Component Specification

I. StructureMatcher

Each solution represents a strategy or an algorithm. So it's a particular way of solving a problem. This is what we call a solution pattern.

Each pattern has a structure and its constraints. The pattern specific constraints describe the use of techniques in this pattern. A pattern also defines a class of exercises which have the same strategy for the problem's solution. Bugs at this level reflect lack of understanding of the task, or an incorrect or inefficient strategy.

We divide the pattern mating process into two steps. The first step is looking for the best pattern structure. The second step check the pattern and exercise specific constraints.

1. Pattern structure matching

A best pattern structure should be found for a given student solution. If a pattern structure can be found, that means that the system knows what kind of strategy the student follows. Otherwise, the system will tell the user “Sorry, i can't help you. Please choose one of the corresponding patterns for this exercise.” A page with available patterns for this exercise will be shown to the user.

If a pattern structure is found and some errors occur. That means that the user knows the solution strategy but he can not remember how to construct this strategy correctly. He can use more or less arguments, subgoals or clauses than needed. Errors occur here reflect lack of the mastery of this strategy. Help on this level can link to a page where the pattern for this exercise is described.

Following are the possible errors which will occur while pattern matching process. Penalty for an error is defined based on the the level of matching: errors occur on clause level will get a penalty of 20 points, on the subgoal level 15 points and on the argument level 10 points.

Penalty range [1..5] is used for constraints.

Penalty on the IDS level is only used for internal searching process. Penalty on the feedback level is used as a priority to correct the errors.

Errors while matching:

Error	**Penalty Ids**	**Penalty Feedback**	**Error description**
match_bags/ match_clause/ superfluous/ missing	20	20	One or more clauses are missing or superfluous
match_bags/ match_term/ superfluous/ missing	10	15	One or more subgoals are missing or superfluous
match_arg, unmatchable	5	10	An argument of the student solution and an argument of the pattern can not be matched.
superfluous_list_element (match_list_struct)	5	10	One or more arguments are superfluous.
missing_list_element (match_list_struct)	5	10	One or more arguments are expected.

Errors while typifying:

Error	**Penalty Ids**	**Penalty Feedback**	**Error description**
misspelled_predicate (Functor1,Functor)	5	15	There are predefined predicates: append, member, not,=,\=. If in any predicate definition uses theses predefined predicates misspelt, this kind of error will be found and suggests a correct predefined prediacte name.
unknown_predicate (Functor)	30	15	The predicate which is used for another predicate definition is unknown.
conflicting_type (Varname,Type)	5	10	A variable has two different types.
must_be_number(Var)	5	10	The found argument must have type of number
no_arithmetic_function (X)	5	15	The term X is expected to be an arithmetic term but X does not unifies with any current_arithmetic_function.
match_lists/typify_var/type_expected	10	10	

Error	Penalty Ids	Penalty Feedback	Error description
lists_do_not_match	30	15	The argument list does not match a given type list.
forgotten_list_element	5	10	The argument list can match a given type list but one or more arguments are expected.
misplaced_list_element	5	10	The argument list can match a given type list but two arguments are misplaced
superfluous_list_element	10	10	The argument list can match a given type list but one or more arguments are superfluous

2. Pattern constraint checking
   A pattern is found. There are pattern specific constraints which will be evaluated. Every constraint has its penalty. The pattern which has the least penalty sum is the most candidate for the corresponding student solution.

Constraint checking may produce errors of the following form:

Error=idiom-error([Pos1], Term, ErrorExplanation, Penalty),
Error=idiom-error([Pos1,Pos2], Term, ErrorExplanation, Penalty)

Errors of this form supply information about location, type, explanation and penalty points of error which will be used in FeedbackHandler component.

II. Diagnose_rec

   the first step of diagnose is checking the input of student on the syntax level. This is done by using the Prolog compiler (consumeAtom_rec, atom_to_term).

If the input is not syntax conform, then the diagnose process will be stopped and returns the syntax errors to the user.

A syntax error returned from the compiler has following form:

error(syntax_error(operator_expected), string("deleteall([H|T],A,Result):-
H=>A,deleteall(T,A,Result) . ", 28))

If the input is syntax conform the diagnose continues with looking for the best pattern.
After the best pattern has been found. A list of errors from matching and constraint analyzing process will be collected.

Errors from the matching process will be converted to matching_errors.

Error	matching_error	Penalty
match_bags/ match_clause/ missing	matching_error(Position, missing_basecase(Clause))	20
match_bags/ match_clause/ missing	matching_error(Position, missing_recursivecase(Clause))	20
match_bags/ match_clause/superfluous	matching_error(Position, superfluous_basecase(Clause))	20
match_bags/ match_clause/ superfluous	matching_error(Position, superfluous_recursivecase(Clause))	20
missing_list_element (match_list_struct)	matching_error(Position, missing_arg (Arg))	10
superfluous_list_element (match_list_struct)	matching_error(Position, superfluous_arg (Arg))	10
match_bags/ match_term/ missing	matching_error(Position, missing_subgoal (X))	15
match_bags/ match_term/superfluous	matching_error(Position, superfluous_subgoal(X))	15
match_arg, unmatchable	matching_error(Position, arg_unmatchable (Arg1,Arg2))	10
use_X_not_Y (RealFunctor,Functor)	matching_error(Position, wrong_right_func(Wrong,Right))	10
misspelled_predicate (Functor1,Functor)	matching_error(Position, misspelled_predicate(Wrong, Right))	15
unknown_predicate (Functor)	matching_error(Position, unknown_predicate(Functor))	15
conflicting_type (Varname,Type)	matching_error(Position,conflicting_type (Varname,Type))	10
must_be_number(Var)	matching_error(Position, must_be_number (Var))	10
no_arithmetic_function(X)	matching_error(Position, no_arithmetic_function(X))	15
lists_do_not_match, forgotten_list_element, misplaced_list_element, superfluous_list_element match_lists/typify_var/type_expected	matching_error(Position, general_type_error)	10

Examples of wrong codes and their explanation:

Code	Error	Explanation
no_doubles([X|Xs],[X|Ys]):- not (member(X,3)), no_doubles (Xs,Ys).	matching_error([1,1,1,2], misplaced_argument_type (1,X,2,lst(number))	X is guessed to be of type list, so it should be member (3,X)
no_doubles([X|Xs],[X|Ys]):- not (member(X,3,Xs)), no_doubles (Xs,Ys).	matching_error([1,1,2], arg_unmatchable(member (X,3,Xs),op2(V9,V10)),10),	Member has two much arguments
no_doubles([X|Xs],Ys):- ember (X,Xs), no_doubles(Xs,Ys).	matching_error([1,3], misspelled_predicate (ember,member),15)	Ember (X,Xs) is misspelled. It should be member
no_doubles([X|Xs],[X|Ys]):- not (member(X,Xs)), no_doubles (Xs,Ys,5).	matching_error([3,2,2], superfluous_argument_type(3,5)	The argument 5 is superfluous in no_doubles
no_doubles([X|Xs]):- ember(X,Xs), no_doubles(Xs,Ys).	matching_error([0,3], missing_arg(V17),10)	no_doubles has one argument. One argument is missing.