INSTITUTE FOR INFORMATICS AND AUTOMATION
PROBLEMS OF NAS RA

Sedrak V. Grigoryan

# RESEARCH AND DEVELOPMENT OF ALGORITHMS AND PROGRAMS OF KNOWLEDGE ACQUISITION AND THEIR EFFECTIVE APPLICATION TO RESISTANCE PROBLEMS

Thesis

for obtaining candidate degree in technical sciences in specialty 05.13.05 —" Mathematical modeling, numerical methods and program complexes"

Scientific adviser: Dr. of Phys. and Math. Sc. Prof. Edward M. Pogossian

Yerevan-2016

# Contents

# INTRODUCTION

**Actuality of the Research**

1. We consider a class of combinatorial problems defined as problems where spaces of solutions are Reproducible Game Trees (RGT) [1], [2].

RGT class includes important problems like computer networks intrusion protection, optimal management and marketing strategy elaboration in competitive environments, defense of military units from a variety types of attacks, communication problems , certain types of teaching, chess and chess-like games [3] [4], [5], [6], [7] [8].

Expert approaches in solving RGT problems, at present, stay the most effective. Thus, the question rises whether it is possible to construct knowledge-based solvers of RGT problems comparable with the experts by their effectiveness. In other words, whether it is possible to construct solvers adequately modeling experts in regards to RGT problems.

2. In [9] [10] it was proven that RGT problems are reducible to each other, particularly, to some standard kernel RGT problem K, say, chess.

Thus, we get an opportunity to integrate the best known achievements in solving particular RGT problems into RGT Solvers letting us to apply those achievements to any of RGT problem.

Following to that idea, our work is based on, certain achievements of knowledge-based solvers in chess [15], [1], [18], intensively studied since Shannon's pioneer work in 1949 [11], and on our findings in intrusion protection [4], [6] .

In chess algorithms of Shannon and his followers, as well as in the algorithms of present day chess engines [12] [13], the essence of chess knowledge is mapped into parameters while the combinatorial nature of situations in RGT problems and their varieties as it was discussed by Botvinik in [14], in principle, cannot be adequately averaged by parameters.

Thus, in RGT solutions, we follow the research lines of Botvinnik [14], Pitrat [33], Wilkins [34] and SOAR project [35] and ones successfully started since 1957 in the Institute for Informatics and Automation Problems at the Academy of Sciences of Armenia and based on modeling of expert approaches [15], [1] involving: knowledge bases, knowledge-based algorithms of decision making and matching situations to classifiers, as well as algorithms of revealing and modifying knowledge.

3. In frame of that approach about 300 units of chess expert knowledge were revealed [15] [16]. The research had shown, that those units consist of strategies, their constituents and descriptions.

Then, the correspondence revealed between those units of knowledge, the winning classes of chess positions described by Zermelo and strategies of game trees argues for a constructive nature of the content of those units of knowledge allowing, in principle, to simulate them.

At the same time, it follows that any real implementation of the contents of those units, in principle, can be only approximations of the original winning game tree structures due to the prohibitive complexity of computations required to prove correctness of the vast majority models of the contents. Simultaneously, we get a precedent of a specification of human expertise presenting essentially personalized approximations of the ideal contents.

For measuring the proximity of chess programs to master ones Reti and Nodareshvili chess etudes were suggested by Botvinik [14] [17] in 1979 as examples of problems requiring extremely high resources for their exhaustive search solving. Particularly, Nodareshvili etude requires search of the depth of 36 of the chess game tree and while parametric search computers analyze billions of positions to solve it chess masters achieve the same goal analyzing only about 500 position.

Following the ideas of Botvinnik strategy search algorithms and programs (IGAF1 and IGAF2) [6] [4] were developed based on common knowledge planning. Even involving restricted types of expert knowledge, those Intrusion Protection Solvers overcome the productivity of minimax algorithm for the depth of search 5 in 14%, using for that 6 times less computing time and searching 27 times less nodes of the tree.

In 2007 Reti and Nodareshvili edutes were solved [18] with usage of software package that implements PPIT (Personalized Planning and Integrated Testing) algorithms. The experiments proved that the program shells, in principle, can acquire multilevel knowledge of experts provided with their help and effectively use them for solving suggested test chess etudes.

Thus, the question rises in involvement and usage of advanced RGT expert knowledge regularly, in systematic ways by appropriate interfaces.

4. With tremendous advances in computers personalized interactive tutoring of students and testing their advances in learning new knowledge becomes possible.

Particularly, while knowledge acquisition by children with ordinary abilities follows to traditions, personal experiment of parents and well known standard methodologies development of unordinary children with positive and negative declines need extremely personalized to each child means.

In [81] were considered the peculiarities and patterns of positive tutoring of autistic children (autistics) in knowledge acquisition and it was found that tools providing the complete tree of

constituents required to acquire certain communalized knowledge followed by providing recommendations on learning the missed constituents can be supportive for any children.

Since testing can be interpreted as an RGT problem, RGT Solvers can be used as advanced instruments to acquire expert knowledge and elaborate effective strategies of its testing while the adequacy of computer based models of explanation of new knowledge and its testing to expert ones have to be substantiated.

5. RGT Solvers 2009-2013 [19] [20] [8] [21] [22] [23] provide the abilities of presentation of RGT problems by specifications as well as provision of RGT expert knowledge for running PPIT algorithms.

Relying on their advances, in this work we aim to answer to some open questions of RGT Solvers.

Namely, we present a model of an advanced RGT expert knowledge presentation and provide an experimental evidence of its adequacy to one of experts.

We suggest RGT strategy search algorithms able to regularly acquire both common and personalized expert knowledge and to its effective usage in RGT solving.

Finally, we describe how our models of RGT knowledge can be processed and how strategy search algorithms can be used in interactive personalized tutoring.

**Objectives of the work are:**

- Provide models and programs for RGT knowledge presentation and algorithms matching situation to them in RGT Solver16.
- Prove the adequacy of developed knowledge presentation models and matching algorithms for various RGT problems.
- Implement strategy searching Personalized Planning and Integrated Testing (PPIT) algorithms for RGT problems based on processing of plans and goals.
- Ensure the adequacy of the strategy search algorithms for RGT class.
- Provide RGT Solver based tools and develop models for personalized interactive tutoring adequate to tutoring by RGT experts.

**Objects of the research**

Objects of the research are RGT Solver, PPIT strategy search algorithms, the software framework for structuring plans goals, their stores, algorithms developed for selecting the best plan from the given set and searching strategies by the given plan, as well as algorithms and methods for tutoring to RGT problems.

**Methods of research**

Methods of research are based on the theory and methods of planning, strategy searching programs development, programming languages, Java, as well as the principles of knowledge-based systems and tutoring approaches.

**Scientific novelty**

1. OOP models for presentation of RGT expert knowledge based on "have-, be-, do-" categories of languages are developed and algorithms for matching situations to those models are realized.
2. Proof of the adequacy of presentation and matching programs is provided based on experiments with various RGT problems.
3. Strategy search algorithms based on processing of RGT plans and goals are developed and embedded into RGT Solver.
4. Experiments proving the adequacy of developed PPIT strategy search algorithms for RGT problems, particularly, for Botvinnik's Reti test etude, are provided.
5. RGT Solver based tool and methods for personalized interactive tutoring modeling ones of chess master are developed.

**Practical significance**

1. RGT class includes a large group of practical problems (defense and attack of military units, computer networks intrusion protection, etc.). The developed models of expert knowledge presentation, as well as matching and strategy searching algorithms, can be used for decision making in any problems of RGT class.
2. The RGT Solver based tool and methods for personalized interactive tutoring provide constructive research environment for developing computer based RGT tutoring, particularly, for chess.

**Practical implementation**

1. RGT Solver has been installed in Institute for Informatics and Automation Problems of NAS RA for educational and research purposes.
2. Developed "RGT Solver" package has been installed in "FimeTech" company, where it is used in a program that, first, finds chess boards in images taken from mobile camera, then sends them to RGT Solver package in FEN (Forsyth–Edwards Notation) format. RGT Solver reveals the characteristics/descriptors of the position at the board, processes PPIT algorithm and recommends moves in the position the best relatively its ad hoc expert knowledge.

**The following topics are presented to the defense**

- Structures and models for presentation of RGT expert knowledge including goals and plans, means for enriching knowledge, algorithms for matching situations to them as well as experiments substantiating the adequacy of the models and algorithms.
- Personalized planning and integrated testing strategy search algorithms based on goals and plans as well as the experiments proving their adequacy.
- Personalized interactive tutoring methods and environment modeling ones in expert tutoring.

**Approbation**

The results of the dissertation have been presented at:

- The Annual Conference of SEUA, Yerevan, 2012.
- International conference - Computer Science and Information Technologies (CSIT) (Yerevan, Armenia) in 2013;
- International conference - Computer Science and Information Technologies (CSIT) (Yerevan, Armenia) in 2015;
- Seminar at Institute for Informatics and Automation Problems of NAS RA.

**Publications**

The main topics of the dissertation are published in [24] [25] [26] [27] [28] [29].

# CHAPTER 1. STATE OF THE ART

Unsolved combinatorial problems represent wide range of important problems. Problem solving approaches in this class of problems are the following: a) finding linkages between solved class of problems (differential equations, statistical theory, etc.) and unsolved ones to present projections of solved problem in solving unsolved problems; b) human approaches in problem solving. The latter appears to be the only effective approach in solving this type of problems and new problems. Human approach includes decision making with effective usage of problem domain knowledge.

Presentation of human-like problem solving approaches is an important research area of AI [17] [30] [31] [32] [33] [34] [35] [36].

The general overview of the problem of understanding of the of knowledge models is discussed in [37] [38] which stands as one of main problems in human-like problem solving modeling.

The aim of the work is to develop models and programs for presentation of concepts of human vocabulary of lexical units of RGT problems, provide enhancements in existed models and matching algorithms of situations to the problem knowledge, as well as develop algorithms for knowledge based strategy search in these problems with the experiments to prove the adequacy of developed strategy search algorithms. The work also aims to provide applicable personalized interactive tutoring methods and models similar to expert tutoring, particularly for chess.

## 1.1 OPTIMAL STRATEGY PROVISION PROBLEMS

In the continuous researches of our team a class of problems is defined as a class of unsolved combinatorial problems [39] [9]. The class named RGT is a sub class of Optimal Strategty Provision (OSP) problems. RGT problems meet the following requirements:

- there are (a) interacting actors ( players, competitors, etc.) performing (b) identified types of actions in the (d) specified types of situations;
- there are identified utilities, goals for each actor;
- actions for each actor are defined.

Actors perform their actions in specified periods of times and do affect situations by actions in time *t* by transforming them to new situations in time *t+1* trying to achieve the best utilities on that situations (goals) by regularities defining these actions.

Starting from any given situation $S_0$ and applying regularities of the actions successively we can generate a game tree representing all possible situations in interactions between players. Thus the game tree represents all the possible policies of players, games and *strategies*.

Strategies are algorithms which define certain actions for players to perform in the given situations to affect those situations considering also any actions performed by the opponent actors. Best strategies are the ones which lead to max utilities.

The quality of strategies is considered by certain criteria based on evaluation of utilities existing on the situations in the game trees. For the given criteria of strategies at time t, starting from a situation $S_0$ and for the actor $A$, who is going to act in $S_0$, we can generate the *game tree* $GT(S_0, A)$ for each strategy with the root at $S_0$, order strategies by expected utilities at *t* and chose the best of them to apply in $S_0$.

We consider games to be finite and they are being finished if any of goal situations defined in the specification of the problem appears.

A *strategy* $GT(S_0, A)$ for player A according to a deterministic program represents all possible *games* of the strategies starting from the $S_0$ situation. In that sense the $GT(S_0, A)$ determines the space of all possible solutions from the $S_0$ situation.

In this case strategies are described by the consecutive lists of actions by each actor, but not detailed commands.

The quality of strategies defined with the criterion K by the evaluation of situations allows to define the best strategy $S^*(S_0, A)$ and reveal the corresponding best action $A$ for that $S_0$ situation.

For example chess interpretation as an RGT problem is the following: the actors are white and black players with checkmate identified as the goal for each side, chess piece moves are actions and compositions of chess pieces on the board specify specific game situations and the chess game tree is reproducible, where each situation of the board is a node and each edge is a move.

However the chess situations are discrete and situations are finite, we also consider other competition problems by approximation situations and discretization of time intervals.

*In Optimal Strategy Search (OSP) problems* with the given quality criterion $K$ of strategies the best action with respect to $K$ criterion for any given situation of the problem is required.

Wide range of important problems are Optimal Strategy Provision problems. Particularly, Network Intrusion Protection, Management and Marketing in oligopoly competitions, Chess and Chess-like combinatorial problems, Anomalies detection and correction in computing, Single Ownship Defense from Air Threats, Computer Terrorism Countermeasures, Disaster Forecast and Prevention, Information Security, etc., announced by the NATO [40] as well as problems in [7] [41] may be reduced RGT class of problems.

*The RGT class* comprises the OSP problems where the OSP requirement to have descriptions of situations after transforming them to actions is replaced by the following stronger requirement:

- The situations, where the actors act in and transformed after actions are applied on those situations, can be *adequately simulated.*

Thus, for RGT problems the game trees can be constructively simulated which allows creating a common methodology and computer based strategy search algorithms to find optimal solutions for corresponding problems.

In [4], [42] [43], [9], it was proved that chess and chess like combinatorial problems, intrusion protection and oligopoly competition are RGT problems and developed common strategy search algorithms and methods can be applied to find high quality solutions for them.

In the following section we will present the state-of-the-art in computer game playing. The complete overview is presented in [44].

Search based strategy construction algorithms are discussed followed by consideration of knowledge-based algorithms and programs for RGT problems.

### 1.1.1 SEARCH BASE ALGORITHMS

The search based strategy construction algorithms use the game tree of the problems for searching the strategies. The strategies are searched to lead to the goal situations of the game. In the game tree each node represents a situation and each edge is an action transforming situations. Thus the strategy is a sequence of actions to be performed by the actor to reach to the target goal situations.

The famous search based algorithms and their modifications are considered below.

#### 1.1.1.1 MINIMAX

**Minimax** is a decision rule used in decision theory, game theory, statistics and philosophy for *mini*mizing the possible loss for a worst case (*max*imum loss) scenario. The algorithm was originally formulated for two-player games. It covered the cases where players perform alternate

actions and the cases where the players perform actions simultaneously. In the future it was expanded to represent decision making in more complex games and in the presence of uncertainty in the game.

The algorithm is recursive. It chooses the next move in an n-player game. A value is assigned to each position or state of the game. *Position evaluation function* performs computation of this value and shows the utilities available for a player in the given position. The player then makes the move that maximizes the minimum value of the situation revealed from the opponent's possible moves. If it is A's turn to move, A gives a value to each of his legal moves.

A possible evaluation method can be represented as assigning +1 for the win of A and -1 for the win of B. This presentation of evaluation method in the following way leads to the combinatorial game theory as developed by John Horton Conway.

An alternative approach is to make the evaluation calculation by assigning positive infinity to the situations where a move is leading to an immediate win for A and negative infinity for the situations where move is leading to an immediate win for B. Any of other actions performed by A is the calculated by selecting the minimum value from the each of B actors possible action values and smiliarly actions for B are calculated by selecting the max value from each of action performed by A. Thus, A is called the maximizing player and B is called the minimizing player, hence forming the name of the strategy search algorithm **minimax**. The described algorithm of minimax assigns value of positive or negative infinity to any situation which represents any of final positions be that winning or losing accordingly.

*Negamax* version of the minimax algorithm is a version of minimax that simplifies the representation of situation evaluations. It relies on the basic idea that $\max(a,b) = -\min(-a, -b)$. The benefit of the following presentation is that it avoids evaluating different players with the separate functions.

The below given pseudo code represents the Negamax version of minimax algorithm.

```
function integer minimax(node, depth)
    if node is a terminal node or depth == 0:
        return the heuristic value of node
    α = -∞
    for child in node:                    # evaluation is identical for both players
        α = max(α, -minimax(child, depth-1))
    return α
```

In the below given minimax algorithm example the algorithm execution in the game tree is demonstrated, where circles represent the actions performed by maximizing player and squares represent the actions performed by the minimizing player. Maximizing player performs the first action.

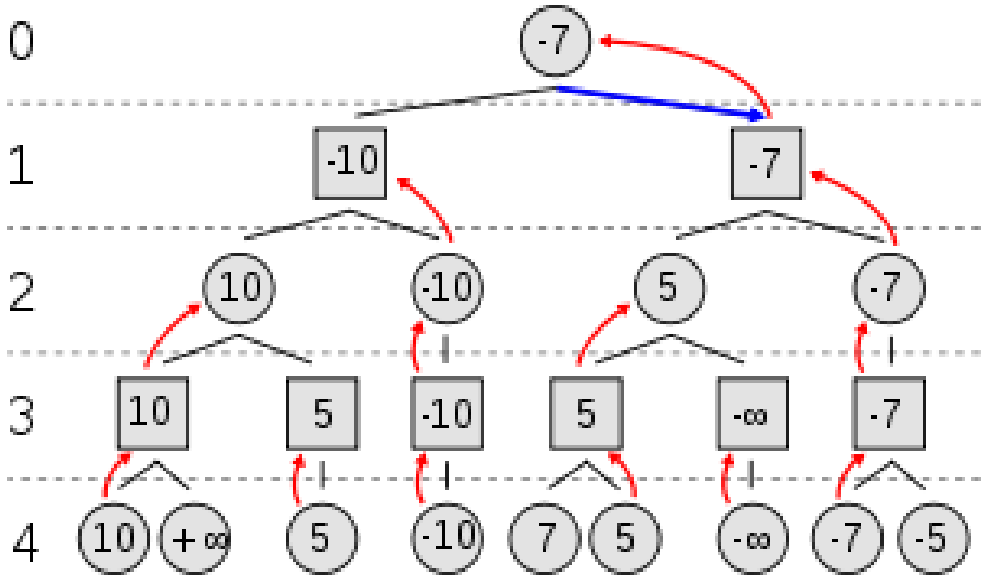



Fig. 1. Minimax algorithm. Circles represent the actions of maximizing actor and squares represent the actions performed by the minimizing actor. The values inside the circles and squares represent the value α of the minimax algorithm. The red arrows represent the chosen actions and the blue arrow the chosen move.

Because of the limitation of computation resources, the generated game tree is limited and 4 depth tree is presented.

The algorithm evaluates each leaf node of the tree using an evaluation function and obtains certain value as it is shown in the figure. The actions which lead to winning situations for the maximizing player are assigned with positive infinity, while the actions leading to winning situations for the minimizing player negative infinity are assigned. At the next level (level 3) the algorithm choses the smallest of its child node values and assigns to the node which it represents (e.g. the node on the left for the 3$^{rd}$ level selects the minimum value from "10" and "+∞", therefore assigning the value "10" to its node). On the 2$^{nd}$ level of the treethe algorithm selects the greatest of the values of its child nodes and assigns to its own node (e.g. for the 2$^{nd}$ level left node the algorithm selects between “10” and “5” and assigns 10 to its node). The algorithm processing is continued until reaching to the root. The selected action from the root node is the action to perfrom according to the minimax algorithm.

### 1.1.1.2 ALPHA-BETA PRUNING

*Alpha-beta pruning* is a search algorithm and improvement in minimax algorithm which aims to reduce the number of nodes in the game tree by cutting hopeless games from it. When an action exists which leads to a situation where the situation is proved to be better that the action which is being processed currently, the algorithm leaves the current action. The basic idea is that such games which are not required to be used in the further processing of the game tree and for cannot be selected as actions to perform.

The algorithm uses two values, alpha and beta, which represent the minimum value that the maximizing player is convinced to have and the maximum value that the minimizing player, is convinced to have respectively.

Initially negative infinity is assigned to alpha and positive infinity is assigned to beta. The difference between alpha and beta becomes smaller during the recursive processing. When the value of beta becomes less than the value of alpha, it means that the current position cannot produce result of best play by both players and hence that game is being cut from the tree.

