Abstract: The aims of this report are:
- specifying the requirements of the project.
- describing possible problems and solutions
- presenting the current architecture of a diagnosing system.
- description of every component.
- limitations and future works

1. Definition the aim of the project

The objective of the project INCOM is to develop an interactive learning environment for students who want to attend a logic programming course. This tool should help students learning Prolog programming with the presence of a human tutor. It is expected to be a diagnosis part of an intelligent teaching system, therefore we should limit the capabilities of INCOM as following:

-The system should provide an user interface with exercise description. Task statements are posed to the student, and he is then requested to write his solution.

-The system analyzes student solution and diagnoses errors by comparing it with a reference solution.

-Based on errors found by the diagnose components, feedbacks should be created in a manner so that students do not have to be stuck with his exercise.

INCOM is destined for use in the early phases of Prolog teaching, to relieve the Prolog tutor from spending too much time on criticizing answers to simple exercises and to let him concentrate more on development of initiative at the problem analysis and program design stage.

More precisely, INCOM is intended for the student who has learned the basics of Prolog syntax and semantics and who has some idea about control flow.

2. Description of problems and possible solutions

INCOM has to take three domains into consideration. First is the didactic issues. Second is the technical feasibility and the last one is the analysis of the application domain, namely logic programming. At this time we have not put much effort into researching the field of didactics. Currently our work has been centered to the problems of

Prolog. The technical feasibility infers from the application domain so that our solution will be very Prolog dependent.

a)Application domain: Prolog

The set of problems considered is divided into two categories: database query and recursion. First, we chose two exercises for database query from the exercise sheets of logic programming course and collected some typical exercises of recursion from different literature.

During the analysis process of database query we addressed the following Prolog specific concepts: type conflict, use of equation, use of instantiation, coreference, misspelling predicate name, missing subgoal, superfluous subgoal, swapped subgoals, missing argument, superfluous argument, swapped arguments.

In addition to the problems of database query, the analysis of recursion results with other issues: missing clause, superfluous clause, subgoal order, clause order, termination problem, use of techniques (see section INCOM architecture), predicate definition returns wrong solution, predicate definition fails to return some true solution.

The concepts of type conflict, use of equation, use of instantiation, coreference will be handled by a type checker component. The issues of misspelling predicate name, missing subgoal, superfluous subgoal, swapped subgoals, missing argument, superfluous argument, swapped arguments will be taken in consideration in a structure analyzer component.

The special issues of recursion such as missing clause, superfluous clause, subgoal order, use of techniques will be examined by an idiom detector (see section Architecture of INCOM). The rest of recursion specific aspects such as clause order, termination problem, predicate definition returns false solution, predicate definition fails to return some true solution will be covered by a dynamic analyzer (see Future works d)).

Although a syntax verification seems to be not necessary because every Prolog compiler does it already, but we think that the common mistyped errors like ".", ",", "[", "]", "(", ")" should be caught. This can help Prolog learner a lot. So, a syntax checker component should be taken into account.

b)Technical domain

The technical domain deals with the questions: how can application specific requirements be modeled and realized? Is the technical solution transferable to other application domains?

The technical feasibility has a lot of contribution for the didactic domain. The more quality of output the analyzer components returns the better value of feedbacks for novice programmer.

