An Intelligent Environment for Learning Techniques and Strategies of Solving Combinatorial Problems

Valeriy N. Kuchuganov[1], Denis R. Kasimov[2]
CAD Systems Department
Kalashnikov Izhevsk State Technical University
Izhevsk, Russia
[1]kuchuganov@istu.ru, [2]kasden@mail.ru

***Abstract*—An important direction of improving the process of algorithmization training is developing interactive declarative programming environments. The purpose of the study is to develop a tutoring system on programming, which allows describing and checking techniques and strategies of solving combinatorial problems in an obvious way. The proposed programming system allows a user to describe conditions of a problem and a solving strategy in a natural way, carries out automatic searching for solutions and their assessing, and performs visual simulation of solutions in order to confirm their correctness. Programming a problem switches from the level of direct planning actions to the level of choosing a strategy of allocating resources and jobs. An example of solving a problem is provided.**

Keywords—combinatorial problem; solver; knowledge; strategy; plan; state; situation

I. INTRODUCTION

Evolution of programming learning technologies is based, in our view, on the increasing the level of declarativeness of programming.

The well-known purely declarative programming language is Prolog. However, it is not widely used due to the following shortcomings:

- it is not very convenient for performing arithmetic calculations;
- sometimes an obvious way of describing a predicate is not effective; a programmer should know and apply methods of optimization of predicates;
- description of some entities is complicated in comparison with embodiments in other programming languages;
- input and output is not always easy to organize;
- data structures created at previous steps can not be modified in the future;
- the cut operator undermines ideology of the language;
- debugging is complicated.

The programming language GOLOG [1] for creating intelligent agents extends traditional imperative programming by logical reasoning about actions, which an agent can perform, on the base of situational calculus. This language has the following constructs: primitive action, waiting/checking a condition, sequence, nondeterministic branching, nondeterministic choise of an argument, nondeterministic iteration, conditional operator, loop with precondition, and subroutine call. In extensions of the language GOLOG there are also additional constructs. In the work [2] it is stated that the considered language causes difficulties for inexperienced users who do not have knowledge of mathematical logic and the language Prolog.

STRIPS-like planning languages (ADL, PDDL) [3, 4] are oriented to synthesizing sequences of actions leading to a goal. The world's state in these languages is described in the form of a conjunction of statements, which is unlikely easily perceived by non-specialists.

In the system G2 [5] for creating real-time expert applications programming is carried out in a structured natural language: properties and behavior of objects, relations between them as well as production rules and procedures are described. A solution of a problem is produced by a logic inference machine. G2 is not oriented to solving combinatorial problems and is complex to learn. The same can be said about the analogue of G2 – the system CLIPS [6].

This paper's goal is to develop a tutoring system for learning programming, which allows describing and checking techniques and strategies of solving combinatorial problems in an obvious way.

In the developed by us system, a user represents a problem's conditions in the form of appropriate semantic knowledge categories and specifies a solving strategy in the specially developed language SDL (Strategy Description Language). The user is provided by libraries of terms' definitions and implementations.

A strategy is represented as a sequence of points of selecting solutions and actions on implementing them [7]. The main differences from the systems considered above are the following:

- the operator *Select(What = , From = , How =)* has been introduced;
- control capabilities of loops have been extended;

This work was partially supported by the State assignment of the Ministry of Education and Science of the Russian Federation (Project No. 625).

© 2016. The authors - Published by Atlantis Press

- actions and things are described in a visual tabular form, where some attributes are designated as the desired ones;
- there are means of assessing alternative solutions;
- optimal solutions are automatically identified;
- there is the possibility of specifying the order of processing alternative solving ways.

For mastering the system it is enough to be able to program at least in one language like Pascal, C, Java, even without possessing skills in object-oriented programming. The system has a knowledge base, in which users' experience is accumulated in the form of descriptions of things, actions, relations, and scenarios. The interface of describing a problem is convenient for accumulating and retrieving knowledge about things, relations, actions, selection methods, situations, and strategies.

II. Description of the System

The system's knowledge is divided into the following semantic categories: property, object (thing, process), relation, compound object, action, situation, and scenario [8].

Programming a problem includes the following stages:

1. Describe things of the problem in a table.
2. Choose appropriate actions from a library or specify new ones by analogy.
3. Bind actions with things (executor, instrument, etc.).
4. Describe explicit and implicit relations (including rules and constraints).
5. Specify arithmetical and logical functions required at the previous stages.
6. Specify a strategy consisting of the problem's actions and the solver's actions (loops-quantifiers and generators of variants), which are also selected from a library.