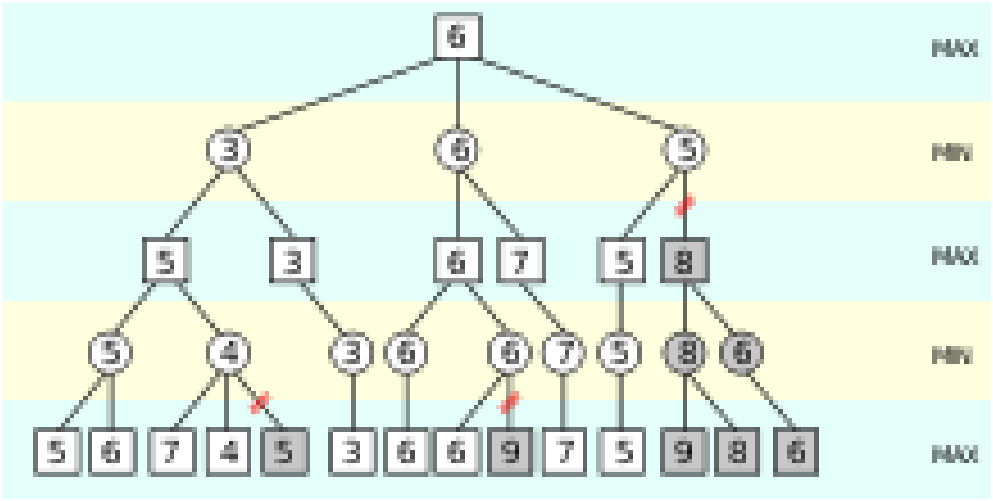



Fig. 2. Alpha-Beta pruning example**.**

The pseudo code of alpha beta algorithm is below:

```
function alphabeta(node, depth, α, β)
    (* β represents previous player best choice - doesn't want it if α would worsen it *)
    if  depth = 0 or node is a terminal node
        return the heuristic value of node
    for each child of node
        α := max(α, -alphabeta(child, depth-1, -β, -α))
```

```
        (* use symmetry, -β becomes subsequently pruned α *)
        if β≤α
          break                    (* Beta cut-off *)
      return α

    (* Initial call *)
    alphabeta(origin, depth, -infinity, +infinity)
```

Alpha-beta pruning is an effective and sound optimization of minimax algorithms since it does not make any effect on the produced result of the processed algorithm which is being optimized.

Branches of game tree can be cut off when processing alpha-beta pruning algorithm. This makes benefit in processing time providing more resources for searching strategies in more promising subtrees and making the depth of the game tree value greater. This optimization reduces the effective depth to slightly more than half that of simple minimax if the nodes are evaluated in an optimal or near optimal order (the best choice for side on move ordered first at each node).

#### 1.1.1.3 LEARNING SEARCH CONTROL

There are different ways of making the search more efficient in search-based game playing programs, particularly the following approaches are famous: search depth parameter tuning, quiescence search parameters usage, heuristic methods for ordering actions and restricting the list of actions by some criteria revealed for each situation, extensions, etc.

Donninger [45] considers optimizations of search parameters in an automatic ways for his NIMZO chess program, rather than tuning the parameters for NIMZO's evaluation function.

There have been many attempts to increase the efficiency of search algorithms in other areas of Artificial Intelligence [46] [32], however it still remains in widely used in the approaches of game playing search-based programs.

Buro [47] introduces the PROBCUT Selective search approaches were suggested in various works, particularly in [47] PROBCUT is suggested, where basic idea is to use the evaluations obtained by a shallow search to estimate the values that will be obtained by a deeper search, the relation between these estimates is computed by means of linear regression from the results of a large number of deep searches. During the play, branches that seem unlikely to produce a result within the current alpha-beta search window are pruned. In the subsequent work, Buro [48] generalized these results to allow pruning at multiple levels and with multiple significance thresholds. A similar approach was previously suggested in [49] [50].

In [51] Null Move pruning is discussed which is a method to reduce the search space by trying a "null" or "passing" move, then seeing if the score of the subtree search is still high enough to cause a beta cutoff.

Extensions are certain actions to perform and extend depth of search for certain moves and try to find better moves faster. To extend a move, its search depth is extended by a certain value. In [52] it is stated that one of the important challenges in the extensions making is the categories of moves. The approach suggests empirical data which allows making good precisions.

#### 1.1.1.4 EVALUATION FUNCTION TUNING

Automatic adjust of weights of evaluation functions is one of the most extensively studied topics learning problems. The basic idea is that the game programmer needs to provide with the important properties to calculate for the evaluation of the situations, thus providing parametric knowledge-based search. In [11] studies for chess by Shannon revealed the following important properties of chess situations: number of pieces of each player on the board, pawns structure, king's safety, etc.

The known information about the situation and parameters is the way the pieces of knowledge can be combined. The question is how to measure the importance of properties and provide relative weights. There are several approaches to solving the problem. Let's discuss them by types.

In supervised learning the evaluation function is being trained by the information given about the situations and the correct values representing those situations. This means learning programs are trained by the examples of positions and actions with the corresponding correct evaluation values.

The next category of evaluation function measurement is comparison training, where collection of action pairs is provided, where each of pair contains the information about preferable one from the given pair. An alternate version can present collection of actions and corresponding situations, from which those actions are performed.

In reinforcement learning, the program works based on feedbacks. It does not receive any direct information about situations and actions, and is being provided with the feedback from the environment detailing about the resulted situations at the end of each action or situation. The feedback thus provides the feedback of actions being good or bad, particularly it can indicate if that was a winning or losing move. Temporal-difference learning is a special case of reinforcement learning which can use evaluation function values of later positions to reinforce or correct decisions earlier in the game.

In chess, particularly, a problem can appear that the position of the root node might have different properties and characteristics from a node in the tree reproduced from the root. The evaluation of similar cases can cause the following issue: the situation is in the middle of a figure trade, say queen trade. The current situation might be evaluated with the value defining that the position “one queen behind”, while a little bit of extension of search in the game tree will show that the board position “one queen behind” but rather equal because the queen can be recaptured in next moves. The application of tuning of the evaluation function on that situation would give the evaluation indicating the situation as equal; however this is not correct in these cases, because these tactical patterns shall be handled by the search and not by the evaluation function. So the evaluation should be applied only on the situations representing leaf nodes of the game tree, which values can be back-propagated to the root node situation.

### 1.1.2

For RGT problems with given arbitrary situation x and actor A, who is going to act in x, we can generate corresponding game tree GT(x, A) comprising all games started from x. Chess and chess like combinatorial problems, intrusion protection of computer networks, defense of navy from air threats, management and marketing problems are urgent RGT problems. Common methodology developed for class of RGT problems can be effectively applied to each of problems to find high quality solutions.

The search-based algorithms use the game tree for searching strategies that guide to the goal situations of the problems starting from the given situations. Minimax and Alpha-beta pruning are famous search-based algorithms.

RGT problem game trees are enormous size thus making impossible search-based algorithms application on searching for solutions for RGT problems, thus more improvements and enhancements are applied on such problem solving algorithms, such as learning search control, tunin of evaluation function.

## 1.2 RGT SOLVERS AND PPIT ALGORITHM

Human-like problem solving approaches are applied for RGT problem solution searching. Integration of human-like problem solving approaches for combinatorial problems are important because of effectiveness of the given solutions comparing to the approach not using human knowledge, as well as it enhances human-computer interaction bringing it to the level of human interaction. Thus we do research in developing expert knowledge-based RGT solvers.

In the following section we discuss developed RGT solver programs and algorithms, as well as give a brief overview of their shortcomings.

In [17] [14] Botvinnik studied knowledge-based approach, highlighting shortcomings of parametric interpretations of knowledge in decision making in chess. Human-like knowledge based strategy search approach was suggested, as well as Reti and Nodareshvili etudes were suggested for such programs to test the quality, however the programs were not completely realized by Botvinnik. His PIONEER program based on the suggested algorithms contained a generalized method of decision-making that, with a few adjustments, enabled it to plan maintenance of power stations all over the USSR.

Human-centirc knowledge acquisition procedures are also discussed in [53] [54] [55] and other knowledge acquisition tools such as [56] [57]. KADS (KBS Analysis and Design Structured Methodology is among the best known methodologies [58]).

The field of automatic knowledge extraction from data has been studied from initial machine learning approaches like [59], semi-automated knowledge engineering [54], to more robust automated approaches like neural networks [60], symbolic rule learning [61], and most recently to proven, practical methods [62], successful applications [63] and mature theoretical frameworks [64]. This basically appears to be complementing to the more traditional human-centered knowledge acquisition approaches. The approaches enable programs creating new knowledge units by revealing from the situations and from owned knowledge pieces, as well as updating existing knowledge, improving their performance. This all is achieved without invention and reprogramming the programs.

There are different famous approaches in knowledge-based systems.

In the production systems "*if* precondition, *then* some action" scheme is used in knowledge presentation and decision providing. SOAR [35] is a famous cognitive architecture, which is a modification of production systems, where rules of if/then form appear in a prioritized order.

Object Oriented languages are another way of presentation of knowledge in these kind of systems. Here the concepts are defined in prototypes, where slots are used. Slots represent attributes of a concept and their values. The hierarchy of classes and instances is used to organize the structure of concepts.

Another approach of representation of knowledge and relations between concepts is Ontology. Ontology mappings allow translations from one domain into another. Ontology based works demonstrate the power of human augmentation of automatically-acquired, partial knowledge, which

helps to complete our integrated vision of an end-to-end system for data to knowledge for reasoning, where full automation is not feasible.

Ongoing researches in understanding natural language are provided in the UNL project [65] UNL, in contrast to approaches which present natural languages, represents information conveyed

| Property | Prod. Sys. | Protégé | OOP |
|---|---|---|---|
| Transparency | - | + | + |
| Reuse | + | + | + |
| Inheritance | - | - | + |
| Polymorphism | - | + | + |
| **Describe static entities (for example goals)** | - | + | + |
| **Define dynamic actions to build the algorithms** | + | - | + |
| **Dynamically generate and integrate new meanings** | + | + | - |
| Dynamically modify existing meanings | + | + | - |
| Change hierarchies of existing meanings | - | - | - |
| **Match situations to meanings** | + | - | - |

Fig. 3. Analysis of the knowledge-based systems, where Production Systems, Protégé - the ontology editor and knowledge acquisition system and OOP languages (C++) are shown. '+' indicates the existence and '-' the absence of the property.

by natural languages, having as a goal idea to present “what was meant”. It also differs from auxiliary languages such as Esperanto with no intends to be human language.

The knowledge-based systems shortcomings are shown in Fig. 3. Overall the mentioned approaches and their shortcomings are discussed in [23].

SDL language discussed in [66] describes knowledge in forms of certain actions to perform in certain situations similar to production systems.

In general planning systems such as STRIPS (Stanford Research Institute Problem Solver) use composition of states, goals and actions to perform. Actions consist of preconditions and postconditions. The following planning approaches are not specific to a problem, however the restrictions common description of states and presentation of plans as a chain of actions restrict their general application to situations where there strategies compose more complication chains, and it is preferable to define priorities of goals to achieve instead of defining chains.

Problems of knowledge-based strategy search algorithms are successfully studied in IIAP of NAS RA. In [65] knowledge-based chess endgames solver is suggested. The problem is studied in

the laboratory of Cognitive Algorithms and Models (E. Pogossian et al.). Below we will give the overview of developed knowledge-based RGT Solvers.

### 1.2.1 COMMON KNOWLEDGE BASED RGT SOLVERS

We search for expert knowledge-based strategies for RGT problems. The researches in the RGT class of problem appear in [65] , as well as in [1] [42].

An interpretation of knowledge based strategy searching algorithms for RGT problems called Intermediate Goals at First (IGAF) [4] were developed and implemented since 2003. The algorithms were using common knowledge planning and dynamic testing of plans in the corresponding game trees. Viability of the approach was demonstrated for the problem of intrusion protection of computer networks (representative of RGT class). For example, for the IP problem it was outperforming system administrators and known standard protection systems in about 60% in experiments on fighting against 12 different types of known network attacks.

To increase the efficiency of the IGAF1 algorithm its more advanced version IGAF2 able to acquire a range of expert knowledge in form of goals or rules and to increase the efficiency of strategy formation with increasing the amount of expert knowledge available to the algorithm was suggested [6].

Using IGAF1 game tree model with a variety of algorithms to counteract to intrusions was experimented. IGAF1 is similar to Botvinnik's [14] chess tree cutting-down algorithm, which is a knowledge-based strategy search presentation. The last is based on the natural hierarchies of goals in control problems and the assertion that search algorithms become more efficient if try to achieve subordinate goals before fighting for the main ones. The trajectories of confronting parties to those subgoals are chained in order to construct around them the zones of the most likelihood actions and counteractions.

As the result of comparative experiments with the minmax and IGAF1 algorithms the following statements were revealed:

- the model, which is using the minimax algorithm, is compatible with experts (the system administrators or specialized programs) against intrusions or other forms of perturbations of the base system
- the IGAF1 cutting-down tree algorithm along with being compatible with the minimax one can work enough efficient to be used for real IP problems.

A more advanced version 2 of the algorithm – IGAF2 then was suggested which is able:

- to acquire a range of expert knowledge in form of goals or rules

- to increase the efficiency of strategy formation with increasing the amount of expert knowledge available to the algorithm.

The algorithm relies on the concepts of "trajectory of an attack" and "zone of counteraction".

The trajectory of an attack is a subtree Ga(S', P'), where S' is a subset of the system states S' $\subseteq$ S and P' is a subset of the actions, consisted of an offensive's conversion procedures Pa and a defender's normal conversion procedures Pdn, i.e. P'=Pa$\cup$Pdn, P'$\subseteq$P, P' $\neq \varnothing$.

The zone of counteraction is a subtree Gz(S", P") built around the graph of the trajectory of an attack Ga(S', P'), i.e. Ga $\subseteq$ Gz, where S" is a subset of the system states S" $\subseteq$ S, which belong to the trajectory of an attack, hence S' $\subseteq$ S", and P" is a subset of the actions, which consist of the conversion procedures, defined on the trajectory of an attack, P' and the defender's special conversion procedures Pds, i.e. P"= P'$\cup$ Pds, P"$\subseteq$P, P" $\neq \varnothing$, hence P'$\subseteq$ P".

The following expert goals and rules had been embedded into the IGAF2 algorithm:

The goals:

1. the critical vs. normal states are determined by a range of values of the states of the system; for example, any state of the system with a value of corresponding criterion function, that is more or equal to some threshold, may be determined as a critical goal
2. the suspicious vs. normal resources are determined by a range of states of the classificators of the resources; combinations of values of the classificators identified as suspicious or normal induce signals for appropriate actions.

The rules:

1. Identify the suspicious resources by the classifiers and narrow the search to corresponding game subtree
2. Avoid critical states and tend to the normal ones
3. Normalize the state of the system. First, try such actions of the defender that influence on the resources caused current change of its state and if they don't help try other ones
4. In building game subtree for suspicious resources use
    a. defending actions able to influence on such resources
    b. use normal actions until there is no critical states
    c. if some defensive actions were used on previous steps decrease their usage priority
5. Balance the parameters of resources by keeping them in the given ranges of permitted changes.

The IGAF2 algorithm consisted of following instructions:

1. Standard min max technique with alpha-beta pruning based on the range of critical/normal state values introduced as the goal 1 is used. Current node is created and the value of its local state calculated. If the node is terminal, the local state value is compared with sibling nodes, and their max (either min) value is sent to the parent node. Program was realized in C++.

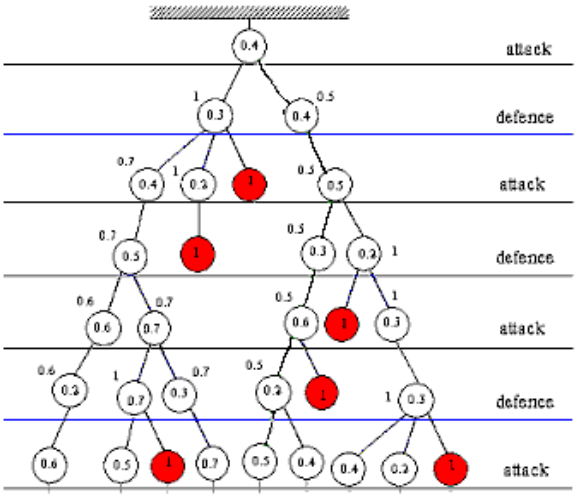



Fig. 4. The game subtree

2. Determine all suspicious resources.
3. Build the game subtree for suspicious resources starting from the root state of the tree and using the 4th group of rules determine the trajectories of attacks (Fig. 4).

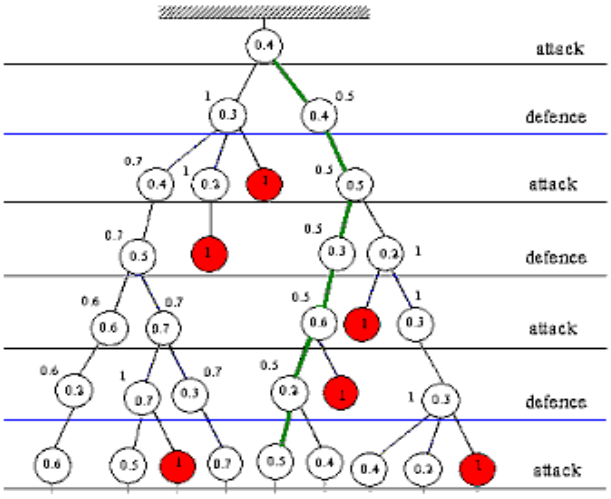



Fig. 5. The best minmax action

4. Calculate the values of the terminal states of the tree, find the values of others by minmax procedure and determine the best minmax action from the root state (the green branch on the Fig. 5).

5. Determine the trajectories of attacks induced by the best action from the root of the tree to its critical states and consider them as targets (Fig 6).

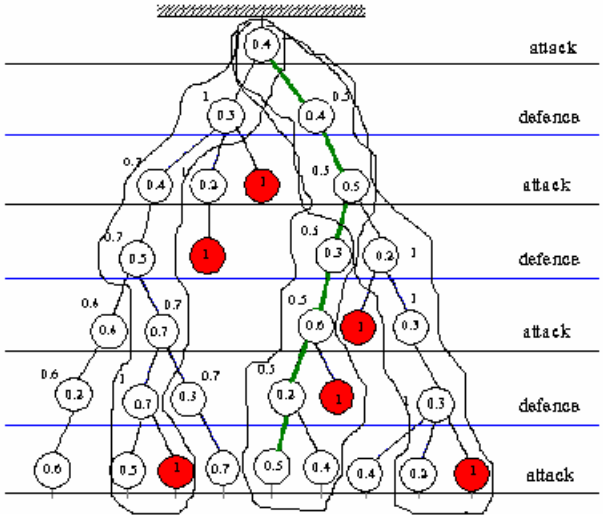



Fig. 6. The trajectories of attacks

1. Build the zones of counteractions for the target trajectories using the 4th group of rules and rule 5th, then calculate the values of the states of the corresponding subtree using the minmax (Fig.7).

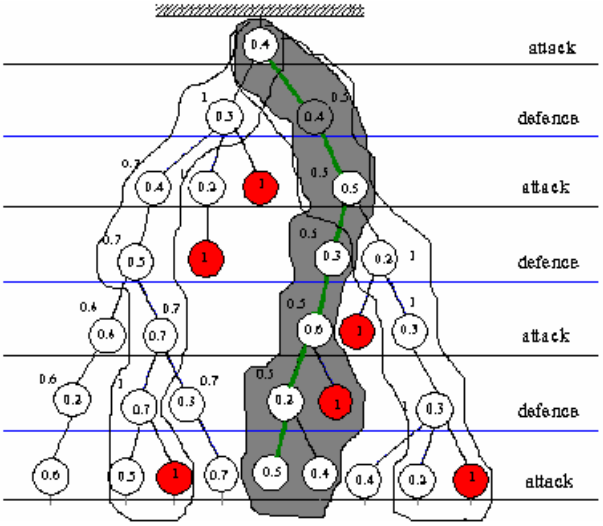



Fig. 7. The zones of counteractions

2. Choose the defender's action from the root as the one leading to the state with min value, i.e. to the most stable state estimated by the minmax.
3. End the defense analysis and wait for the attacker's actions.

The effectiveness of the IGAF2 algorithms was successfully tested for the network intrusion protection problems against representatives of four classes of attacks: SYN-Flood, Fraggle, Smurf and Login-bomb, which allowed formulating, in particular, the following statements:

- The number of nodes searched by the IGAF2 algorithm for making decisions with all expert rules and subgoals is the smallest compared with the IGAF1 algorithm or with the minimax algorithm in which the depth of the search is increasing up to 13.

The recommended version of the IGAF2 algorithm with all expert rules and subgoals, for the depth of search 5 and 200 defending steps, outperforms the productivity of the minmax algorithm by 14% while uses 6 times less computing time and searches 27 times less nodes of the tree.

### 1.2.2 PERSONALIZED KNOWLEDGE BASED RGT SOLVERS

#### 1.2.2.1 PPIT1 ALGORITHMS

In [18] personalized knowledge-based approach of chess solver is suggested.

Concepts are defined as a kind of knowledge to identify and recognize realities where concepts are considered as goals with elements of intensions or requirements to achieve them. Motives are attributes used to argue the preferences of some goals in analyzed situations. Strategies, plans, goals and rules are defined as expert knowledge to specify compositions of his actions in time. Rules are kind of if x / then y operators to specify a procedure y for the realities that fit to the x requirements. Any strategy may be determined as a composition of rules.

Two sources of strategy expertise were identified represented by UCR [15]:

- concepts and attributes determined by specification of the game tree;
- winning strategies induced by examining the game tree.