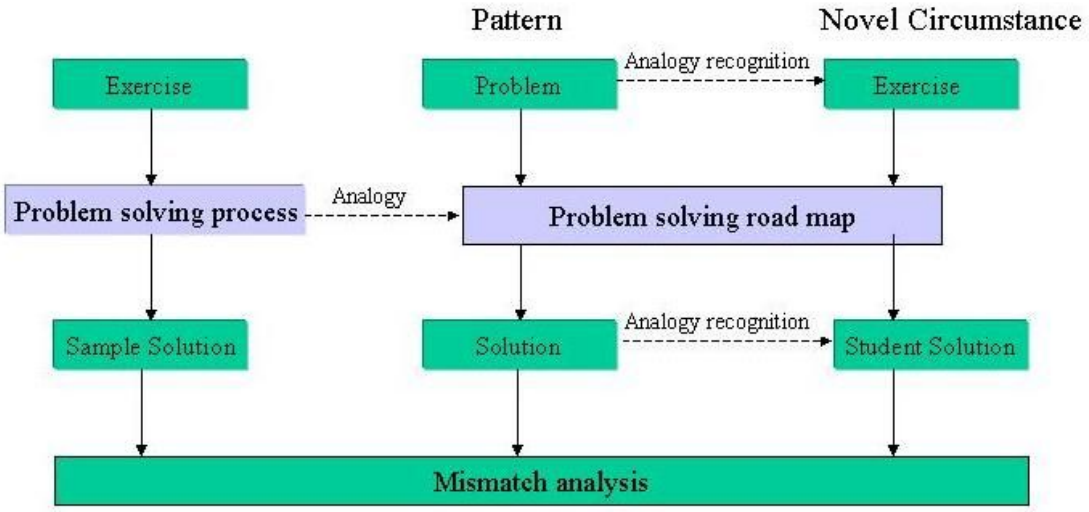

Figure 1: Solution strategy

For database query exercises we will compare the student solution with an appropriate sample solution and find the mismatches between them. The mismatched information will be forwarded to a pedagogical component where feedbacks should be generated.

For recursion exercises we need more information in order to deliver more task-specific explanations. These added-value informations can be gathered by comparing the student and sample solution via a pattern. A pattern is a named problem/solution pair that can be applied in new contexts, with advice on how to apply it in novel situations ([Larman98]). Based on the specification of a pattern we can determine what kind of mistake the student has made and how he should remove it. (See Figure 1: Solution strategy).

c) Didactic domain

First, we have to determine what kind of pedagogical effect we want to achieve. Researchers of the field Intelligent tutoring Systems (ITS) try to build a student model that represents the current state of student knowledge. With an ITS the state of student knowledge should be extended and reach the state of expert. Currently we can find following student models in literature: overlay student model, differentiate student model, perturbation student model ([HoDuJoGr94]). All these models assume that the expert knowledge is limited. But in reality during the teaching process the expert knowledge can be extended by communication with students. This aspect should be taken into account when trying to build a student model.

With this project we intend to build a diagnosis system for helping students overriding the blockades of doing Prolog exercises in the early phase. The system will be applied in a tutor session with presence of a human tutor. This system is not intended to replace a tutorial of Prolog.

The mismatch informations which are gathered from different analyzers will be used by a pedagogical component in order to find appropriate feedbacks. A feedback should contain three kinds of information: where in the input is error located, what kind of error and hints how it should be solved. There are two kinds of feedbacks. The first class is the conceptual dependent and the other class is exercise dependent.

In addition to returning a feedback, the pedagogical component should bold the error position in the student's input.

3. Architecture of INCOM

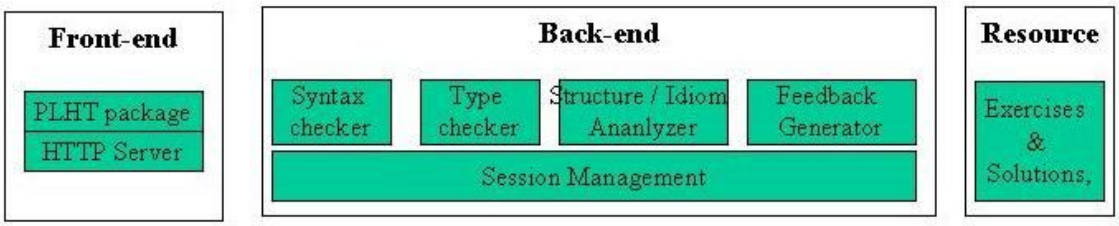

Figure 2: Architecture of INCOM

The architecture of INCOM can be divided into three layers: front-end, back-end and resource layer.

The front-end layer is responsible for reading solutions to a task statement and representing feedbacks. Front-end works on the HTTP server of SWI-Prolog. With a PLHT library dynamic HTML pages can be created and are able to call application logic written in Prolog.

Currently PLHT library provides means to execute following control structures: looping, if-then condition, writing atoms and terms, calling a Prolog predicate.

The back-end layer is responsible for analyzing student solution and returning instructional feedbacks. The analysis process can be divided into two phases. The first step of process begins with task-independent analysis. The basic concepts of Prolog will be examined in the student solution. The second step is a task-dependent analysis. It will compare the student solution with an appropriate sample solution. For recursion exercises we will find mismatches between student solution and sample solution by the means of a pattern specification.
For the task-independent analysis we need two components: syntax checker and type checker.

The syntax checker verifies the student's input if it is syntax conform and brings the student solution to an internal normal form. It can find following syntax errors:
"." , "[", "]", "(", ")", "," is missing or superfluous
forgotten_argument in case "a()".

The type checker verifies whether there is no type conflict within the student's input. (In addition to type mismatches the type checker examine following conceptual shortcomings:
???)

