithm is said to identify the target program in the limit, if eventually there comes a time when the modifications it perform results in a program with the same input/output behavior as the target program, and it does not modify this program afterwards.

Note that within this definition, a inference algorithm based on patching alone will not identify a program in the limit if the initial program behaves incorrectly on an infinite number of inputs. The algorithm that has a fixed initial program (say, the empty program) is called an inductive inference algorithm.

4.2. The algorithm for inductive program synthesis

The original MIS (Machine Inference System) is published in [SHA83].

4.3. The modified MIS

if the program ***P*** succeeds on a goal known to be false
then detect a false clause ***p*** in ***P*** and remove ***p*** from ***P*** and mark it.

We modified the part of the Algorithm 3 listed above. Originally for detecting the false clause in the logic program the Shapiro's single stepping algorithm was used. Instead of this we introduced the modified single stepping algorithm (see Section 3.1) to avoid the questions which must be answered by the user.

Initially we have some knowledge about the behaviour of the program to be learned and the initial test database. All the evaluations in the test database are *undefined*.

We use in every step of the learning algorithm those frames of test database that are valid. We mean those frames that are consistent with some positive samples. If the algorithm produces a new program which is correct for every sample and we want to learn a new sample with the help of the inference system we must qualify again our test database.

5. Conclusion

In this paper a system has been presented which combines the Category Partition Testing with the algorithmic debugging techniques introduced in [SHA83]. A similar method is presented in [FGK91] to diagnose imperative programs but as far as we know the IDT system presented in this paper is unique in the context of logic programming. This integrated tool can be used in the testing, debugging and learning of Prolog programs.

The Category Partition Testing Method in Prolog environment have already been implemented. The integration of CPM with the single stepping and divide and query diagnosis algorithms are also implemented. The integration of CPM with the other two diagnosis algorithms and the MIS is currently under development.

6. References

[FGK91] Peter Fritzson, Tibor Gyimóthy, Mariam Kamkar, Nahid Shahmeri: *Generalized Algorithmic Debugging and Testing*. Proceedings of the ACM SIGPLAN '91 Conference on Programming Language Design and Implementation, Toronto, Ontario, Canada June 26-28, 1991. pp. 317-326.

[HGA93] Tamás Horváth, Tibor Gyimóthy, Zoltán Alexin, Ferenc Kocsis: Interactive Diagnosis and Testing of Logic Programs. Proceedings of the Third Symposium on programming languages and Software Tools Ed: Mati Tombak Kääriku Estonia, August 23. 1993.

[OSB88] Thomas J. Ostrand, Marc J. Balker: The Category-Partition Method for Specifying and Generating Functional Tests. CACM 31:6, June 1988. pp. 676-686.

[PGH93] Jukka Paakki, Tibor Gyimóthy, Tamás Horváth: An Integrated Method for Algorithmic Debugging of Logic Programs. in manuscript

[SHA83] E. Y. Shapiro: Algorithmic Program Debugging MIT Press 1983.