The “Mat” positions are the only exclusion where winningness is determined by a few game specification attributes calculated either statically or by one-two plies. It is proved that chess concepts become elements of specifications of winning by Zermelo positions what argues for possibility of their simulation.

Personalized Planning and Integrated Testing (PPIT) programs aimed to acquire strategy expert knowledge to become comparable with a human in solving hard chess problems are described. In fact, the following two tasks of knowledge acquisition can be identified the process:

- construction of shells of the programs allowing to acquire the contents of units of chess vocabulary and
- construction of procedures for regular acquisition of the contents of the units by the shells.

PPIT1 programs appear as a composition of the following basic units:

- Reducing Hopeless Plans (RHP)

- Choosing Plans with Max Utility (CPMU)
- Generating Moves by a Plan (GMP)

The algorithm works as defined in [18].

For the given P1 questioned position and a store of plans, RHP recommends to CPMU a list L1 of plans promising by some not necessary proved reasons to be analyzed in P1. The core of the unit is knowledge in classification of chess positions allowing identifying the niche in the store of knowledge the most relevant for analysis the position. If the store of knowledge is rich and P1 is identified properly it can provide a ready-to-use portion of knowledge to direct further game playing process by GMP unit. Otherwise, RHP, realizing a reduced version of CPMU, identifies L1 and passes the control to CPMU. CPMU recommends to GMP to continue to play by current plan if L1 coincides with list L0 of plans formed in the previous position P0 and changes in P1 are not essential enough to influence on the utility of current plan. If changes in P1 are essential, CPMU analyzes L1 completely to find a plan with max utility and to address it to GMP as a new current plan. Otherwise, CPMU forms new complementary list L1/ L1*L0 from the plans of L1 not analyzed, yet, in L0, finds a plan with the best utility in that list and comparing it with utility of current plan recommends the one of them with a higher utility.

Utilities are calculated by trajectory-zones based technique (TZT) [14] originally suggested to estimate utilities of only captures of the opponent pieces. For example, to choose capture with max utility TZT chains the moves to each piece of the opponent (trajectories) without accounting possible handicaps for real capturing then using all available knowledge “plays the zones” of the game tree induced by the trajectories followed by estimation of their values to choose the best. The utility of units of knowledge the operators assemble from utilities of corresponding arguments in some predetermined ordering. Thus, each operator can provide by a request the arguments which are analyzed at the moment.

For example, realizing current plan the shell can determine the goal in the agenda which in turn determines basic attributes to be considered followed by indication of the arguments of those attributes.

Utility estimation operators rely on the principle of integration of all diversity of units of knowledge the shell possess at the moment. In fact, the operators represent a kind of expert knowledge with a variety of mechanisms and leverages to make them better. Along with dynamically changed parametric values of pieces they can include rules, positions with known values and strategies to realize them, other combinatorial structures. To estimate expected utilities the operators take into account the cost of resources necessary to get them.

In the C++ realization the units of knowledge were realized as OO classes with specialized interfaces for each type of knowledge and one common for the shell itself.

Experiment was done in solving Reti and Nodareishvili etudes (Fig. 8) requiring by Botvinnik [14] [17] intensive expert knowledge based analysis not available to conventional chess programs.

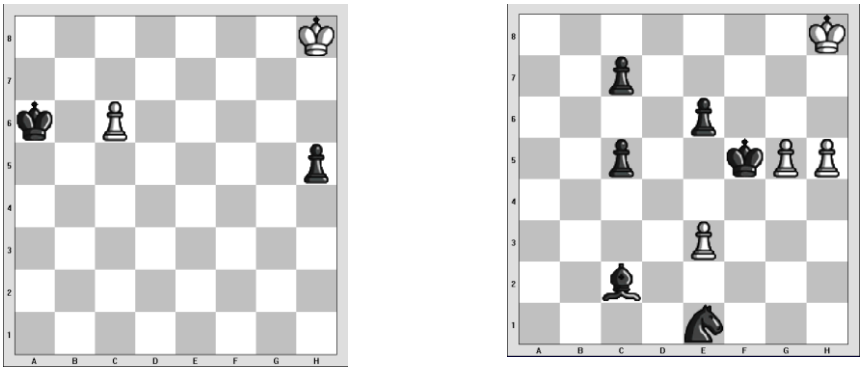


Fig. 8. Reti and Nodareishvili etudes.

Reti etude requires achieving draw and Nodareshvili etude requires achieving win for white.

The plan for Reti etude solution was the following:

*G1. Capture the pawn*

*G2. Move forward the pawn at passant*

*G3.Protect your pawn*

*G4.Be max close to both: your and opponent pawns.*

The plan for Nodareshvili is the following:

*G1.Move forward pawns at passant*

*G2. Protect your pieces*

*G3. Occupy or protect the most important squires*

*G4. Identify infinite check*

*G5. Avoid infinite check*

*G6. Decrease suppression of your king*

*G7. Capture the queen*

*G8. Find opponent pieces on the same line with its king*

*G9. Attack the king*

*G10. Attack the pieces on the line with the king*

*G11. Occupy the last horizontal line*

*G12. Approach king to your queen*

*G13. Be opposite to the pawn closest to opponent king.*

The plans for experiments were presented as follows:

Plan1: Do G1and G6

Plan2: Do G1and if G4 do G5 and G6

Plan3: If G8 do G9 and G10

Plan4: Do G11 and G12 and G13.

The experiments provided in the same research proved that the shell, in principle, was able to acquire the above units of knowledge, to choose corresponding plan for each stage and realize it.

Though the realization proved the adequacy of suggested PPIT algorithms in solving RGT problems, the real implementation was strongly restricted to the defined problems and did not provide environment for regular knowledge acquisition and generic strategy search algorithms for the entire RGT class.

### 1.2.3

Knowledge-based systems such as production systems, ontologies and OOP languages are currently used in various expert systems to represent knowledge, while several shortcomings are revealed in the mentioned models, particularly important restrictions in regular acquisition and their usage for OO languages representation are 1) dynamic change of class hierarchy, 2) matching situations to attributes of classes to find instances of classes in the given situations.

Knowledge-based strategy search algorithms IGAF1, IGAF2 use common knowledge in decision making. The algorithms were successfully developed and experimented for Intrusion Protection problem of RGT class. The shortcomings of the approach were:

1. developed programs did not allow regular acquisition of knowledge for the given problem of RGT class
2. chunk of knowledge used by human in decision making in these problems is personalized.

PPIT1 personazlied planning and integrated testing algorithms and programs were developed and experimented for well-known Reti and Nodareshvili providing correct solutions. The shorcomings revealed for the algorithms were restrictions in regular knowledge acquisition methods.

## 1.3 THE GRAPHICAL LANGUAGE INTERPRETER OF THE SOLVER

The initial implementation of the PPIT algorithm used knowledge units that were hardcoded as C++ language classes. The approach didn't allow adding expert knowledge in a regular way – there wasn't any regular method for formalization and representation of the expert knowledge. The algorithms were strictly for a certain chess problem solution, while we search for solutions for whole space of RGT problems, meantime providing experiments in chess.

To achieve to a regularity of expert knowledge acquisition for RGT Games a graphical language similar to the UML has been developed [66] [19], [20], using which experts have a possibility to formalize and insert contents of knowledge units into the Solver.

The overview complete of developed interface, as well as enhancements provided in the initial implementation are provided in [23].

The components of the Interface are designed for specifying both game rules by the game specification and knowledge pieces related to strategies.

The interface enables acquisition of RGT expert knowledge in forms of patterns (abstracts). Abstracts are used to define classes as well as operations [2], thereby providing a considerable uniformity of the structure of the language. Abstracts are defining problem specific types and composition of other abstracts. The latters are composed of attributes which can be filled with objects that instantiate other abstracts. New abstracts can be built based on existing abstract by the following main instantiations of already existing abstracts: 1) by composition and 2) by inheritance. The language considers single inheritance approach; hence the semantics of inheritance is similar to [67] and can be easily defined.

The research provided in the works [20] [21] [68] lead to the construction of 4 different types

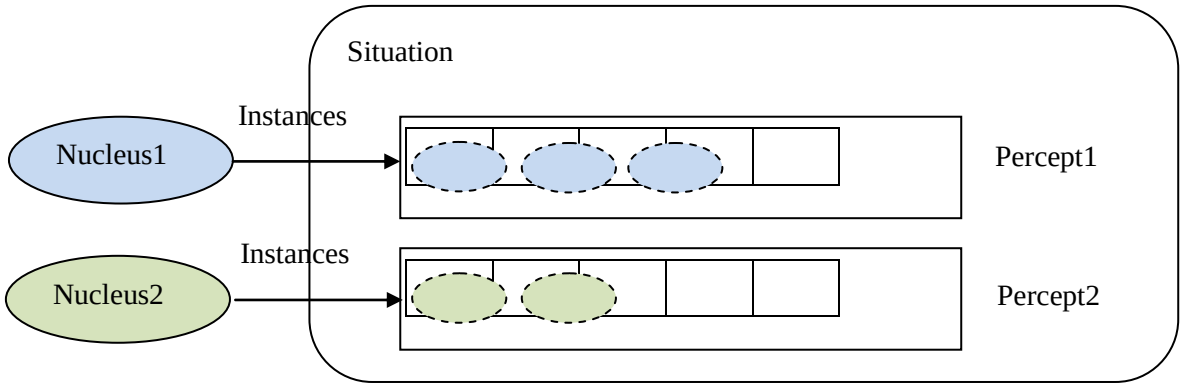



Fig. 9. Percepts are arrays that store instances of nucleus types.

of components to describe RGT problems:

1. *Nucleus Abstracts* 2. *Composite Abstracts* 3. *Sets* 4. *Actions*

The aim is to have a small number of basic units and rules using which it will be possible to build higher level abstracts without modification of the source code of the agent.

Users of the Graphical Interface have possibility to see the already defined abstracts, modify them. The Interface also has a graphical toolbox using which users can create units of the abstracts description language.

The definition of situations is given as compositions of values of distinguishable parameters [21].

The situations are modeled within the computer memory as compositions of instances of certain classes – **nucleus types**. For each type of distinguishable parameter a corresponding nucleus type is required.

The definition of nucleus types and their instantiation for building the initial situation of the problem is considered as a task for the user of the Solver. This means the user decides the level of presentation of RGT problem by defining nucleus types. This may depend on how detailed and constructive the definition of the problem is expected to be. The more detailed presentation of a problem requires lower level of nucleus types to be defined.

The elementary units within the situations are the instances of nucleus types called nucleus abstracts. The latters are defined as **classes** with **one numeric value** attribute. The Interface allows specifying the possible values of *value* attributes as defined by game specification [68] [22].

A nucleus type defines the range of possible numeric values of the attribute. And instances of the nucleus type have values from the corresponding range. Values out of the range can't be applied. Visually nucleus class is represented in the following way:

| Nucleus Type | | |
|---|---|---|
| value | operator | arguments |

Fig. 10. The format of the nucleus type's.

The name of the single attribute is "value". The possible range of values is defined using operators and arguments that are numeric. The figure below shows the permissible operators with the corresponding attributes:

| Range Description | | |
|---|---|---|
| Operator | Arguments | Example |
| =, != | number | value = 1; value != 2,3,7 |
| >, >= | number | value > 2; value >= 5 |
| <, <= | number | value < 2; value <= 5 |
| IN | Number range | value IN [1-5] |

Fig. 11. Permissible operators for defining numeric ranges of nucleus concepts' "value" attributes.

The **extending** (inheritance) relation between abstracts is defined as a constitution of a subset of the source abstract. The abstract instances which compose the larger set are referred to as parent or base abstracts. The abstract which extends the parent abstract is called a child abstract. Similar definitions could be found also in the OOP terminology [69].

The available by the rest of abstracts most common way to achieve the description of situations in the Solver is implemented using models of OOP [70] [71] classes that are called **composite abstracts** [68]. The **composite abstracts** include either composite abstracts which have already been defined or nucleus abstracts or types as attributes, as well as *Sets*. The attributes of the composite abstracts are named also as **inner abstracts**. The scope of an arbitrary abstract is defined as a multitude of inner abstracts and inner abstracts of its inner abstracts recursively till the most nucleus ones are considered.

a.

| Composite Concept | | |
|---|---|---|
| **attr1** | **Primitive Concept** | |
| value | IN | [1, 5] |
| **attr2** | **Composite Concept1** | |
| **attr1** | **Primitive Concept1** | |
| value | > | 1 |
| **attr2** | **Primitive Concept2** | |
| value | = | 2 |

b.

| Composite Concept | |
|---|---|
| **attr1** | **Primitive Concept** |
| **attr2** | **Composite Concept1** |

Fig. 12. Composite Abstract representation detailed (a) and short (b) diagrams.

12 (a, b) shows the graphical representation of a composite abstract, which contains two attributes – a nucleus abstract and a composite abstract with two attributes. It is possible to represent the composite abstract by using detailed and short representations.

In order to define a collection of objects in the solver **Sets** are defined. Sets are similar to arrays in programming languages with the difference that it defines common characteristics of elements [21]. They have two mandatory additional attributes: minValue and maxValue, which define the number of attribute instances in the set object.

Software provides ability of definition of **Actions** transforming situations of RGT problems. The term "action" will be used to refer to the model of "action" in the Solver, and "real action" term will be using when referring to the real actions.

| Vertical Line | | |
|---|---|---|
| field | | Field |
| cordX | = | :lineNumber |
| lineNumber | = | * |
| minCount | = | 8 |
| maxCount | = | 8 |

Fig 13. The graphical diagram of the concepts description graphical language, which represents a set “Vertical Line”.

In [68] it is also stated that he RGT class problem’s game tree contains models of all the possible situations of the game including the future states. For modeling the game tree it is important to have mechanisms for inducing future states from the model of the current situation. Software agents affect on the problem states by executing actions. The results of actions executions are changes within real situations. By simulating the real actions of the agents and by executing them over the models of situations it is possible to model the future states of the problem. Here and after we use a term "action" to refer to the model of "action" in the Solver, we will explicitly use the "real action" combination when mentioning the real actions.

In the Solver actions are defined as changes of the values of some attributes in the situation according to game rules [21]. Action items are composed from precondition and postcondition units.

Precondition defines the initial situation, which needs to be satisfied before action applies. In its turn, the postcondition is composed from a set of rules (represented as expressions) which evaluation results a post situation. They are similar to functors (function objects) of programming languages.

Thought the initial version of Interface provided abilities of definition of RGT problems, it has several shortcomings fixed in the next stage of development [23]. The main enhancements are described below according to [23].

Assuming that the perception module of the Solver is capable of distinguishing different nucleus abstracts and assign the same indexes to the ones appearing together (otherwise, there will be no way to assemble the characteristics of a contiguous object from a situation) there comes a need of a description unit which would define the expected group of nucleus abstracts representing the single contiguous object. We shall note here, that relations between different contiguous objects are not defined by their indexes (because the index is a pretty abstract concept, it can be basically anything, although usually, it is represented through 1, 2 or 3 dimensional coordinates).

Functionally, this new unit does not differ from the composite abstract, however, it should still hold several specific restrictions. We define this new unit as a special type of a composite abstract and name it ***a****bstract satisfying* ***r****ule number* ***1*** or, short, ***ar1***. The Rule number 1 dictates the following three restrictions:

- an ar1 must contain only nucleus attributes,
- there must be utmost one attribute of the given nucleus type,
- all attributes must belong to the same Id Group - i.e. have the same indexes.

In other words an ar1 represents the description of a position in a space, where Id Group serves as coordinates of the position. From the definition it follows that relations between attributes are implied in the ar1 (belong to the same Id Group), thus, ar1-s become flexible semi-indivisible logical units (they are still divisible to the nucleus abstracts, however, they compose the same logical object – just representing different characteristics).

This definition, and the introduction of ar1 concept plays an important role in the simplification of the definitions of relations between the attributes of composite abstracts. It also gives certain benefits in the implementation of the matching algorithm which will be discussed in the next chapter. In the graphical interface, the acquisition units of ar1-s does not differ from the one developed for composite abstracts. However, in order to threat the unit as an ar1, a special checkbox has been added to the graphical module which should be checked in the case of ar1.

Ar1-s, also introduce several further restriction in the definitions. Particularly, an abstract inherited from the ar1 automatically inherits the restrictions defined for the ar1, thus, itself becoming an ar1. This is expected, because the activation of a child abstract must lead to an activation of the parent abstract. Therefore, the rules cannot be loosen. The other restriction is that all the nucleus attributes should be encapsulated within the ar1 abstract. However, this doesn't affect the interface - such nucleus attributes are silently translated into the ar1-s. This is done to gain certain representational and fucntional efficiency.

2 of realized shortcomings of **Composite Abstracts** presentation were also fixed in [23] as described below.

Model of initial Composite Abstracts didn't allow definition of abstracts having multiple level nested attributes. In the example of *Check* concept it is seen that *Field Under Check* contained in *Check* also has composite attributes.

Within the handling of multiple level nested attributes handling also attribute accessing mechanism for different types of abstracts is developed using "." accessor. For example to access the attribute *tar* in *FUC* attribute of Check concept fuc.tar is used. Relative names are used to access and handle dependencies between attributes of a composite abstract.

In the same research **virtual abstracts** importance is sustained, where virtual abstracts are similar to abstract classes of OOP in their essence. An abstract is virtual if it contains at least one attribute with undefined relations (the attribute condition is undefined if it is set to equal to '?' special value to indicate the virtuality). To make the instantiations of such kind of abstracts possible one has to inherit from them and specify undefined conditions/relations. This procedure is called virtual abstract *specification.* Virtual abstracts serve as shortcuts and can be used in other abstracts as attributes. We call this mechanism a virtual abstract *usage.* There can be more than one virtual attribute used in an abstract. Consequently, the instance of the abstract can be built from various compositions of specific attribute instances.

### 1.3.1

Current RGT Solver provides Graphical Language Interpreter for definition of RGT problems and situations. The models allow regular acquisition of RGT domain knowledge.

Several improvements according to the shortcomings of initial implementation were provided.

Developed models of knowledge presentation do not present them in a common structures thus making impossible making common decisions when acquiring new abstracts, as well as common knowledge matching algorithms to the situations are required.

Also RGT problems require interpretation of negation in concepts, e.g. "Mate" concept requires "king has **no** defense" [22], which is the negation of "king has defense" chess concept. Defined actions did not provide indication of side which performs the action and did not provide algorithms for their application on the situation to transform them. These problems are subject to this research.

An urgent shortcoming of the developed package is that there are no means provided for acquisition of strategy defining knowledge, i.e. goals and plans, as well as strategy searching algorithms based on those structures.

## 1.4 PERSONALIZED INTERACTIVE TUTORS IN CHESS

Testing advances of students and computer-based interactive personalized tutoring is an urgent problem. While testing can be interpreted as an RGT problem, the methods of tutoring, explanation and testing of knowledge understanding have to be adequate to models of tutoring by experts.

The classical approach of teaching includes teacher and implies interaction between the students and the teacher.

- Some concept of chess is defined and is explained in depth. The depth of explanation depends on a student, because the student also needs to understand the concepts used in definition of what the teacher define. If the student knows all the required related concepts, then the explanation is not deep, it just defines the teaching concept. For each of the related concepts the following steps are performed if required. Usually if a student has missed an

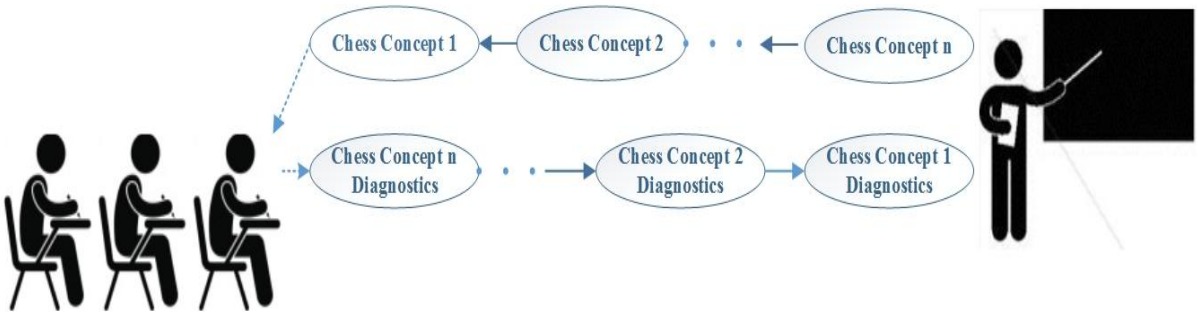



Fig. 14. Chess teaching classic approach (sequential learning, concept by concept, test by test).

  explanation of some concept, teacher will not go back and teach for that concept again in future sessions or levels.
- Chess exercises and puzzles are suggested to solve. If the student is able to solve the problems then it is considered as already learned. If the student is unable to solve the problem, then it is assumed that the student did not learn the concept and step ‘a’ needs to be performed again.