For database query exercises we have a structure analyzer component compares student solution with sample solution and produces following possible results:

-alternated argument positions
-missing arguments
-superfluous arguments
-alternated subgoal positions
-missing subgoals
-superfluous subgoals
-coreference error
-incorrect use of (in-)equations.

For list recursion exercises we have a component which recognizes the pattern of the student solution and then compares it with it's sample solution. This component can find following bugs:

- missing a base case
- missing a recursive case
- superfluous base

- superfluous recursive case
- false subgoal order (techniques same, before, after)
- misuse of list composition/ decomposition (techniques list_head, list_subgoal)
- misuse argument types (input, output, info, accu)

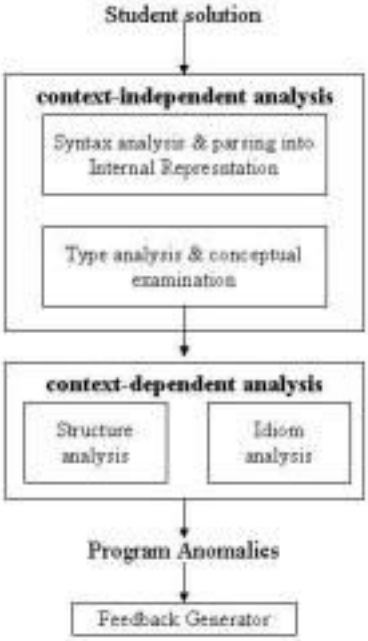

Figure 3INCOM Analysis flow

The last component of the back-end layer is the feedback generator. After the analysis process is finished, all errors found will be forwarded to the feedback generator which then return appropriate instructional hints.

The last layer of the INCOM architecture contains exercise descriptions and sample solutions that are saved in form of XML. For every exercise domain we have to define internal types for predicates and arguments. For database query exercises we need a fact database. The last two informations are stored in normal Prolog knowledge databases.

4. Limitations

Most intelligent tutoring systems focus on analyzing student solution and generating didactic feedbacks, but not take work load for tutors into account. The task of human tutor is to think up new exercises and sample solutions and put them into database. INCOM tries to limit works for tutor as much as possible. Currently, if a tutor wants to publish new exercise, he has to put this exercise and sample solutions into a XML file. If it is a database query exercise, he has to define new

database of necessary facts and corresponding types. A user interface for creation or modification of new exercises is desirable.

XPCE web server has single threading so that just one user can make a request to the web server. Beside this limitation, PLHT library is not thread safe, and therefore it cannot be used with a threading web server. The session management module is still so weak that sessions will never timeout.

Iterative deepening search is the main algorithm which we use for INCOM in order to check syntax conformity, type conflicts and compare structures of student's and sample solution. Unfortunately, the search space will grow too big if we miss an open or closing parenthesis so that the search will fails. This has to be investigated and improved.

Prolog predicates such as "not/1", "!/0" are not taken in consideration at this moment.

5. Future works

a) The syntax analysis will produce several solutions if an open parenthesis is missing. For example the input entfernen([X|A],Z,X|B]) can be interpreted as:
entfernen([[X|A],Z,X|B]), or
entfernen([X|A],[Z,X|B]), or
entfernen([X|A],Z,[X|B]).

With the type checker the types of expected inputs are determined which helps to choose the appropriate result from syntax checker.

b) As described above the algorithm of iterative deepening search seems to be not suitable for analyzing syntax. We have to test another search algorithm like A* best-first search.

c) At this moment we just have one sample solution for analyzing student solution. The range of possible student solutions is huge so that we have to specify new patterns for list and arithmetic recursion problems. A pattern is a road map of solution for a class of problems. The wider is this class the more general is the solution pattern and the less task-specific our feedbacks to the students will be. Therefore we should examine at which generality a pattern should be defined.

d) A component for doing dynamic analysis is required. It is possible that the structure of student solution is not similar to a pattern but it could produce same results as intended. We need to run the student solution on a Prolog compiler with some test cases and compare the

results with a sample solution. If the is result mismatch, then we can begin with the static analysis as described above.

6. Literature:

[Larman98] Craig Larman; Applying UML and Patterns, Prentice Hall PTR 1998.
[HoDuJoGr94] Peter Holt, Shelli Dubs, Marlene Jones, Jim Greer; “The State of Student Modelling” in Student Models: The Key to Individualized Education Systems; Springer Verlag 1994, 1-35.