Instructions on creating and processing variants are mainly given by the operators *Select* which, like the quantifier ∃, bind variables specified in attributes *What* with one of a set of permissible variants of values. The operator *Select* creates a branching point in the state tree. The operator has different semantics; the interpretation required by the user is specified in the attribute *How*:

- *CreatingAllPossibleSetsIncludingEmptyOnes*. In the attribute *From* an expression-selecting or a concrete set of things is specified. In the result of executing, a thing-group is added to the table of things; its attribute *Composition* contains selected participants of the group. The variable defined in the attribute *What* is binded with this created thing-group.
- *CreatingAllPossibleSetsOfGivenSize*.
- *CreatingAllPossibleNonEmptySets*.
- *SequentialEnumeration*.
- *BinaryEnumerationForSearchingMin*. This method in the first solving way selects center value in a given range, in the second alternative way – center value in the left subrange (if the first way has led to a correct solution) or in the right subrange (if the first way hasn't led to a correct solution), etc. The last variant is a minimum value, with which a solution is achieved.
- *BinaryEnumerationForSearchingMax.*
- *AllPossibleIntegerPartitioning*. Variants of the selection are sets of values of variables, which satisfy the equation $What_1 + \ldots + What_n = From$.
- *AllPossiblePartitioningIntoSubsetsIncludingEmptyOnes*. In the attribute *From* a thing-group, expression-selecting or a concrete set of things is specified. In the result of executing, several things-groups are added to the table of things; their attributes *Composition* contain participants selected from the attribute *From*.
- *AllPossiblePartitioningIntoNonEmptySubsets*.
- Etc.

Executing an action of a problem is performed as follows. At first the solver generates variants of the action; multivariance occurs when several possible values are specified in some attribute in the description of the action. Each variant of the action is considered as an alternative solving way (branching in the state tree is formed). Variants that do not satisfy to the precondition of the action are removed. Then realization of a variant of the action is performed: a new state is created by copying the current one, operators specified in the action's implementation are performed. After that the post-condition is checked.

In general, in the course of solving the problem the solver generates variants of values, checks pre- and post-conditions of actions, executes actions' implementations, performs control of correspondence of formed states to given constraints, adds new states to the state tree, and assesses solving ways.

Algorithm 1 presents the pseudocode of the main procedure of searching for solutions:

Algorithm 1 Main procedure of searching for solutions.

```
1. CurState ← InitialSituation
2. CurState.Op ← MainPlan.Operators[1]
3. Queue ← [ ]
4. repeat
5.   ContinueToPerformPlan(CurState)
6.   CurState ← Queue.ExtractNext()
7. until (CurState = nil) or (Interrupted)
8. DetermineOptimalSolutions()
```

The solver starts with processing the state corresponding to the initial situation of a problem. In each state the solver executes operators of a solving plan specified by a user. When executing operators *Select* as well as multivariate actions, the solver forms branching in the state tree and adds new states to

the processing queue. The order of processing states is a component of a solving strategy. If the order is *"in depth"*, the work continues always with the most recently created state; the solver explores each solving way entirely before switching to another way. In the case of the order *"in breadth"*, the work always continues with a state created earlier than others; the solver explores all possible solving ways simultaneously. The second order is reasonably to use for solving problems where some solving ways can be very long, but the way leading to the desired solution, on the contrary, is short, as well as for solving problems where it is necessary to determine the minimal amount of steps needed for achieving a goal.

The system has traditional for imperative programming environments debug tools: breakpoints, stepwise execution, viewing the current state including values of variables, parameters, attributes of things, etc.

III. Example

Consider the work of the system on the classic problem of a wolf, a goat and a cabbage. Fig. 1 shows the formal representation of this problem in the system.

	1 Name	2 No	3 Place	4 Cargo
1	Bank	1		
2	Bank	2		
3	Wolf		Bank[No = 1]	Yes
4	Goat		Bank[No = 1]	Yes
5	Cabbage		Bank[No = 1]	Yes
6	Waterman		Bank[No = 1]	Yes
7	Boat		Bank[No = 1]	Yes

(a)

	1 Name	2 Place
1	Wolf	Bank[No = 2]
2	Goat	Bank[No = 2]
3	Cabbage	Bank[No = 2]

(b)

	1 Name	2 Who	3 What	4 OnWhat	5 WhereFrom	6 WhereTo	7 Precondition	8 Implementation
1	Transport	Waterman	Group	Boat	Bank$A	Bank$B	Who.Place = WhereFrom; What.Place = WhereFrom; Count(What) <= 1; OnWhat.Place = WhereFrom; WhereFrom <> WhereTo;	Assign Who.Place := WhereTo; Assign What.Place := WhereTo; Assign OnWhat.Place := WhereTo;