The mentioned mechanism of teaching chess a) requires teacher’s effective involvement in the studying process, b) is not flexible enough c) is not personalized for each student and does not differentiate between autistic, genius and other levels of students in learning, usually provides the same approach to all of the students.

Other chess learning mechanisms can include chess books [73] or software with static databases, e.g., video lecture course database, where all lectures are included in videos [74], after

each video lecture there might be suggested learned concept testing exercises, too. The restrictions of these mechanisms are: a) they are not interactive b) mechanisms might be personalized but the student needs to indicate the level of his/her knowledge by him/herself, thus making difficult having results in different performance levels, e.g., for autistics or geniuses

- the feedback for testing of the suggested learning concept exercises is not well organized
- they do not provide student performance measurement mechanisms.

Many psychologists and education specialists such noted in their works that chess education, as well as other disciplines education must take into account the individual psychological characteristics of a student making the teaching more personalized [75] [76] [77]. Taking into account the individual features allows to maximally uncovering one's internal resources, to take into account the nuances of student's mental processes.

Among the methods that, in one way or another, take into account the individual characteristics of the students, differentiated, individual and individualized approaches to teaching are classified [78] [79] [80]. Differentiated approach involves the allocation of similar individual, personal qualities of students. Individual approach involves training without contact with other students, but in the same pace for all. The individualized approach is built taking into account the peculiarities of each individual student. The concepts of education individualization and differentiation are revealed in the works of I. Osmolovskaya, I. Unt, A. Kirsanov, A. Granickaya, V. Shadrikov, I. Smirnov and others.

In [81] personalized tutoring approach is discussed, particularly chess tutoring is discussed for ordinary and autistic children tutoring process requirements are discussed, where several requirements are revealed and certain software is suggested as a personalized tutor for chess. Software selection is substantiated. Following tests for personalized tutoring are suggested 1) generation by the models of meanings positions on the board to be commented by the experts for possible corrections of the models, 2) using complementary means to minimize the threat that only a part of the meanings of the expert classifiers were “caught ” by the models like the following tests: * experts generate examples which are tested by the models,
* known positions that meet certain criteria of completeness are taken from some repository to compare expert responses with the model ones. Acquisition of concepts can be done by a variety of strategies like the one where tutors sequentially decompose the given concept A, for each decomposition reveal not learned units, decompose each of them to find new unlearned components, etc., and, finally, composing the map of acquisition of A layer by layer move up while achieving the learning of each missed component of the layers. Comparing the suggested approach

with the suggestion of knowledge-based chess bishop-pawn endgames tutoring in [82] the following extensions are revealed - considering not only endgames but arbitrary positions including the complex middle game ones, - tutoring learning the strategies of important for applications of other competition problems, particularly the intrusion protection, defense, management ones. More extensions are suggested for the algorithms to enable tutoring math.

### 1.4.1 DESIGNING PERSONALIZED INTERACTIVE CHESS TUTORS

Generally computer-based tutors based on expert knowledge presentation models are expected to be developed. In the following section we discuss amodel of tutoring to be adequate to the expert one.

Students want to learn chess concepts provided by their names. Tutoring Software is searching for the concept in its chess concepts graph. If the concept is found, software provides concept explanation by giving the name, relations with other known concepts, regularities, description if defined. Then it provides concept examples. The process is recursive, the related concepts explanations and examples can also be requested. Afterwards software checks if the user understood the concept by chess exercises and puzzles. If the student does not pass checking, expected valid solution is shown, he can request the concept explanation and examples again, so the process can be repeated.

Diagnostics that checks student's understanding of a concept can be requested at any time. So it can also be requested before starting of learning.

Designed software can provide an explanation of any defined concept on board visualization description level. The lowest level of explanation is the explanation of the lowest level concept by visualization on the board. The lower level explanation is out of software chess tutoring scope.

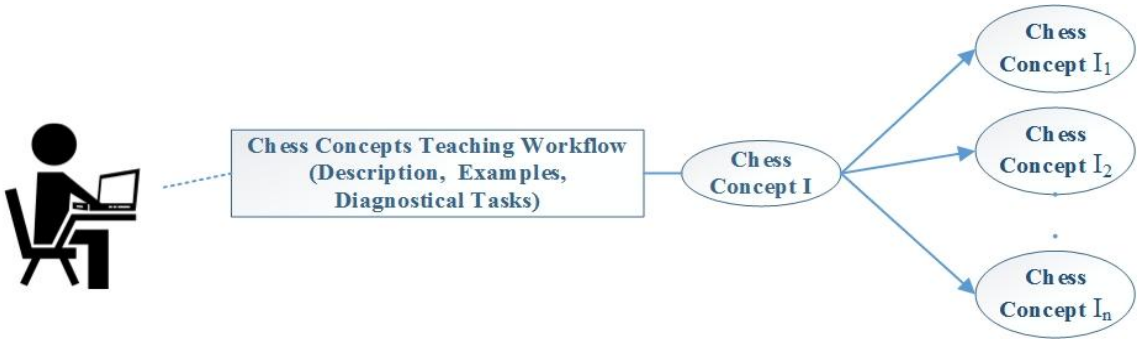



Fig. 15. Suggested chess tutoring approach (interactive personalized software Workflow, random chess concept learning, learning of any required chess concept I, related to concepts I1, I2, ..., In).

Expectations of chess tutors:

a. provide personalized tutoring, enabling individual approach to any student,

regardless of the level of student's understanding, where autistic students, regular chess students, students with huge speed of performance in learning chess can be involved and each of them will have personalized learning.

b. make interactive platform where any learning concept can be described level by level depending on the student knowledge, where problems and chess puzzles are being suggested and student solutions are checked

c. not involve human teachers, they can still be required in several cases, 1) the teacher will be required for inserting the whole set of chess domain knowledge which will be transferred to students during the teaching process, 2) if the software meets difficulties in explaining and teaching for some chess concepts the final action is to ask the human teacher to explain the concept to the student and improve the definition of that chess concept in the software.

d. provide feedback and analyze the results. If solutions to the suggested exercises do not match the solutions suggested by the software while checking test results for each learned concepts, they will be corrected and explained step by step.

e. rate students and compare them against rated and validated chess players to validate the learning results.

## 1.5 SUMMARY OF CHAPTER 1

In the chapter the problem of development of human – computer communication is considered which is a central problem in AI [83] [84] [85]. Particularly, the problems of knowledge acquisition, matching to the situations are considered. This problem exists and is an actual problem since the first attempts of expert systems. Our aim is to develop approaches that will enable human knowledge acquisition and adequate representation by computers. We consider the RGT class of problem, that includes problems which space of solutions can be represented as *reproducible game trees*. The problems of RGT class meet the following requirements: there are: (a) interacting actors performing (b) identified types of actions in the (c) specified types of situations. d) Some of situations are identified as benefits (goals) for each of the actors. Actors perform actions to transform situations.

IGAF1 and IGAF2 (Intermediate Goals At First) algorithms based on common knowledge planning and dynamic testing were implemented and successfully experimented for the problem of Intrusion Protection of computer networks.

The PPIT (Personalized Planning and Integrated Testing) RGT Solver initial implementation induced strategies relying on personalized expert knowledge for the given problem, where

knowledge appears in forms of goals and plans. Their adequacy for Reti and Nodareshvili etudes suggested by Botvinnik was experimented.

Future developed RGT Solver provides ability of RGT problem regular acquisition, which was successfully experimented for chess.

The following problems were revealed in the chapter:

1. The problem of extending the current RGT Solver to model knowledge of expert knowledge, including goals and plans and develop algorithms for matching those models to the situations;
2. Ensuring the models and algorithms of matching are adequate to RGT problems;
3. Developing knowledge-based strategy search algorithms for RGT problems within RGT Solver;
4. Ensuring the adequacy for strategy search algorithms in intensive knowledge-based analysis requiring RGT problems;
5. The problem of interactive personalized tutoring for chess.

6.

# CHAPTER 2. ENHANCEMENTS AND PROOF OF ADEQUACY OF KNOWLEDGE REPRESENTATION IN REPRODUCIBLE GAME TREES

## ABSTRACT OF CHAPTER 2

The representation of knowledge in the Solver of RGT problems is described in Chapter 1. However the developed algorithms and structures [19] [20] did not provide basic means for definition of RGT problem related expert knowledge for knowledge-based strategy search algorithms.

Experiments for testing the adequacy of knowledge representation in RGT Solver were not provided in our previous works to ensure the models are able to acquire RGT problems and not restricted to chess acquisition.

In this chapter two general problems will be discussed:

1. The extension of knowledge representation structures by enriching their structures and activation algorithms, particularly:
    a. improvements in definition, including negation integration, continuous sets, actor side indication,
    b. goals and plans structures
2. The proof of adequacy of overall knowledge representation by certain pieces of expert knowledge, particularly by chess, marketing and intrustion protection knowledge experiments.

## 2.1 MODELS FOR PRESENTATION OF ABSTRACTS AND ALGORITHMS FOR MATCHING SITUATIONS TO THEM

Considering the properties of different knowledge representation models and, particularly, the shortcomings of the OOP languages underlined in the chapter one, we develop a knowledge representation model of English grammar, particularly *be-*, *have-*, *do* dimensions are integrated. The refinement of meanings and the relevance of their representation using *be-*, *have-*, *do* dimensions of English grammar are argued in [86], [2], [39].

Strategy search implies, that there must be a procedure which will detect active knowledge pieces in the situations and will provide them to the decision making algorithm at each cycle of processing.

Presentation models of knowledge shall allow matching classes via matching its attributes – the instance matching functions of modern language classes do not use the class' attributes for checking the instance existence. They operate only with direct links between classes and attributes that were created during the class instantiation.

### 2.1.1 PRESENTATION OF KNOWLEDGE IN GRAPH OF ABSTRACTS AND THEIR INTEGRATION ALGORITHM

The Solver embodies the universe of internal models of knowledge as a mixed graph. It continuously acquires the defined abstracts into the internal graph - Graph of Abstracts (GA) by finding and constructing appropriate be-, have- or do connections with already existing nodes of Graph.

The acquisition of knowledge leads to having different types of GA nodes which we will describe bellow with their main characteristics and roles:

*NT - Nucleus Type,* the types of the smallest representation units of knowledge. They compose the set of GA roots. The user (expert) is required to define them in order to construct more complex abstracts.

*NA - Nucleus Abstracts,* the instances of Nucleus Types with additional restrictions (they represent the subset of the types' values).

*AR1 - Abstracts satisfying Rule number 1,* these hold a level between nucleus and composite abstracts and play an important role in the simplification of the definition of relations between composite abstracts' attributes. Rule 1 defines the following three restrictions:

1. *abstract contains only nucleus attributes*
2. *there is at most one attribute of a given nucleus type*
3. *all attributes belong to the same IdGroup*

In other words an AR1 represents the description of a position in a space, where IdGroup serves as coordinates of the position. From the definition it follows that relations between attributes are implied in the AR1 (belong to the same IdGroup), thus, AR1s become flexible semi-indivisible logical units (they are still divisible to the nucleus abstracts however they compose the same logical object – just representing different characteristics).

*CA – Composite Abstracts,* the most common form of the abstract representation. They are composed of AR1s, Sets or of other Composite Abstracts. They are able to represent all kinds of regularities.

*SA – Set Abstracts,* these nodes are used to represent the group of abstracts with similar characteristics. They are composed from a single composite abstract element and a rule specifying the number of elements in the group.

*Action* – these are the only nodes which are representing actions in the GA. They are composed from Pre-condition: a composite abstract and a Post-condition: a set of expressions. In the GA the Precondition is separated from the Action node and connected to the later with a Do connection, while, with the rest of the graph the Pre-condition is connected like other abstracts.

VA - Virtual Abstracts, abstracts which have undefined values for their attributes. They are similar to the Interfaces of OOP languages (Java etc.) with a difference that here virtuality is enhanced by the set of undefined relations for attributes while in OOP it is usually done by undefined methods. Consequently, they inherit almost all advantages of OOP interfaces (polymorphism etc.). It is obvious that the use of Virtual Abstracts is possible only if there are inherited abstracts specifying the undefined relations because virtual ones are not able to compose instances themselves by putting together the instances of attributes. That is why Have connections are not constructed

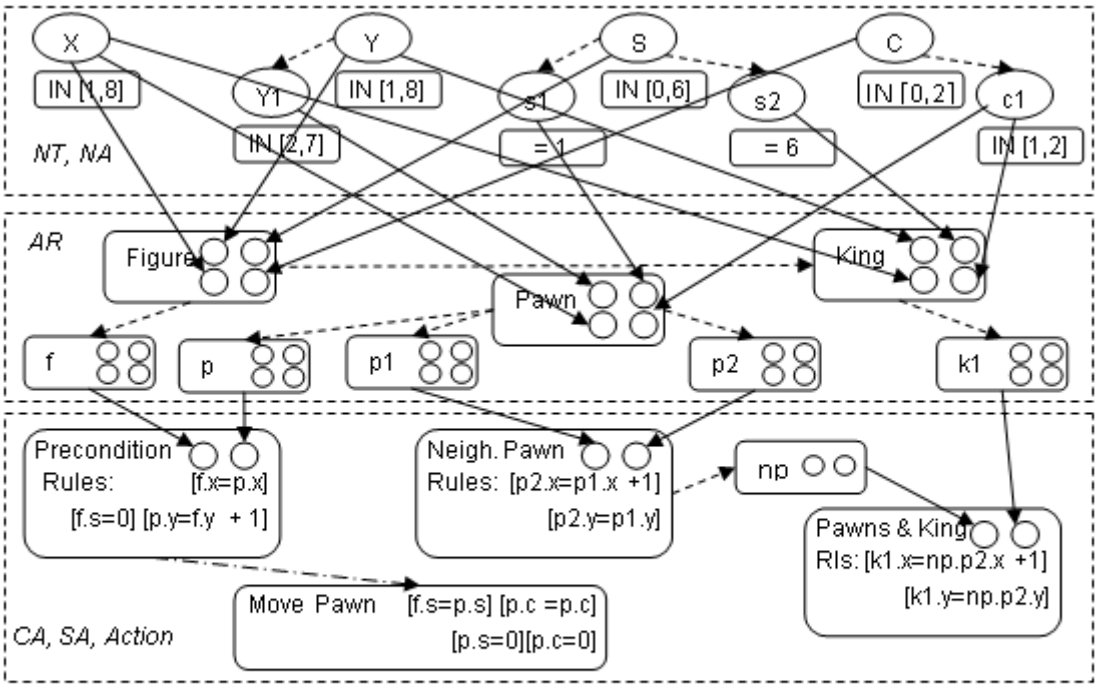



Fig. 16: A fragment of a Graph of Abstracts for the example of chess representing Be (dense), Have (dashed) and Do (dot-dashed) connections.

between them and their attributes. The integration of VAs into the graph is done by Be connections to their children abstracts and Have connections to their composers.

```
# The function of decomposition and integration of a new abstract into
the graph of abstracts.

function GANode InsertAbstract(Abstract: source)
     inputs: source              # the abstract which needs to be
inserted
     result: node: GANode        # the node in the graph of abstracts
     if source is Action:
          node <- CreateActionNode(source)
          precondition <- InsertAbstract(source.Precondition)
          CreateDoConnection(node, precondition)
          return  node
     if source has parent and is identical to source.Parent:
          return source.Parent.GANode
     if source.Type is NucleusAbstract:
          node <- CreateNucleusNode(source)
          if source has not parent:
               node add to GA root list
     else if source.Type is AR1:
          node <- CreateAR1Node(source)
          foreach attribute: source.Attributes
               GANode attr <- InsertAbstract(attribute)
               if source is not virtual:
                    CreateHaveConnection(node, attr)
     else if source.Type is Composite Abstract:
          node <- CreateCompositeAbstractNode(source)
          foreach attribute: source.Attributes
               attrNode <- InsertAbstract(attribute)
               AddRules(node, attribute.DependentRules)
                    if source is not virtual:
                         CreateHaveConnection(node, attrNode)
     else if source.Type is Set:
          node <- CreateSetNode(source)
          element <- InsertAbstract(source.Element)
          CreateHaveConnection(node, element)
          if source.ElementCount is not dependant:
             AddRules(node, source.ElementCount)
     if source has parent:
          if source is usage:
               CreateBeConnection(source.Parent.GANode, node)
          else:
               CreateBeConnection(node, source.Parent.GANode)
    return node
```

Fig 17. The pseudo code of the knowledge acquisition and integration algorithm.

The integration of different types of abstracts is described in figure 17.

The basic idea behind the algorithm is to create a Be connection to the Parent node if any, Do connection to Action node and Have connections to the attributes. The algorithm takes care of extraction of regulations between attributes of composite abstracts. We classify three kinds of relations: *independent, dependent and pending.* The relations are independent if they can be handled in the sub abstracts and dependent, if they are feasible to handle in the scope of the abstract (excluding the independent ones). In which case they are bound to the abstract and the later takes care about their satisfactions when constructing its instance. On the contrary, the pending regulations are ones which contain references to abstracts out of the current scope. They are pushed down in the graph to the abstracts which have enough information about all pending attributes.

However, the picture changes if the abstract is virtual. As we said above no Have connection is built between Virtual Abstract and its elements. Well, this is not the end of the story. We said that for using VAs one has to inherit from them and specify the undefined relations between attributes – this process we call *VA specification*. On the other hand the main purpose of VAs is to create a virtual shortcut to a big subset of real definitions, thus, it is supposed to be used in other abstracts. The difficulty is that in this case, too, one has to create an attribute in the abstract by inheriting it from the VA – this process we call *VA usage*. Thus, the task in the front of the acquisition algorithm is to distinguish between VA specification and usage and create proper connections in the GA. The difference between these connections is that VA node shall be activated by VA specifications (it does a kind of pre-filtering) and VA node shall activate the usage node (which does post-filtering) while, by definition, VA is the Parent node both for VA specification and for VA usage.

The beauty of the solution in the algorithm is that it does not add any additional connection type but rather builds a redirected Be connection between VA and VA usage. In this case the activation of VA will automatically forward the instance to its parent, which (VA usage) will apply its rules and will either fire or halt the further processing of the instance.

Another interesting aspect of the Solver is the way it interchanges meanings with users, the way it extracts the initial connections of the new meanings and how it manages the modifications of the initial definitions of meanings. The Store of Meanings (Fig. 18.) module is designed to cover these functionalities by:

*constructing abstracts* – the meanings are received from the interface as a serialized json strings, which contain the list of all attributes and values including the types (parents) of attributes. This complete representation helps the Abstracts Manager to extract the exact relations for each attribute as well as bind to them the restrictions applied by parents.

*managing abstracts* – from the abstracts' point of view its storage is like a graph, as because it keeps the references to its attributes. However, from the Solver's view the storage is a composition of lists of different type abstracts (nucleus, composite, set and action). Solver takes care about the modifications/corrections in the abstracts. In that case it triggers a remove of the old meaning from the Graph of Abstracts and reinjection of a fixed one.

*storing and loading meanings from permanent storage* – the Solver performs an incremental (level by level) store and load of meanings into the key-value MongoDB store. The incremental character – allows the Solver to load only the necessary part of the meaning into the memory as well as supports point updates – by rewriting only changed attributes instead of the whole abstract.

*finding, serializing and returning the meanings* – it first checks if the required meaning exists in the in-memory store and if it is not, the request is forwarded to the permanent one. After getting the corresponding abstract (including attributes) Solver translates it into a json message and sends back to the interface.

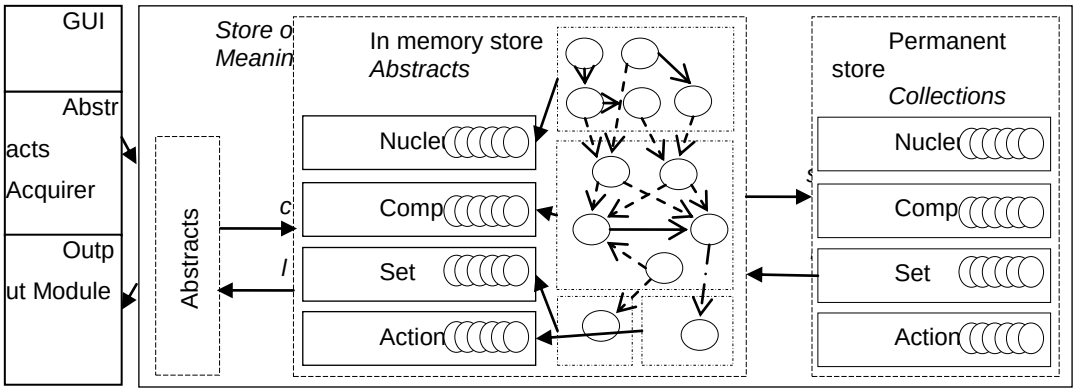



Fig. 18: Store of meanings. It shows create/store/load cycles of meanings'

## 2.1.2 ALGORITHMS FOR MATCHING ABSTRACTS TO SITUATIONS

In the Solver, we **model a situations** as prints of relalitites with a set of groups of contiguous nucleus attributes. This modeling allows having a flexible representation of a print. The AR1 is a semi indivisible abstract work with which is more effectively in situation processing.

The Solver treats print as a bunch of instances of AR1 abstracts, thereby, implying that perceiver is capable to produce contiguous objects after the pre-processing phase of stimulus.

Although each *t-print* represents the active universe at time point *t* the difference between successive prints is considerably smaller compared to the prints' sizes. This property is used by the

Solver when memorizing t-prints into the *store of prints.* It distinguishes two types of entries, namely, *t-print* and *delta-print*, where *delta-print* stands for the difference between prints at time *t+1* and *t.* Consequently, the store is composed from a list of t-prints serving as checkpoints (snapshots) at specified times and chains of delta-prints between them.

This representation optimizes the algorithm of meaning matching, which after having processed the t-print at time $t_1$ can get the list of matched meanings for time $t_2$ by only processing delta-print instead of a complete t-print.

Solver's **processing of situations** iteratively triggers the matching or ablation of the instances of abstracts lying in the Graph of Abstracts, consequently, building the matched set of abstracts in the Solver.

The procedure of composing abstracts from sub-abstracts represents a classical *constraint satisfaction problem (CSP)* which influenced heavily on the design of the graph of abstracts. The latter allows effective synthesis of abstracts' instances by traversing the nucleus instances through the graph and checking the identical conditions *only once*. As a base a standard Rete [87] CSP algorithm is taken. Consequently, GA nodes are divided into two functional categories: *filtering* and *conjunction* nodes. The first ones decide whether to discard or propagate further matched instances by applying conditions. While, the *conjunction* nodes serve as assemblers trying to match different sets of matched attributes' instances for generating a new instance corresponding to the node's abstract. Thus, each abstract's matching fires an instance of its type, which is a regulated composition of instances fired by sub attributes. This means that parallel to the Graph of Abstracts the Solver maintains a less obvious Graph of Instances (or partial instances) which is tightly coupled with the former.

The different characteristics of GA nodes draw distinct rules for instance formulations. Bellow we will discuss the matching procedures for each node type.

Nucleus nodes serve only as filters in the GA, hence, input instances satisfying the abstract's conditions are fired forward to the node's descendents or composers, otherwise they are dropped. AR1 nodes are representing *conjunction* nodes checking the id groups of attributes to compose new AR1 Instances. For each matched attribute they check whether there is already a partial instance with received id group. If there is not then a new partial instance is created and the received instance is set as an attribute to it, otherwise, the instance is set to the existing partial instance and if the latter gets complete (all attributes are set) the AR1 Instance is fired. Note that an instance is called *partial* if it lacks some of its attributes.

Composite nodes; as one would expect, are combining the characteristics of both node types uniting the instances of attributes and filtering out the ones which violate the rules. The complexity in the abstract matching is the necessity to check all possible combinations of the attributes. In the Solver it is achieved by keeping the list of partial instances which represent all allowed (by rules of the abstract) combinations of already arrived sub-instances. It is done by the following steps (Fig. 19):

- create a new partial instance which contains only the new instance as an attribute and clone each existing partial instance which doesn't have the received attribute and apply the new instance on it
- if the dependency rules are violated discard the cloned instance. The interesting advance here is that only dependent rules are required to be checked because independent ones shall already be satisfied when sub attributes fire their instances
- else if the partial instance gets complete, fire it forward, otherwise add it to the list of pending partial instances.

This procedure guarantees that all possible combinations will be checked and the promising ones (which still lack some attributes but rules are not violated) will be held in the partial instance list while satisfied ones will be fired.

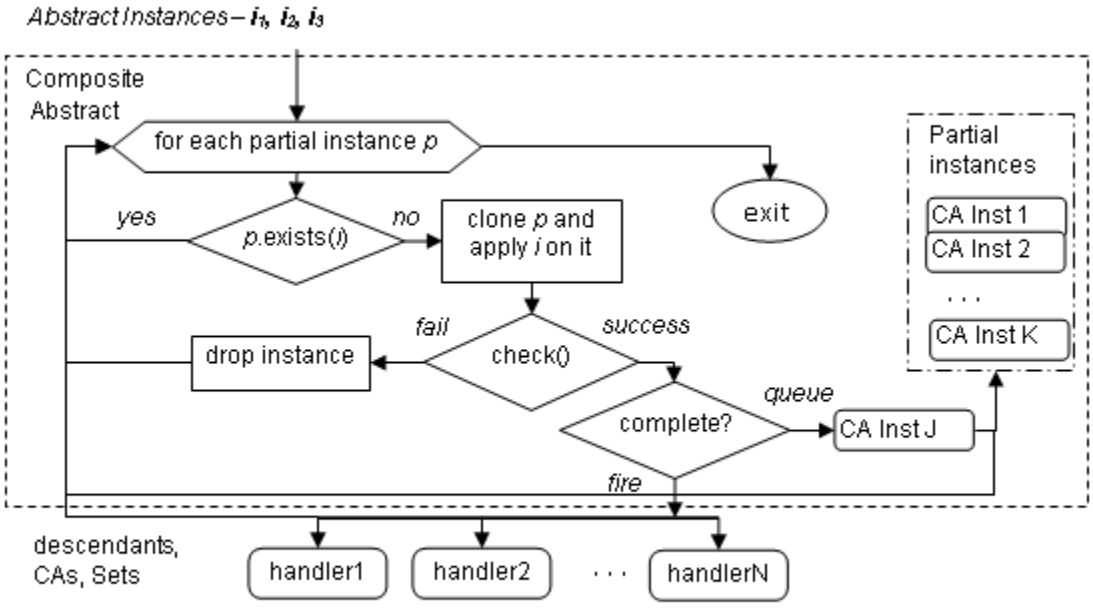



Fig. 19: Composite abstracts' activation algorithm. It checks all combinations of sub-meanings to compose an instance and, consequently, drops, queues or fires it

*Dependency rule checking* - Each dependent condition is represented as a Rule object which specifies relations between nested nucleus attributes of the abstract. In order to check whether the rule is satisfied or violated the correct references to nucleus instances are required to be found and set from the abstract's instance. This is done by using nested attribute naming capability, which uses composer abstract's name as a prefix separated by '.' delimiter: from the Rule full names of dependent nodes are extracted and checked if the instance contains them. If it does then corresponding attributes are set and the rule is checked, otherwise the rule remains in a pending state represent all allowed (by rules of the abstract) combinations of already arrived sub-instances.

- create a new partial instance which contains only the new instance as an attribute and clone each existing partial instance which doesn't have the received attribute and apply the new instance on it
- if the dependency rules are violated discard the cloned instance. The interesting advance here is that only dependent rules are required to be checked because independent ones shall already be satisfied when sub attributes fire their instances
- else if the partial instance gets complete, fire it forward, otherwise add it to the list of pending partial instances.

This procedure guarantees that all possible combinations will be checked and the promising ones (which still lack some attributes but rules are not violated) will be held in the partial instance list while satisfied ones will be fired.

The matching of Sets is as simple as collecting the same type elements (from the matching of the single attribute) and firing all possible groups which appear in the given range

The action matching is the same as the matching of its precondition which is represented as a composite abstract. Therefore it doesn't require any special handling.

The matching of virtual abstracts is done as described in Fig. 20

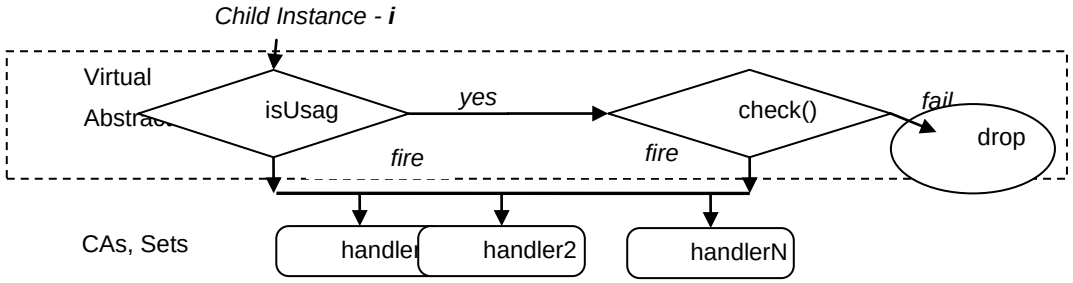



Fig. 20. Virtual abstracts' activation algorithm. If it is a usage then filtering conditions are also checked else, it is immediately fired further.

- they are being matched not by attributes but rather by matching of their children when the later ones get matched
- when they are used as attributes in other abstracts (the *usage* of virtual abstract) then some new relations might be defined in the abstract which shall be checked against the received instance in order to fire it forward. Here they behave as filtering nodes.

This procedure of checking rules against the instance in the *usage* of virtual abstract is called *post-filtering.* For example it can be used to search for the meaning like "Find all attacked fields by white figures in the given chess situation". This can be modeled using "Find all attacked fields" generic abstract and to post-filter it with "Where attacker has a white color".

## 2.2 ENHANCEMENTS IN THE KNOWLEDGE REPRESENTATION

Developed knowledge representation structures had several shortcomings which are being highlighted obviously when PPIT algorithms search for strategies using the knowledge pieces acquired by the Solver. The following main shortcomings are noticed:

1. Definion of negations are not provided. For example definition of "Mate" concept in chess also requires definition of negated knowledge "king has no defense". We developed structures and activation algorithms for this type of concpets, which will be discussed below.
2. Definition of continuous sets. In general it is revealed that activation of Sets in knowledge processing is a process requiring exponential time (dependent on Set size). Particularly in chess there may be defined concepts like "emptyFields" with minValue=3, maxValue=8. The activation of this much abstracts in the processing is not expected.

   Also in many situations concepts of continuous sets are expected, which are basically subclass of current provided Set, where only continuous sets are activated, while current implementation activates any of them. Improvements of strucutres of Set as well as their activation algorithms will be discussed.
3. Actor side indication in the actions. The previous implementation of definition of actions in the RGT Solver provided ability to define actions without indication of the side which performs the action, which brought to confusion when searching for strategies. Strategy search algorithms need to know the acting side. For example in chess, when searching for allowed actions on the given situations the algorithm needs to perform it for the certain side, black or white, while current implementation did not provide ability to identify the

actor. Below the solution to the described problem will be discussed, as well as application of actions to the given situations is discussed.

Structures of plans and goals required for definition and execution of PPIT strategy search algorithms in RGT Solver are discussed. These structures also represent RGT knowledge pieces to be acquired by the solver.

### 2.2.1 INTEGRATION OF HANDLING OF NEGATION, CONTINUOUS SETS AND ACTOR SIDE OF ACTIONS IN KNOWLEDGE REPRESENTATIONS

Described in section 2.1 structures of knowledge presentation provide ability of definition of RGT problems' strategy related basic knowledge pieces, however several more enhancements are required.

#### 2.2.1.1 INTEGRATION OF HANDLING OF NEGATION

Negation is intensively used knowledge representation concept in RGT problems space, particularly in chess there are various examples of its usage, for example king move to a position is allowed only when the target field is not under attack, and the concept "field is not under attack" is required for definition of king moves. The concept of negation in RGT Solver is integrated into the Graph of Abstracts.

Knowledge of RGT problems in the Solver is represented in the Graph of Abstracts, where acquisition of each type of abstract is handled specifically to that type.

We developed algorithms for integration of negated concepts into the Graph of Abstracts and for activation of them when matching the situation. The algorithms work as described below.

1. Integration of negated concepts. For each type of abstract in RGT Solver we provide ability to indicate if an abstract is negated or not. If a abstract is marked as negated, the integration of the abstract into the graph is done by two main steps:

A) Not negated version of the concept is integrated into the graph of abstracts as regular integration mechanism.

B) A new node for the negation is created in the Graph of Abstracts. A new "be" connection is created between node created during step A, the newly created node is the child for the node create in step A.

All of negated nodes are kept in a list for direct access.

2. Activation of negated concepts. The negated concept is represented as a separate node in the Graph of Abstracts. The matching process with negation requires iterative activation of abstracts by following steps.

For the first step the regular activation process is being done. Abstracts which are negated not being activated at all. Their parent abstracts can be activated in a regular manner if found on the situation.

For the second step of the activation for each of negated nodes parent activation is being checked. If the parent node has no active instance on the given situation, then an instance of the negated abstact is being fired and the process of activation is continued in regular way.

During the third step activation of any new negated node is being checked. If there is any activation during the second step, step two is being executed again.

The process of matching is stopped when no new activation is triggered.

It is worth to mention that any of negated concepts has a parent concept which is not negated version of the same concept; this allows the algorithm to search for parent activations for each of negations in second step execution.

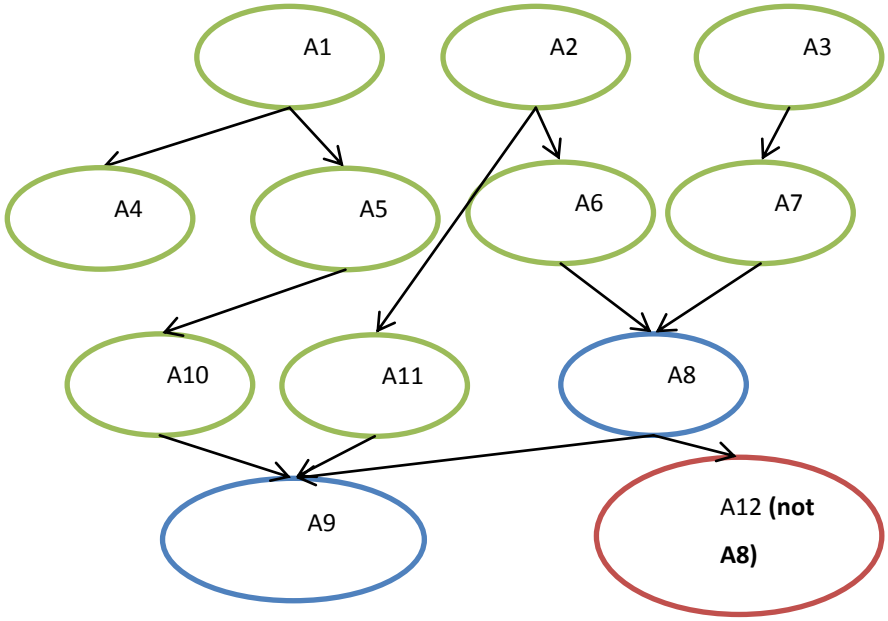



Fig. 21. structure of nodes with negted node integrated

### 2.2.1.2 INTEGRATION OF CONTINUOUS SETS

Representation of Set in RGT Solver has the shortcoming of definition of continuous sets. The importance of continuous sets existence is factorial size of active instances for any activated Set on the given situation. This leads to restriction of Set usage as it is taking too long to activate Sets. For

example activation of a Set in chess, defined as an array of fields of the board with minValue = 3 and maxValue = 8 requires several minutes. However the usual matching does not expect to find any combination of those fields: continuous fields set is expected instead. In fact continuous Sets are subclasses of not continuous Sets.

This led us to the integration of new attributes and handling of them in Set to enable continuous sets.

For definition of continuous sets a marker is provided. To define the Set as continuous just the marker is needed, also the definition requires the direction by which the Set needs to be conituous, for example in chess they can be X and Y coordinates.

The activation is different from regular activation of Sets. The regular activation of Set just checks if the given active instances count is between the given ranges of minValue and maxValue.

For the activation of continuous ones, the algorithm first finds the lower and upper bounds for the given active instances. For example if the algorithm needs to activate a set “field” abstracts of chess, defined as “empty vertical” Set abstract, where minValue=2 and maxValue=8, direction=X, it finds the lowest and highest values of X fom the given instances of “fields”. If the distance between lower and upper bounds is more or less than number of instances, then the Set is not matched for current stage. The algorithm then iterates over each of the instances and checks if all the X coordinate values are covered and if so, then the instance is of Set abstract is active.

The algorithm and structure provide widely used Set abstract definitions in RGT problems, particularly in chess which cannot be defined with regular Sets that Solver had before.

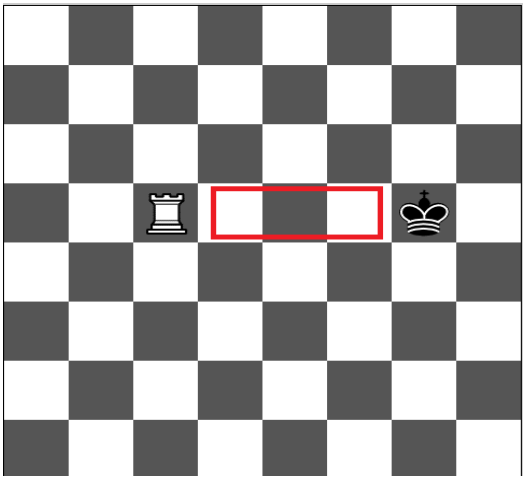


Fig. 22 FieldsBetween concept which requires definition of continuous Sets

### 2.2.1.3 INTEGRATION OF SIDE INDICATOR

The representation of actions in Solver did not provide ability to figure out the side which performs the actions. Since Actions in the Solver are only composition of preContion and regularities that change the preCondition, there were no means to find the side in the definition which performs the action. This issue is solved by adding a new special indicator where the attribute representing the key concept is indicated. The key concept is the concept that represents the side of actor, partifularly for chess, the key concept is “figure color”, and for an action “move king” the indicator may be “king.figureColor” which shows that king’s color is the concept which indicates the side of actor.

The activation process and changing of the situation require the mentioned side indicator in order to provide correct active set of Abstracts and Actions and correctly transform the situation.

The solution algorithm is the following: the algorithm of activation now checks if the action and side which acts on the situation are corresponding to each other. Only the actions that correspond to acting side are activated.

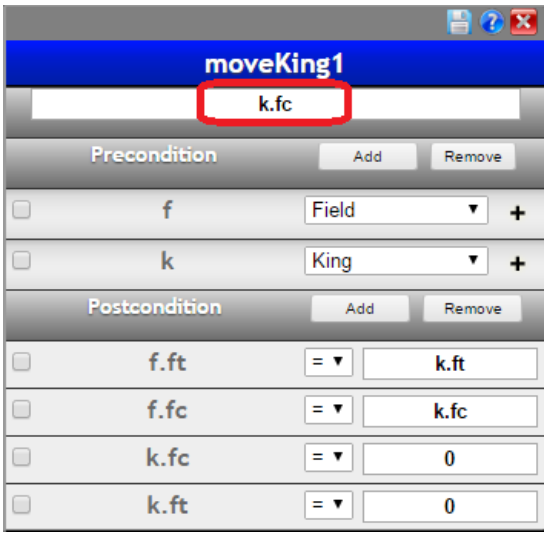



Fig. 23 Indication of sides when defining actions. In the following k.fc representing king color is the side indicator for moveKing1 action

The application of the action to the situation is done by the following algorithm:

1. Current situation is being copied; all of its nuclear instances are being copied by the same groupIds.
2. All the expressions defining the change by the action are evaluated and instances that are being changed in the new situation are found.

3. New nuclear instances with changed values by the actions are created.
4. Old instances in the copied situation are replaced by the changed instances.
5. Copied situation is returned.

### 2.2.2 STRUCTURES OF GOALS AND PLANS

Strategy searching algorithms in PPIT rely on plans, where plans represent general descriptions of strategies. In PPIT algorithms plans are compositions of goals, where each goal has its priority in the plan.

Both goals and plans are certain pieces of expert knowledge, thus we develop structures of those types by enhancing RGT Solver to be able to construct strategies based on domain knowledge defined with these structures.

The following requirements have been revealed for the goals of RGT problems:

A. It needs a preCondition situation, for which this goal is applicable, because there are situations where a goal is not achievable, e.g., if the situation contains only two kings and a pawn, a goal like “make check with the queen” can’t be applied. This basically defines the pattern of situations where goal is meaningful. Note that for some goals the preCondition can be any situation, so this is not obligatory to define some pattern in preCondition.
B. It needs to have a postCondition situation. This is the situation which appears when the goal is achieved, e.g., if the goal is “make check with the queen”, after it is achieved the opponent king is under check of queen in the given situation, this describes the postCondition situation. This defines the pattern of achieved by the goal situations. Similar to preCondition, postCondition also can be any situation.