(c)

```
1 (Goat.Place <> Wolf.Place) OR (Goat.Place = Waterman.Place);
2 (Cabbage.Place <> Goat.Place) OR (Cabbage.Place = Waterman.Place);
```

(d)

	1 Field name	2 Initial value	3 Estimation
1	NumberOfTransportations	0	Minimum

(e)

Fig. 1. Formalization of the problem: (a) things in the initial situation; (b) target situation; (c) actions; (d) constraints; (e) format of the protocol.

In the Algorithm 2 a possible plan of solving this problem is presented.

Algorithm 2 Simple plan of solving the problem.

```
1.  MainPlan
2.  {
3.    Repeat(While = not GoalAchieved)
4.    {
5.      Select(What = Group$A,
6.             From = Thing[Cargo = Yes],
7.             How = CreatingAllPossibleSetsIncluding
8.                   EmptyOnes)
9.      {
10.       PerformAction Transport(Who = Waterman,
11.                               What = Group$A);
12.       Increase Protocol.NumberOfTransportations
13.       By 1;
14.     }
15.   }
16. }
```

This strategy assumes repeated selection and transportation of a cargo from the current bank to another one until the target situation is achieved.

Fig. 2 shows results of solving the problem by the presented strategy: elapsed time – 47 ms, the number of processed states – 164, the count of processed solving ways – 106, the number of solutions found – 2, both are optimal by the number of required transportations. For achieving the target situation it is necessary to perform 7 transportations.

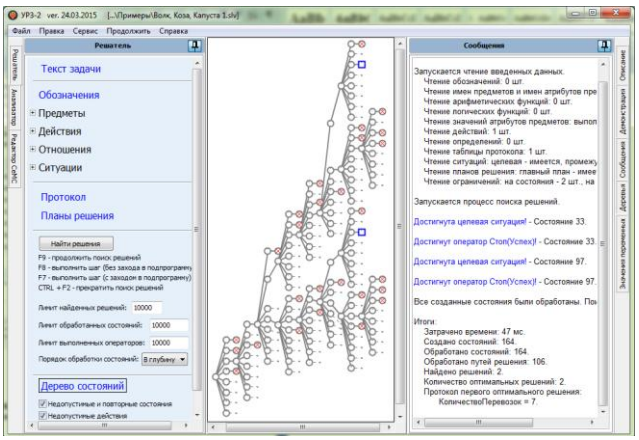

Fig. 2. Results obtained with help of Algorithm 2.

Another strategy of solving this problem is presented in Algorithm 3.

Algorithm 3 Second plan of solving the problem.

```
1.  MainPlan
2.  {
3.    Repeat
4.    {
5.      Select(What = Group$A,
6.             From = Thing[Cargo = Yes,
7.                          Place = Bank[No = 1]],
8.             How = CreatingAllPossibleSetsIncluding
9.                   EmptyOnes)
10.     {
11.       PerformAction Transport(Who = Waterman,
12.                               What = Group$A,
13.                                WhereFrom = Bank[No = 1],
14.                               WhereTo = Bank[No = 2]);
15.       Increase Protocol.NumberOfTransportations
16.       By 1;
17.
18.       If GoalAchieved Then
19.         Stop(Success)
```

```
20.         Else
21.         {
22.           Select(What = Group$B,
23.                  From = Thing[Cargo = Yes,
24.                               Place = Bank[No = 2]],
25.                  Which = (Group$B <> Group$A),
26.                  How = CreatingAllPossibleSets
27.                        IncludingEmptyOnes)
28.           {
29.             PerformAction Transport(Who = Waterman,
30.                           What = Group$A,
31.                           WhereFrom = Bank[No = 2],
32.                           WhereTo = Bank[No = 1]);
33.             Increase Protocol.NumberOfTransportations
34.             By 1;
35.         }
36.       }
37.     }
38.  }
```

The second strategy concretizes that it is necessary at first transport things from the left bank to the right one, and then – backward, at that the group transported back should differ from the group that has been delivered from the left bank.

Fig. 3 provides results of solving the problem by the second strategy: elapsed time – 32 ms, the number of processed states – 120, the number of processed solving ways – 49, 2 optimal solutions have been found, which are the same as in the first case.

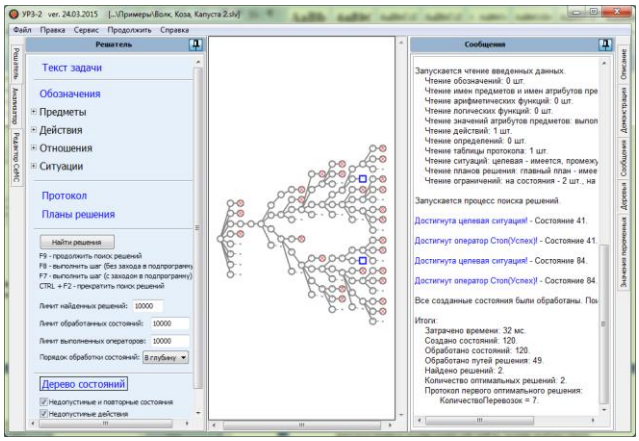

Fig. 3. Results obtained with help of Algorithm 3.

The statistics provided by the solver (figures 2, 3, right field) says that the smaller number of instructions in the first strategy results in automatic enumeration of the larger (approximately in 2 times) number of variants.

By this example we wanted to show that the system allows inputting and exploring various problem solving strategies, which can have elements of indeterminacy, be not completely concretized.

From the state tree it is possible to extract the sequence of actions by which the target situation is achieved from the initial one (Fig. 4).

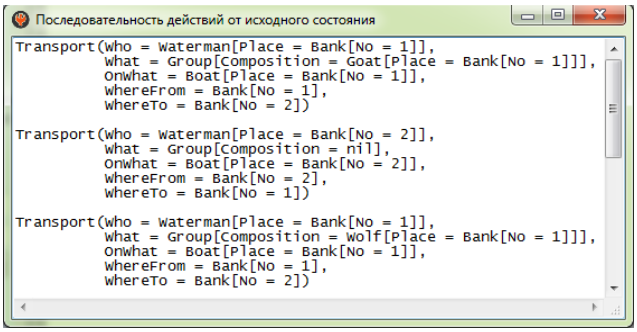

Fig. 4. Sequence of actions leading to the goal.

In the system we successfully solved problems proposed at international school Olympiads on programming [9].

IV. Conclusion

The proposed programming system allows a user to describe conditions of a problem and a solving strategy in a natural way, carries out automatic searching for solutions and their assessing, and performs visual simulation of solutions in order to confirm their correctness.

Strategies of solving the same problem, which are specified by different users, are usually differ by ways of selecting variants of arguments' values, by the order of processing states, and ways of organization of cycles. All these nuances affect final performance indicators of a user's strategy.

So, programming a problem switches from the level of direct planning actions to the level of choosing a strategy of allocating resources and jobs.

References

[1] H.J. Levesque, R. Reiter, Y. Lespérance, F. Lin, and R.B. Scherl, "GOLOG: A logic programming language for dynamic domains," The Journal of Logic Programming, vol. 31, no.1-3, pp. 59-83, 1997.

[2] A. Ferrein, G. Steinbauer, and S. Vassos, "Action-Based Imperative Programming with YAGI," Proceedings of the 8th International Cognitive Robotics Workshop at AAAI-12, pp. 24-31, 2012.

[3] R.E. Fikes and N.J. Nilsson, "STRIPS: a new approach to the application of theorem proving to problem solving," Proceedings of the 2nd International Joint Conference on Artificial Intelligence (IJCAI'71), pp. 189-208, 1971.

[4] S.J. Russell and P. Norvig, Artificial Intelligence: A Modern Approach, 3rd ed. Prentice-Hall, Upper Saddle River, NJ, 2009.

[5] R. Moore, P. Lindenfilzer, L. Hawkinson, and B. Matthews, "Process control with the G2 real-time expert system," Proceedings of the 1st international conference on Industrial and engineering applications of artificial intelligence and expert systems, vol. 1, pp. 492-497, 1988.

[6] J. Giarratano and G. Riley, Expert systems: principles and programming, 4th ed. Thomson, Boston, 2005.

[7] V.N. Kuchuganov and D.R. Kasimov, "An Environment of Programming of Strategies on the Base of Knowledge," JCKBSE 2014, CCIS, vol. 466, pp. 726-734, 2014.

[8] V.N. Kuchuganov, "Elements of associative semantic theory," Large-Scale Systems Control, vol. 40, pp. 30-48, 2012. (In Russian).

[9] V.M. Kiryuhin and S.M. Okulov, Methods of solving problems in International Olympiads in Informatics. BINOM LZ, Moscow, 2007. (In Russian).