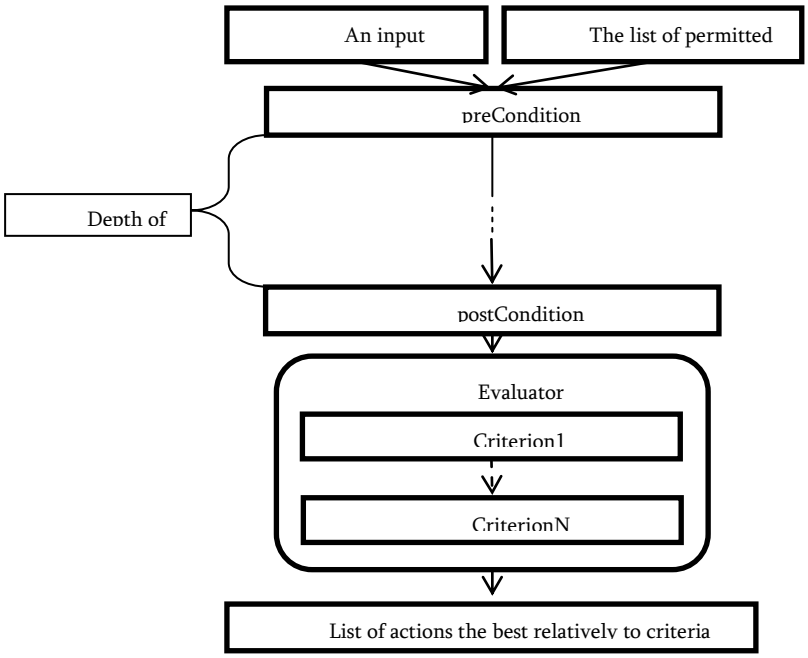


Fig 24. The structure of goals, their inputs and outputs.

C. For some goals the depth of game tree needs to be more than one move, e.g., if the goal is “make perpetual check”, we need to construct a tree and make several moves to see if this goal can be achieved.

D. Goals need to have some evaluation. There are goals like “put mate” or “avoid stalemate” where there are only two evaluation states, which indicate whether the goal is achieved or not, but there are some goals which do not show “an achieved” or “not” result, they show how good the goal is achieved, e.g., a goal “keep king closer to the opponent king” goal does need some criterion to define that the lesser distance between kings is, the better is the goal evaluation. For that purpose we define evaluator, which is a set of prioritized criteria that are being defined to evaluate the goal. For the above

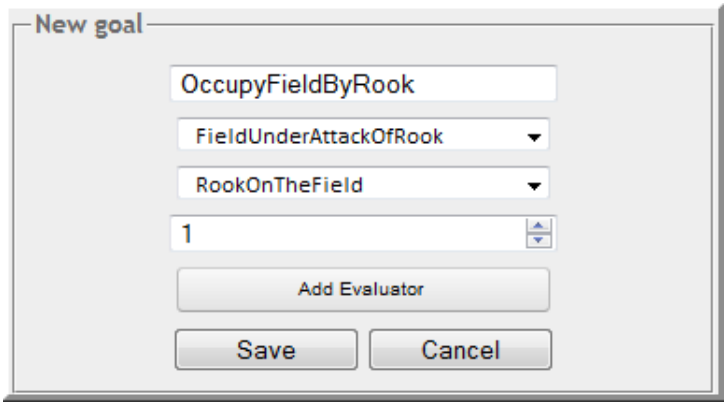


Fig. 25. Interface to define goals

example only one criterion exists and it is the distance between two kings.

From the described above we reveal that the goal consists of preCondition and postCondition, which are situations (in the Solver we define these situations as composite abstracts), depth of three, which is a number that defines how deep the tree can be constructed for checking if the postCondition is achieved (by default it is 1) and the evaluator which evaluates how good the goal is achieved.

Also one important point we need to define the concept of absolute goal (which is just a flag on the goal), like mate in chess and indicates that the game is over.

Plans are goals chained by their priorities. In the interface users need to add the goal and

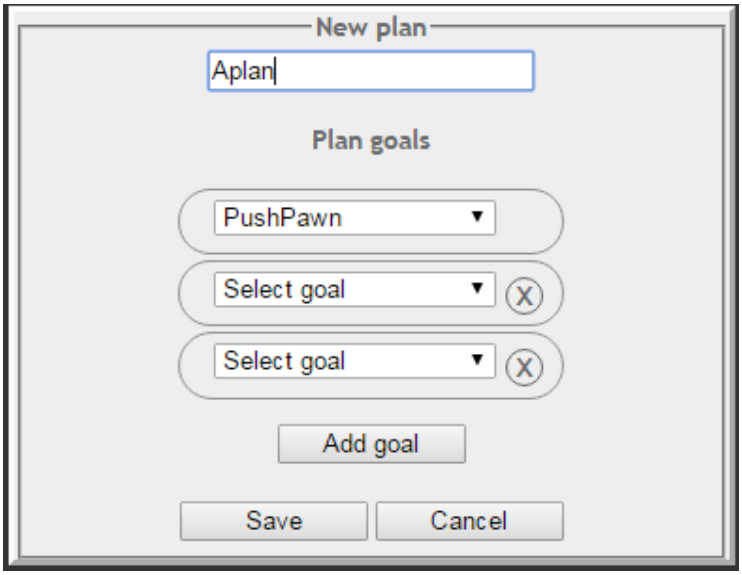



Fig. 26. Plan's structure

indicate its priority in the plan they belong to or the list of existing ones. Plans can include only one goal, e.g., the final one, or can include several goals in solving given sub-problems.

### 2.2.3

Thus in the above section the following results are revealed: RGT Solver provides structures for acquisition of RGT problems, strategy-related knowledge, particularly provided enhancements in the structures and representation of RGT problems enables definition and processing of the given problems. The ability of definition provided before was incomplete and had shortcomings which are fixed here:

1. Handling of knowledge negation is integrated into the abstracts graph, where each negated knowledge is integrated as a new node in the graph with the parent of the same concept with positive (not negated) value.

2. Definition of continuous Sets intensively used in RGT problems, particularly in chess are integrated providing ability to avoid definition and processing of regular Sets instead. Processing of regular Sets is not effective and requires exponential time period.
3. Indication of side for the actions defined for RGT problems. Actions side indication was not solved previously which also lead to disambiguation when searching for possible actions for a side in the RGT game.

Other than described above enhancements we provided also structures for definition of strategies in the Solver. The structures used in PPIT algorithms allow us to define goals and plans, where goals define certain utilities to achieve in a situation and plans appear as list of goals chained by their priorities.

## 2.3 PROVING ADEQUACY OF KNOWLEDGE REPRESENTATION IN RGT PROBLEMS

The representation structures and model as well as enhanmcents are required to be experimented for RGT problems and adequacy proof is needed. This section is devoted to the experiments of proving adequacy of the developed model of knowledge presentation and processing.

Particularly, we detail the procedure of building new nodes and connections within the Graph of Abstracts during the acquisition and analyze the growth of the graph for three different problem instances of RGT class, namely, chess, management and intrusion protection.

### 2.3.1 PRESENTATION OF CHESS CONCEPTS

For chess experiments we consider the modeling of Rook Abilities concepts from [15]. Although it is a relatively simple concept in chess, it transparently demonstrates the presentation of knowledge. Its modeling leads to a construction of various types of nodes and relations as well as underlines the optimizations developed for the model.

Particularly, the definition of Rook Abilities requires the modeling of both specialized and generalized abstracts. For example, how to model the of "rook move" abstract which would include all possible moves of a rook? - this is basically the part of Rook Abilities concept. The other difficulty is the definition of a "group" meaning. For example the definition of empty fields on the same line. This is simple by its specification, but it has a specific structure when representing its meaning as well as when matching it in situations.

As for every chess concept, we need to have defined nucleus concepts, the basics of the game. As discussed in [22] the basics are chess board, figures and the sides (white, black players). We will skip some of the definition details due simplicity the limits of the paper.

The board can be defined as a composition of fields, where each field has its coordinates. The figure has color and type, which identifies whether this is a pawn, a knight, or some other type of figure.

It is important to note that all these nucleus abstracts will formulate the entry nodes or roots of the graph of abstracts.

They basically play two important roles:

- they serve as connectors between the knowledge and the situations,
- they serve as nucleus units for construction of other abstracts.

Clarifying the abstracts and the roles of nucleus abstracts as the next step we define the Field abstract by simple composition of these nucleus ones.

As the value ranges are not specialized for Field, only have connections to the existing nucleus nodes are built. Therefore, the relations of Filed can be presented as {be=""; have={fc:FigureColor, ft:FigureType, cordx:X, cordy:Y}; do=""}. We define Figure as a Field, which has specialized color and type. which leads to creation of two new nodes: Figre.fc (figure color) and Figure.ft (figure type). Figure will have the following relations: {be="Field"; have= {fc:Figure.fc, ft:Figure.ft, cordx:X, cordy:Y}; do=""}.

Other figures are defined like Figure. For example Rook is derived from Figure, and specifies new value of Figure.ft attribute (=4). Other values are not changed. So its integration is similar to the integration of Figure.

To define the more complex units of meanings, we define a set called EmptyInterval. Variations of this set (the specializations) can represent different successive empty fields between given positions. It has an element of Field type with EmptyInterval.element.ft=0 and minCount=1, maxCount=8. To create EmptyInterval first the algorithm creates EmptyInterval.element.ft abstract with a value = 0, and adds a be relation to FigureType, afterwards, it creates EmptyInterval.element abstract, adding a be relation to Field and have relations to X, Y, FigureColor and EmptyInterval.element.ft nodes. Once EmptyInterval's attribute is created the algorithm creates have connection from EmptyInterval to EmptyInterval.element. This abstract in fact can contain any combination of empty fields where number of Fields is 1 to 8. Therefore it can be described by the following: EmptyInterval {be:""; have: {element:Field}; do: ""; minCount=1; maxCount=8}. EmptyInterval is defined as a continuous Set.

Continuing the definitions we get to a virtual abstract called FieldsBetween. The meaning of this abstract is to describe a Figure and a Filed on either the same vertical or on the same horizontal, while in either case there shall be only empty fields (or no field) between them. Consequently, it is defined as a composition of a Figure and a Field abstract which X and Y coordinates are undefined. As this is a virtual abstract, it doesn't create any have relation to the existing abstracts (it interacts with the rest of nodes with its specifications and usages). There are 4 specifications of FieldsBetween abstract which include EmptyInterval abstract and 4 others which do not (in the case of the latter, Field and Figure are neighbors). The difference between specifications stands within their coordinates: 1) Fig.X=Fld.X and Fig.Y>Fld.Y, 2) Fig.X=Fld.X and Fig.Y<Fld.Y, 3) Fig.X>Fld.X and Fig.Y=Fld.Y, 4) Fig.X<Fld.X and Fig.Y=Fld.Y.

We will discuss only the first case, while the others can be specified similarly.

We define FieldsBetween1 in the following way. Obviously it is derived from FieldsBetween virtual abstract. Here Field has already specified values for X and Y according to the above description of this case, Field.X=Figure.X and Field.Y<Figure.Y. EmptyInterval also defines new dependencies here: EmptyInterval.element.X=Figure.X, EmptyInterval.element.Y=IN[Field.Y+1, Figure.Y-1] (this rule shows that Y attribute range is between Figure and Field). Moreover, we can add one more specification here, EmptyInterval.element.ft=[4,5], which indicates that the concept is

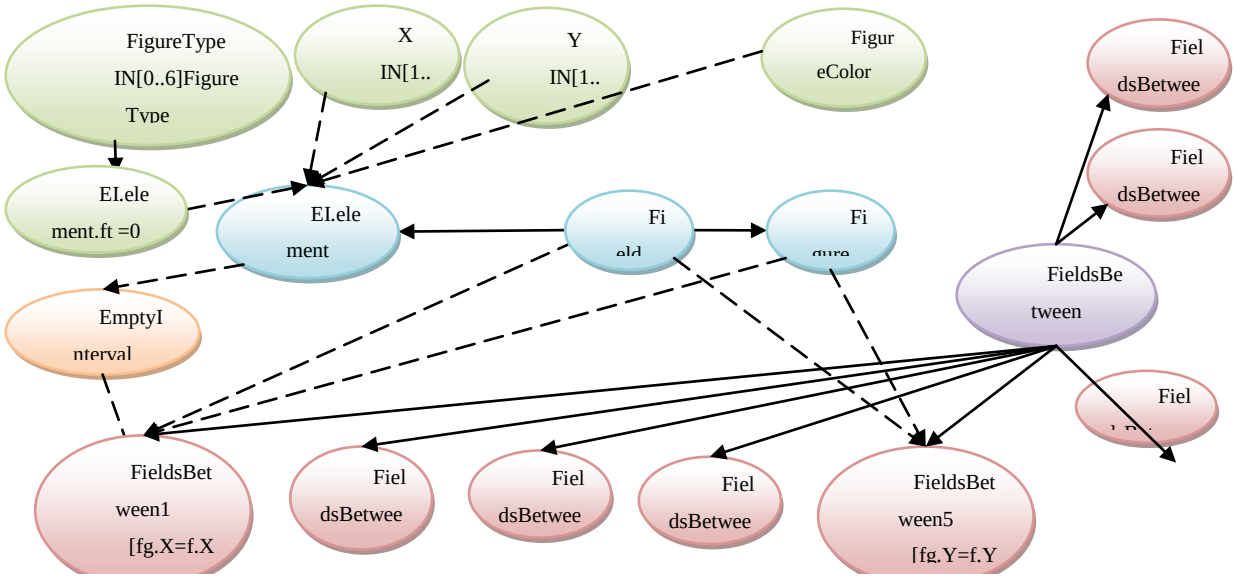



Fig. 27. The representation of FieldsBetween and helper concepts in the Graph of Abstracts. Some of the nodes and relations are removed from this picture to simplify the view. Solid lines represent be and dashed lines have connections.

applicable only for Rook and Queen. EmptyInterval.minCount=Figure.Y–Field.Y-1 and EmptyInterval.maxCount=Figure.Y–Field.Y-1, where the last two specify the number of fields between Figure and Field. During the integration into the GA the algorithm first creates be relation

to the FieldsBetween virtual node, then have relation to Figure and Field nodes and EmptyInterval. The new dependencies between Figure's, Field's and EmptyInterval's attributes are kept in the FieldsBetween1 node. Consequently, the abstract will be represented by the following: FieldsBetween1 is {be: FieldsBetween; have: {fg:Figure, f:Field, ei:EmptyInterval}; do:""; f.X=fg.X; f.Y<fg.Y; ei.element.X=fg.X, ei.element.Y IN[f.Y+1, fg.Y-1], ei.minCount=fg.Y–f.Y-1, ei.maxCount=fg.Y–f.Y - 1}.

The differences of the other four specifications (which do not include EmptyInterval abstract) are as follows: 1) Fig.X=Fld.X and Fig.Y=Fld.Y+1, 2) Fig.X=Fld.X and Fig.Y=Fld.Y-1, 3) Fig.X=Fld.X+1 and Fig.Y=Fld.Y, 4) Fig.X=Fld.X-1 and Fig.Y=Fld.Y.

Thus, FieldsBetween6 will look like this: {be: FieldsBetween; have: {fg:Figure, f:Field}; do: ""; fg.X = f.X, fg.Y = f.Y - 1}.

After above definitions the graph of abstracts will have the structure represented in Fig. 27.

RookAbilities can be defined as an abstracthaving a single FieldsBetween attribute and specifying the type of FieldsBetween.Field.ft=4 (rook). It will have the following representation: {be:""; have:{fb:FieldsBetween}; do:""; fb.fd.ft=4}.

Similar to the RookAbilities we can define the BishopAbilities concept. The latter will require having a concept FieldsBetweenDiagonal similar to FiledsBetween, however, here the fields shall be on the same diagonal. The latter can be achieved by defining EmptyIntervalOnDiagonal concept analogical to EmptyInterval.

The advantage of such approach is that the defined units can be reused extensively. For example, once having BishopAbilities and RookAbilities defined we can define QueenAbilities concept just through three simple concepts. We will need to define a QueenAbiities virtual abstract, then define two specifications, where one of them will use FieldsBetween and the other FieldsBetweenDiagonal abstract.

### 2.3.2 PRESENTATION OF KNOWLEDGE IN MANAGEMENT

As a management problem we consider the Supply Chain Management (SCM) problem, which is concerned with planning and coordinating the activities of organizations across the supply chain, from raw material procurement to finished goods delivery [88].

The main nucleus abstracts for TAC are: Date, Price, Quantity, Speed, ComponentType, ComponentQuality, Brands, Factory capacity, Factory utilization, Bank account, etc. More complex (composite) entities are: Assembled product, Particular supply component, Bank, Factory, Agent,

Customer, Supplier, etc [89] [88]. This type of knowledge is represented in Graph of Abstracts exactly like we described for chess.

However, a certain interest represents the specification of Moves and Strategy Plans in TAC SCM. Strategy Plans (SP) for TAC specifies the qualitative changes of some operational parameter(s). In frames of TAC problem, we can separate two concepts from each other: Move and Strategy Plan (SP). We call as "Move" a statement which contains entities with all their parameters specified with exact values and defined applicable action(s) over the given entities (an equivalence to "move knight" in chess). In opposite, we call as "SP" a statement, composed of one move or set of moves, in which at least one parameter within a single move is not specified with its value. Again, a TAC move describes an exact (quantitative) action over TAC entity (entities), whereas an SP describes qualitative action(s). In Graph of Abstracts the formers are defined through Action entities, while the latters are described goals and plans.

In case of move the Agent directly executes the statement.

In case of SP (with missing one or more parameters' values) the Agent may generate several moves from the same SP by substituting the missing values with ones from their acceptable value

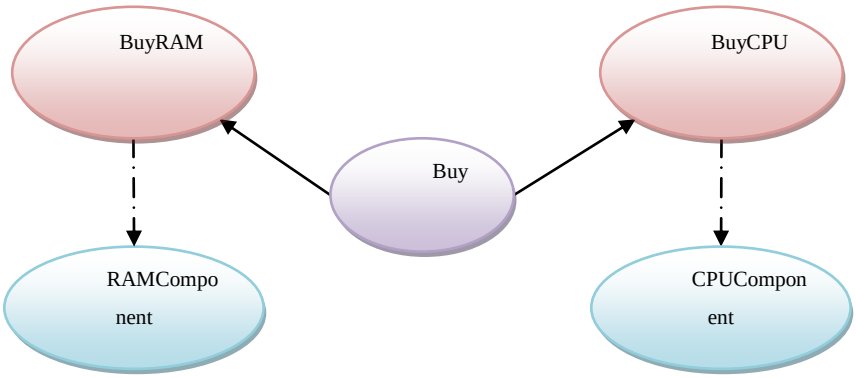



Fig. 28. Presentation of Buy virtual action and its specifications in GA. Solid lines represent be and dot-dashed lines do connections. The specification details of RAMComponent and CPUComponents are not provided to simplify the view.

range.

A typical example of TAC move specified with a goal would be the following statement: "Buy N quantity of Queenmax 2GB RAM by D date". Here we have 3 predefined entities: "Quantity", "Queenmax 2GB RAM" supply component and "Date".

In this example there is an action – "Buy". Here in TAC we should describe action objects with further addition of their implementations into each entity it may be applied to. In the case above the "Buy" action should be described in general and its appropriate implementation should exist for "Queennmax 2GB RAM" entity. That action may also take "Quality" and "Date" entities as

mandatory (the first) or optional (the second) parameters. If no “Buy” action is defined for “Queennmax 2GB RAM” entity then that statement loses its meaning and that move or SP will take no effect. “Buy” can, as well, be defined for other type of entities, like "Pintel CPU 3Ghz".

### 2.3.3 PRESENTATION OF KNOWLEDGE IN INTRUSION PROTECTION

The knowledge for Intrusion Protection problem as defined in [4] [6] is represented in the form of goals and rules. Both of them are easily represented within our model. Particularly, the goal states are defined by the combination of certain important variables: the processor time, buffers size, number of incoming packages. These basically compose the input nucleus types, and their combinations are defining the critical and normal states and goals.

### 2.3.4 EXPERIMENTS WITH MATCHING ALGORITHM

Once we have the GA constructed, we can match situations to the GA and find activated concepts by the matching algorithm described in this chapter. Here we consider the activation process and report the results of following two experiments:

a) Measure the following numbers when keeping the situation unchanged and increasing the number of concepts:

- number of checked conditions (NCC)
- number of active partial instances (NAPI)
- number of active instances (NAI)

b) Measure the number of checked conditions when changing the situation (atomic changes) against the same fixed set of concepts.


Fig.29. Left position represents the initial situation and right position presents the situation change

As a static situation we will consider the simple situation described in left position of Fig. Fig.29. and as concepts we will consider the iterative development of RookAbilities concept, thereby, we will consider 3 separate phases: a) only nucleus abstracts and filed/figures, b) adding FieldsBetween with all its specifications, c) adding RookAbilities.

As we described before, the situations are translated into the lists of instances of nucleus abstracts. Thereby, all fields of the chess board are translated into the groups of nucleus instances which completely define its content. For example, the value of the field containing Pawn in left position of Fig. 29. is: { "groupid" : 34, "instances" : [{ "type" : "X", "value" : 2 }, { "type" : "Y" , "value" : 4 }, { "type" : "FigureType", "value" : "1"}, { "type" : "FigureColor", "value" : "1"}]}.

Let's consider the matching of the concepts from the first group. The algorithm iterates over the nucleus instances of the situation and fires the instances to the nucleus nodes of corresponding types. In this case it starts from X instance (value is 1) with groupid = 1. It finds X nucleus type node in GA and checks if the value of the instance satisfies the regulations of the node. As the value is 1, it is fired forward. It also creates partial matches for Field, Figure, Rook and other types of Figures. Next comes Y = 8 instance of the same groupid = 1, it is being passed to Y node, which itself adds its instance in the partial matches of Field, Figure, and all types of Figures, except Pawn (Y=8 doesn't satisfy the rule defined in Pawn), with the same groupid. Similarly FigureType of the same groupid fires instance of FigureType nucleus node and registers its instance to partial match Field with the same groupid. Finally, FigureColor instance is being fired, which makes partial match of Field complete, thereby, leads to firing of an instance of Field forward. Similar procedures are applied to the rest of the fields. The summary is given in Table 1.

Now, let's consider the processing of the second phase concepts. We shall note, that the

| Type | Description | Sum |
|---|---|---|
| NCC | 64x4 (dispatching)+64 (conditions in Pawn.Y node)+64x(1+7) (conditions for FigureType per each figure - including dummy figure)+64x4x(1+1+7) (conditions to create instances in field/figures). | 3136 |
| NAI | 64 (Field) + 3 (Figure) + 61 (DummyFigure) + 1 (Pawn) + 1 (Rook) + 1 (Knight) + 64x4 (all nucleus instances) | 387 |
| NAPI | 61 (Figure) + 3 (DummyFigure) + 63 (Pawn) + 63 (Knight) + 63 (Rook) + 3x64 (the rest of figures) | 509 |

Table 1. The number of checked conditions, active instances and partial active instances for phase one concepts.

matching algorithm works in depth first fashion, thus, once the nodes are activated they are fired further and lead to partial/complete matches of successive nodes.

The activation of FieldsBetween is triggered by the activation of one of its specifications. Let's first discuss the activation of FieldsBetween5... FieldsBetween8 nodes (this are a bit simpler than the first four). As was described before, they do not add any additional nodes but rather are connected to Field and Figure nodes. Thereby, the instances fired from Field or Figure nodes are directly captured by them. Hence, the number of input instances is only 64+3=67.

Each of the instances leads to a creation of a partial instance and all possible combinations are considered.

On the other hand, the first four specifications use also an additional concept: EmptyInterval, we shall note that the Set nodes are not activated as free nodes rather they are activated in a backward chaining fashion - once the conditions of the Set's attributes are evaluated. This helps to dramatically decrease the explosive number of instances created by a Set node. The analysis is given in Table 2.

| Type | Description | Sum |
|---|---|---|
| NCC | 3x64x8 (all pairs of input instances for each of 8 specifications) + (5+5)x64 (per each satisfying Figure +Field combination being on the same line, we group the satisfying fields in EmptyInteraval concept) | 2176 |
| NAI | 4 (the neighbors of Rook) +1 + 2(other specification with fields between) | 7 |
| NAPI | 4+3 (the Rook with Fields where EmptyInterval's conditions are not satisfied) | 7 |

Table 3. The number of checked conditions, active instances and partial active instances for phase two concepts.

The last analysis of this experiment is to consider the activation of phase three concepts given in Table 3. Here the Rook abilities works as a filtering node.

| Type | Description | Sum |
|---|---|---|
| NCC | 7 (only fired instances of FieldsBetween node are checked) | 7 |
| NAI | 7 | 7 |
| NAPI | 0 | 0 |

Table 2. The number of checked conditions, active instances and partial active instances for phase three concepts.

What follows from the experiment, is that each successive layer reuses the achievements of the previous ones.

In our next experiment we will consider an atomic change of the initial situation given in Fig. 29, and will report the number of condition checks triggered by the change. The aim of the experiment is to show that the subsecutive situations lead to only a few new condition checks.

In the given situation only 4 nucleus instances are changed, they are the FieldColor and FieldType of 34th and 26th groups. All partial and active instances using these nucleus ones are deactivated. The re-activation analysis is given in Table 4.

| Type | Description | Sum |
|---|---|---|
| NCC | 4 (nucleus instances) + 4x(1+1+7) (conditions to create instances in field, figures) + 2x64 (for checking one of the modified specifications of FieldsBetween) + 2 (filtering conditions in RookAbilities) | 170 |
| NAI | 1 (Pawn) + 1(DummyFigure) + 2 (Field) + 2 (FieldsBetween) + 2 (RookAbilities) | 8 |
| NAPI | 1 (Pawn) + 1 (DummyFigure) + 5 (other figures) + 3 (FieldsBetween) | 10 |

Table 4. The total number of checked conditions, active instances and partial active instances triggered by the situation change.

## 2.4 SUMMARY OF CHAPTER 2

In the following chapter two main problems were considered:

1. The extension of knowledge presentation by modeling graph of abstracts and different types of nodes in that, as well as enhancements in the presentation, including negated concepts, continuous sets and side indicators. Algorithms for matching situations to the abstracts are considered too.
2. Structures for defition of goals, plans are developed, where the structures are generic and allow definition of any RGT problem goals, plans.
3. The proof of adequacy of abstract representation structures as well as matching algorithms by expert knowledge pieces of RGT problems, particularly by chess and SCM. The experiments let us state that developed structures provide ability of acquisition of RGT problems in general and match situations to them.

# CHAPTER 3. STUCTURE, PERFORMANCE AND ADEQUACY OF PERSONALIZED PLANNING AND INTEGRATED TESTING ALGORITHMS

## ABSTRACT OF CHAPTER 3

The basic idea of Personalized Planning and Integrated Testing (PPIT) algorithms is given in Chapter 1. The algorithms use personalized expert knowledge for solving problems of RGT class.

The initial implementation of the algorithms [18] used knowledge units that were hardcoded as C++ language classes. This approach didn't allow adding expert knowledge in a regular way – there was not any regular method for formalization and representation of the expert knowledge, and developed algorithms were not generic for regular expert knowledge processing and execution.

Future developed structures and algorithms as well as enhancements of the knowledge representation given in Chapter 2 of this work were intended to develop dynamic and regular formalization and representation of expert knowledge into RGT Solver.

The generic algorithms for personalized knowledge-based strategy search were not yet developed.

In the following chapter two main problems will be discussed:

1. Structures and performance of PPIT algorithms which are able to solve any of RGT class problem.
2. The adequacy of PPIT algorithms on well-known chess etude of Reti and rook endgames, spreading the results on the whole class of RGT problems.

Hereinafter PPIT algorithm essence and initial implemented version is discussed, shortcomings are underlined followed by the detailed description of PPIT algorithms structure and performance. Furthermore proof of adequacy of PPIT in chess by application on several chess situations is brought.

## 3.1 ONGOING STRUCTURE AND PERFORMANCE OF PPIT ALGORITHMS

The PPIT1 program [18] was designed as a composition of the following basic units:

- Reducing Hopeless Plans (RHP)
- Choosing Plans with Max Utility (CPMU)
- Generating Moves by a Plan (GMP)

PPIT algorithms provide ability for searching strategies using personalized expert knowledge. Developed RGT Solver package already provides ability to acquire personalized expert knowledge as discussed in previous sections.

GMP module in fact represents the strategy search algorithm based on the given plan, where plan is selected by procedures applied in CPMU module, which presents the algorithms for choosing the most valuable plan from the list of plans applicable for the current situation. RPH module removing of hopeless plans after retrieval of all plans somehow related to the given situation. In the following we aim to construct algorithms that do strategy search by plans defined by users, thus currently we stay in the scope of CPMU and GMP modules of initial implementation of PPIT algorithms.

Improvements indicated in Chapter 2 provide ability to define goals and plans, where plans are generic representations of strategies. We develop PPIT algorithms in RGT Solver aimed to search for strategies using provided structures of plans. In this section strategy search algorithm and implementation are discussed in details within the Solver.

### 3.1.1 STRUCTURE OF PPIT

Structures for PPIT algorithms are described in Chapter 2. To construct a strategy the algorithm uses plans. A plan in RGT Solver is a list of prioritized goals.

Goals are composition of several attributes:

a. PPIT algorithms have to construct a sub-tree for each goal, thus it has restriction of goal tree depth,
b. precondition describing situations where goals achievement can be expected,
c. postcondition describing the situations where goals are achieved,
d. evaluator to check the effectiveness of achieving the goal.

PPIT algorithms use structures wrapping plans to select best plan from the given set of plans. Those wrapping structures are basically descriptions of situations where given plans can be good. Since each plan can be applicable to different situations and there are situations where several plans can be good, we provide the following structure for plan wrappers:

- Pattern describing the situations where a plan is good. The pattern in current RGT Solver represents a composite abstracts,
- Reference to the plan applicable for the situation.

The mentioned wrapping structures let us to define correspondence between any situation and plan.

### 3.1.2 PERFORMANCE OF PPIT

Ongoing PPIT algorithms developed in RGT Solver are reinterpretations of previous PPIT algorithms. PPIT developed in RGT Solver provides ability of its execution for any RGT problem with only defining strategy related knowledge in the Solver, such as pieces of knowledge, goals and plans. In this section we demonstrate the algorithm and implementation details in the Solver.

Strategy searching algorithm in RGT Solver is currently implemented by definition of plans and searching for the best action to perform in the given situation by the selected plans. As described in Chapter 2, plan in the Solver is a composition of goals, thus we divide strategy searching algorithm into two main modules:

1. The best actions for given goals.
2. The best actions for given plans.
3. The best plans for given situations

#### 3.1.2.1 THE BEST ACTIONS FOR GIVEN GOALS

The structure of goal identified in Chapter 2 indicates precondition and postcondition used in it, where precondition describes the situations where the given goal is meaningful and can be achieved, while postcondition describes the situations where the given goal is already achieved.

The algorithm of searching for the best action on the given situation for achieving a goal is described below.

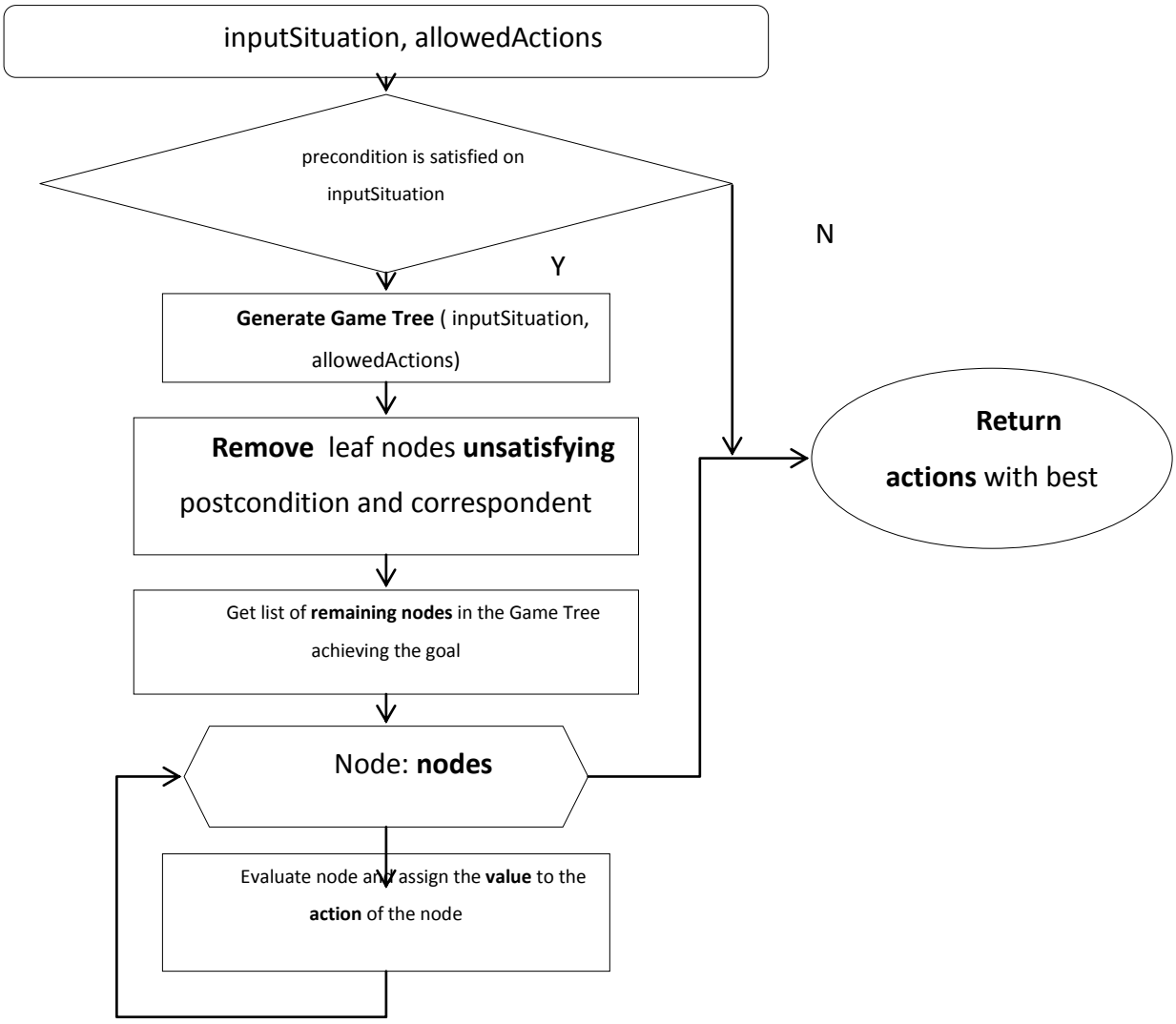



Fig. 30 Block scheme for best action search for the given goal

Best actions searching method receives an input situation and list of allowed actions to perform to achieve the goal.

In the brought above algorithm each block represents an algorithm, which we will discuss in details below.

1. "**Generate Game Tree** (inputSituation, allowedActions)" is an algorithm of generation of game tree for the given set of actions called allowedActions in the given inputSituation. The tree itself is represented as a structure containing root node, acting side on the root node, maximum allowed depth of tree for the goal, reference to list of all the nodes of the tree. Each node itself contains situation representing the node, reference to the tree to which

the node belongs to, depth of the node in the tree processed expert knowledge on the situation of the node, side to act on and active for the side actions on the situation. Also evaluation value for the action of the node is contained in the node. For the root node only allowedActions are applied to generate next nodes. For the rest of nodes all possible actions are checked.

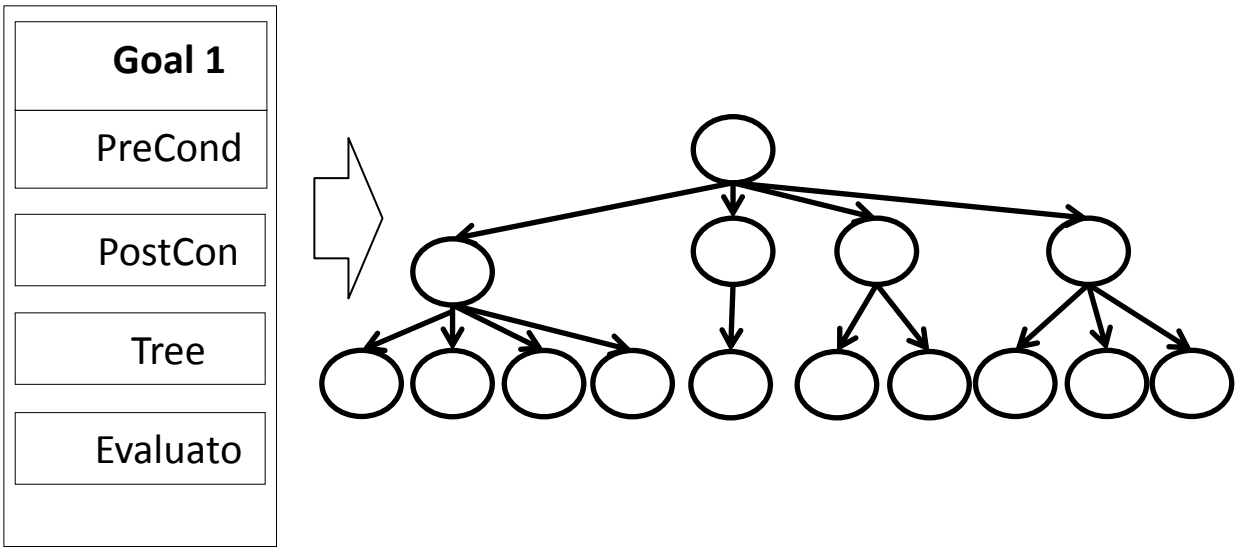



Fig. 31. Tree generation for goal

2. After generation of the game tree **leaf nodes that do not satisfy postcondition** and games directing to those nodes are removed from the tree. The algorithm of finding not satisfying nodes and removing works as shown in the block-scheme below.

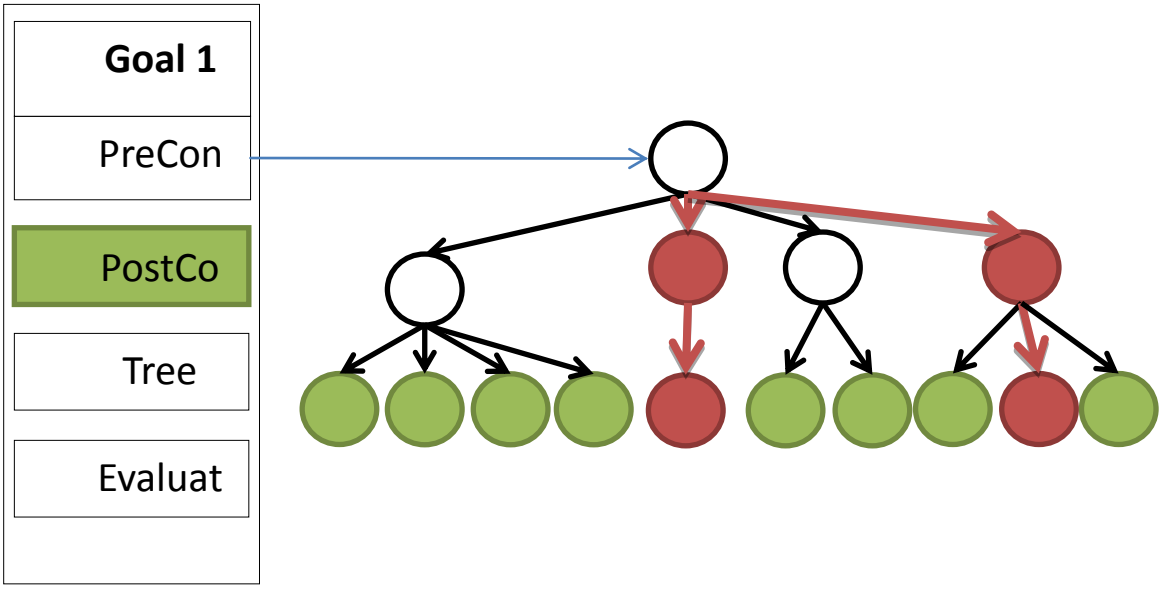



Fig. 32. Removing of nodes not satisfying postcondition

The highlighted in red nodes are the nodes that does not satisfy postcondition of the goal, thus they are being removed with the sub-trees generated from them.

3. For each satisfying leaf node **evaluator is processed** and list of values evaluated by each criterion of the evaluator are added to the map of evaluation value of the node, where evaluation values are mapped to the priorities of their criteria. Comparison of two nodes by their evaluation values is done the by comparing each priority criterion values. Comparison is started from the criterion with the highest priority. If values of current criterion for two nodes are different, then high value is selected for strategy searching side, and the low value for the opponent side. If values of the criterion are equal, next priority values are compared.

The games not having best evaluation values are removed. Remaining nodes are the nodes with the best strategy for the goal. The actions bringing to the best nodes are the best actions, which are returned as suggested for the goal.

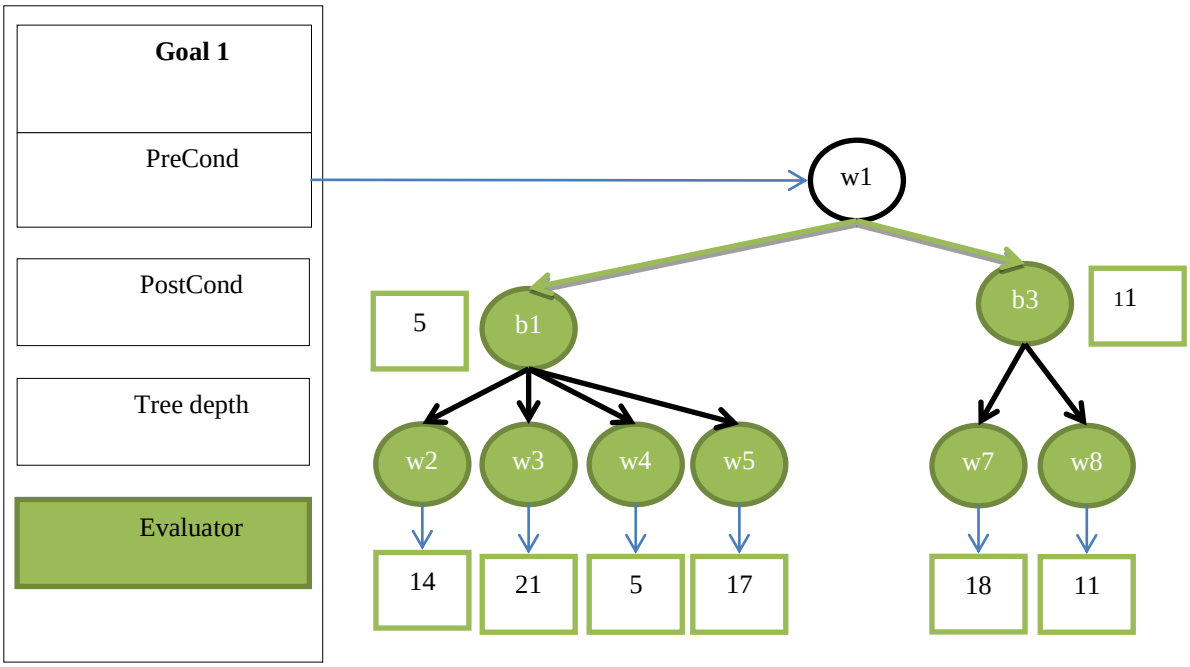



Fig. 33. Evaluation of nodes

### 3.1.2.2 THE BEST ACTIONS FOR GIVEN PLANS

Plan is a certain composition of goals with their priorities. The plan processing triggers processing of component goals. Goals processing is done by the order of their priorities in the plan. If current processed goal is final and an action is suggested to perform, we assume plan is achieved; otherwise actions suggested by the current goal processing method are passed to the next goal for processing.

Plan execution algorithm pseudocode has the following form

As described in the pseudo-code (Fig 34), the algorithm calculates best actions and returns a list

```
ArrayList<ActionInstance> function executePlan(Situation currentSituation, int side) {
Input: Situation currentSituation,
        int side
Output: ArrayList<ActionInstance> currentBestActions
processSituation(currentSituation);
currentBestActions := getActiveActions(side);
for (int i := 1; i <= goals.size(); ++i) {
    Goal goalToExecute := goals[i];
    ArrayList<ActionInstance> tempList := goalToExecute.findBestMoves(currentSituation,
currentBestActions, side);
    If tempList is empty {
        If goalToExecute is primary goal {
            return tempList;
        }
        If tempList.size() = 1 {
            // no need to continue further processing if only one best action is suggested
            return tempList;
        } else {
            currentBestActions := tempList;
        }
    }
}
return currentBestActions;
}
```

Fig. 34. Pseudo code for plan processing algorithm

per each goal. The best action list searching for the given goal is done by the algorithm described in **section 3.1.2.1.** Note that goals are prioritized and processed starting from the goal with the highest

priority in the given plan. For the returned list of best actions of the given goal it is staying unused if the list is empty. In this case currentBestActions list is not changed, thus it continues processing goals and searching for best actions. If the list contains 1 best action, then the action is selected and processing is finished. This means that no need of further processing when there is only 1 action to perform for achieving the goal. currrentBestActions list represents the list of best actions after the end of processing.

### 3.1.2.3 THE BEST PLANS FOR GIVEN SITUATIONS

To find the best plan in the current state of PPIT we use defined set of plans. Plans are defined by the structures described in section 3.2.2.2, where each plan is composed of goals.

Structures being developed need to provide ability to define mappings between different situations and plans. The research process has shown that:

1. There may be plans useful for different situations.
2. There may be several plans useful for the given situations.

To provide ability of definition of several plans useful for the given situation and several situations applicable for the given plan we define a new structure plan wrappers.

PlanWrapper structure consists of two attributes: a) situations, where the given plan is applicable; b) plans to be used.

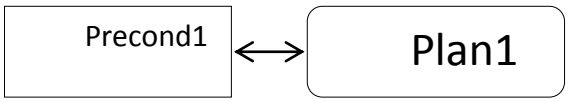



Fig. 35. PlanWrapper structure

Situations applicable to plans are defined in patterns called preconditions similar to goals. Preconditions are composite abstracts which describe the situations.

Plans used in the wrapper are references to already defined plans, where plans are defined in structures described in the previous section.

The wrapper structure also contains evaluator with criteria which can be defined to describe how good the plan is for the given situation, the evaluator is just like the one defined for goals and calculated values are related to the precondition of the wrapper. The evaluator can be undefined too.

The execution of the algorithm has to solve several problems.

1. How to find the plans good for the given situation?
2. How to select the plan which will be used for strategy construction?

Here the algorithm of the best plan selection for the given situation is discussed, as well as solutions to the mentioned questions are described.

The algorithm gets situation as an input.

For the first step the algorithm processes the given situation and iterates over all of the plan wrapper structures. Each of wrapper preconditions are searched in the situation and if found, added to the list of wrappers - matchedPlans.

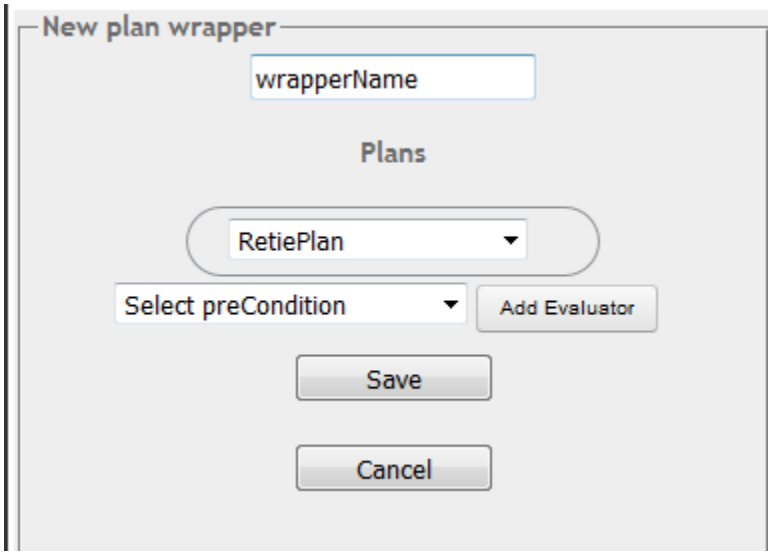



Fig 36. interface for definition of planWrapper

MatchedPlans list represent all the plans which are applicable for the given situation, thus this contains good plans. Now the algorithm needs to select the best plan from the given list. First it calculates all the evaluators, if there are and sorts by the best evaluator values. From the given set it selects the best evaluation values.

Note that plan evaluation criteria should be similar in wrappers containing similar wrappers, because otherwise the algorithm will not be able to make correct comparison between two plans.

If there are wrappers in the list that has no evaluator, then the algorithm calculates the number of appearances of plans appearing in the situations. E.g. a plan P1 appears with 5 different preconditions and P2 with 2 preconditions. The plan with the most number of appearances is selected.

From the given list of most appearing plans and best evaluation value plans a plan is selected randomly.

The current implementation of requires intensive usage of expert knowledge. The more detailed and deep levels of expert knowledge provide ability to define more correct plan selection wrappers.

Anyway the insertion of models of planWrappers provide ability to make the presence of expert less required for searching strategies. All the situation processing requires is defining the situation and providing the side to act in the given situation. The algorithm will be able to select the best plan from the defined set of plans by their wrapping models, and then find best action for the given plan. The best action is being provided to the interface.

### 3.1.3

As a result we have:

1. Restrictions of knowledge acquisition in initial realization of PPIT algorithms made it less flexible and not general for RGT problems, which lead to the current realization of PPIT algorithms within RGT Solver package enabling regular knowledge acquisition and generic strategy search methods for the algorithms use in any RGT problem.
2. Ongoing PPIT algorithms developed in the RGT Solver are certain reinterpretations of previously developed PPIT algorithms, where strategies are presented by plans and goals.
   - The best actions searching algorithm for the given goals is developed. The algorithm generates game trees for the goals and searches for actions to achieve the goal using expert knowledge.
   - The best actions search algorithm for the given plans is designed and implemented, where best actions list is revealed by sequential process of goals, where each goal has a certain priority in the plans.
   - The best plans selection algorithm for the given situations is designed and implemented using structures of wrapping plans to have mapping of plans and situations corresponding to them.
   - RGT Solver strategy searching algorithm is generic and is not restricted to any specific problem, this provides ability of solving RGT problems.

## 3.2 ADEQUACY OF PPIT

Developed PPIT algorithms require effective testing to prove viability. We prepare and provide various chess experiments of these algorithms in RGT Solver. Achievements in a kernel K problem of RGT problems space can be spread on the whole space of RGT, thus results of experiments of certain RGT problem, particularly chess, are generic and not restricted only to the given problem. Experimenting chess as a kernel problem of RGT class lets us spread our experiment results on the other problems of RGT space.

The following situations were formulated for PPIT adequacy of chess

- Rook endgames: rook and a king against opponent king (side of rook and king puts mate), two rooks and a king against opponent king (again the side which has advantage puts mate)
- Reti etude, suggested by Botvinnik as a knowledge requiring etude, which was solved by previous implementation of PPIT.

### 3.2.1 ADEQUACY BY ROOK ENDGAMES

For the demonstration of our algorithms we consider chess endgames, like “rook against king” or “two rooks against king”.

We will try to define only the mate on one direction to make to simply the planning algorithm, similar to the algorithm of strategy description described in [90]. Let’s take vertical direction only for our future definitions. Similarly we will be able to define putting mate on horizontal direction. The algorithm of best plan selection will choose the plan to play with (verial or horizontal direction).

A plan for the “rook against king” endgame will have the below goals:

1. Put mate
2. Avoid stalemate (note that this is quite important because some situations can appear with stalemate and we need to avoid it)
3. Escape rook from attack
4. Push king to the edge (without putting rook under attack)
5. Make a waiting move when preOpposition appears
6. Bring white king closer to the black king

The definition of each goal is described in details.

1. Putting mate - preCondition is any situation, and postCondition is a situation where there’s mate, the depth is 1, this is absolute goal. There is no evaluator defined for this goal.
2. Avoid stalemate - preCondition is again any situation and the postCondition is a situation where no stalemate appears. The depth is 1 and no evaluator again.
3. Escape rook from attack - the preCondition is “rook under kings attack” abstract, so the goal is applicable only for situations where the rook is under the opponent king’s attack. The postCondition is a situation where rook is not under attack and the vertical coordinate of the rook is not changed. It has a depth value 1 and the evaluator will have

one criterion defined which calculates the distance of the rook and opponent king by vertical direction.

4. Push king to the edge- preCondition can be any situation and postCondition is “rook is not under attack” situation and depth is 2. The evaluator has two criteria. First is: moves of opponent king are closer to the edge are better (this basically means the horizontal distance of opponent king from the edge is calculated and for each action the value of criterion is calculated as the highest value of king’s distance from the edge). The second criterion for this goal evaluator is the number of actions opponent king can do, and the better action is the action which allows fewer number of actions by opponent king.
5. Make a waiting move when preOpposition appears - preCondition is preOpposition situation. PreOppositionByVertical abstract in the Solver can be defined as below. This is a virtual abstract which has two attributes – black and white kings. It must have 4 specifications

A. Whiteking.cordX = BlackKing.cordX + 2
whiteking.cordY = blackking.cordY + 1
B. Whiteking.cordX = BlackKing.cordX + 2
whiteking.cordY = blackking.cordY – 1
C. Whiteking.cordX = BlackKing.cordX - 2
whiteking.cordY = blackking.cordY + 1
D. Whiteking.cordX = BlackKing.cordX - 2
whiteking.cordY = blackking.cordY – 1

which is complete enough to define the precondition of preOpposition. The postCondition is a situation where the king position is not changed and the rook vertical coordinate is not changed. Depth of goal is 1. The evaluator again has one criterion, which shows the distance of the rook from the opponent king.

Bring white king closer to the black king, but avoid opposition – preCondition and is any situation and postCondition is a situation where no opposition appears, depth is 1. The evaluator has one criterion, which defines the distance of the king from the opponent king to be minimal. We can calculate this by the following formula “$(king.cordX-opponentKing.cordX)^2 + (king.cordY-opponentKing.cordY)^2$”.

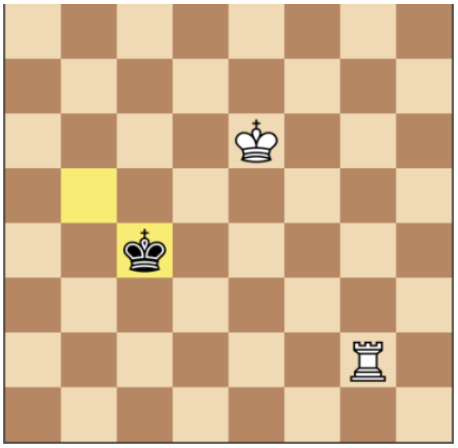


Fig.37. K., R. vs. B.K., An initial position

The algorithm of searching for a strategy with the given above plan looks like this.

1. Algorithm tries to find moves which bring to mate, and returns the empty list.
2. Since the returned list of the 1st goal is empty it takes the initial list of moves and returns the whole list of possible moves since all of them brings to situations where there is no stalemate, so the whole list of moves is passed to the 3rd step
3. "Escape rook from attack" goal is not applicable for this situation, so it just does nothing
4. "Push king to the edge" for all the moves that does not put rook under attack it calculates the first criterion value. Let's see what values it assigns to three of moves.
   a. Rc2… this puts check to the black king, for all king moves it calculates the distance from the vertical edge. King moves can be Kd4, Kd3, Kb4… for Kd4 and Kd3 it assigns will assign the highest value of 4 (the distance from edge is 4). Kb4 will have value 2, so the value assigned to move Rc2 is 4.
   b. Rd2… king can do moves Kc3, Kc5, Kb4… for Kb4 again value as mentioned above is 2, for Kc3 and Kc5 is 3, so the value for Rd2 move is 3.
   c. 1. Rg3… in this case also black king can move to d4 position, so the value will be 4. Similarly all moves other than Rd2 will have 4 value, the minimum value is 3, and only Rd2 has that, so after processing the 4th goal the algorithm will return move Rd2

Since only Rd2 move is returned the algorithm is not processed anymore and this move is applied.

Let's assume black does Kc3 move (attacking rook).

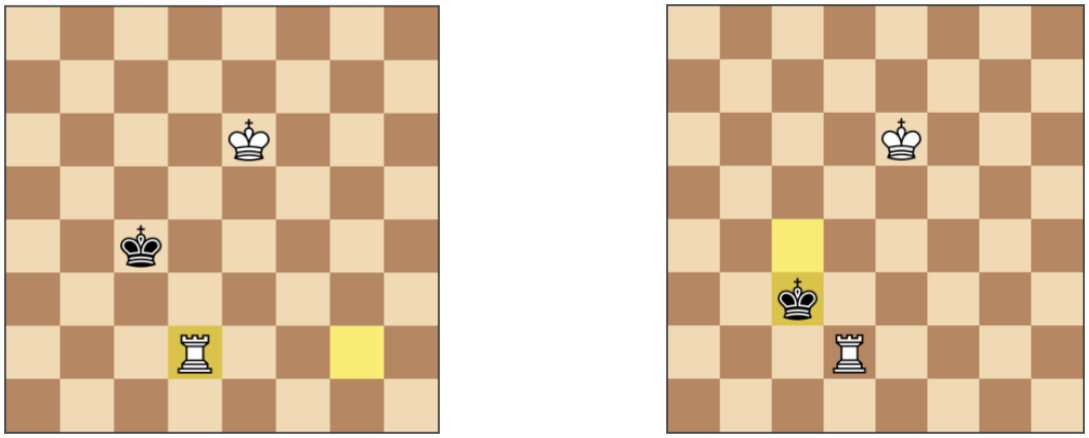


Fig. 38 The left: the position after Rd2. The right: the position after Kc3.

After Kc3 move algorithm works again

1. For mate goal again empty list is returned
2. For stalemate all moves list is returned
3. “Escape rook from attack” goal’s preCondition is matched to the situation and rook moves are considered to achieve the goal where rook is not under black king’s attack since postCondition is “rook not under attack”. The criterion to evaluate the move is vertical distance of rook and black king, so Rd8 move is chosen since it has the highest vertical distance from black king. Since the list has only one move in it, the procedure is stopped here and Rd8 move is returned

Rd8 is applied to the situation. Let’s assume black makes Kc4 move.

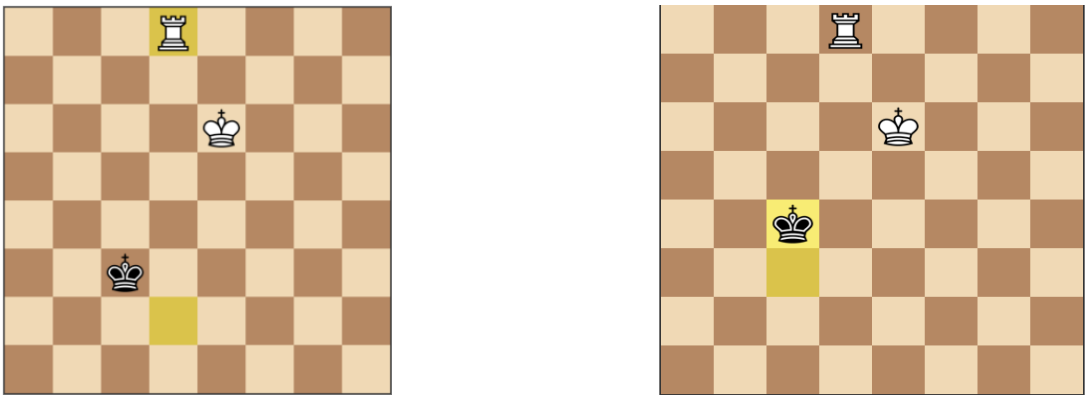
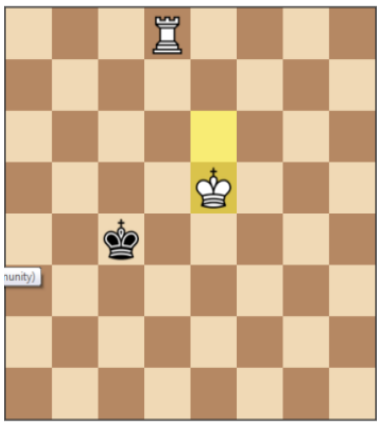


Fig. 39 The left: the position after Rd8. The right: the position after Kc4.

Algorithm works again and now with the following result.

1. For goal mate again empty list is returned
2. For stalemate all moves list is returned
3. No rook under attack so this is just omitted
4. “Push king to the edge” for all the moves where rook is not under attack it checks the evaluator, which have two criteria, the 1st is kings distance from the edge is minimum. So for moves Rd1, Rd2, Rd6, Rd7,Ke7, Ke5, Kf7, Kf6, Kf5 the distance of king from the edge will be calculated as it was done for the 1st move, and the value will be 3, which is selected as the minimum value. Then the second criterion (which is the number of moves opponent king can make) is checked for the moves which are best for criterion 1. Number of moves of black king is always 5 for all the mentioned moves. So the whole list is returned from this goal processing procedure.
5. The situation is not a preOpposition, so preCondition is not matched, this goal is just omitted.
6. “Bringing king closer” preCondition is any situations, and postCondition is a situation where no opposition appears. The list of moves is [Rd1, Rd2, Rd6, Rd7, Ke7, Ke5, Kf7, Kf6, Kf5], which does not bring to opposition, so all of them satisfy postCondition. The evaluator criterion is that distance between two kings needs to be minimum. For the moves by rook distance value will be 8 ((5 -3)2 + (6-4)2). For king moving by f vertical the value will be rising, e.g., after Kf6 criterion returns 13 ((6 -3)2 + (6-4)2). The best move will be Ke5, which will have evaluation value 5 ((5 -3)2 + (5-4)2). Ke5 will be returned.

Since only Ke5 is returned this is applied to the situation. To make the example shorter let’s consider Kd5 move for black.



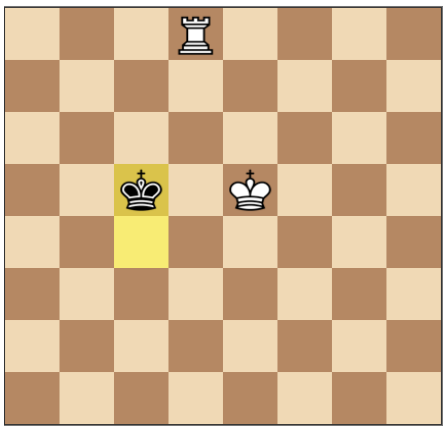


Fig. 40 The left: the position after Ke5. The right: the position after Kc5.

After Kc5 move similar to the 1st move for “push king to the edge goal” Rc8 move will be selected. Again we will assign black king moves which finish the game sooner, we will consider the move Kb4. So after the following moves

1. Rd2 Kc3 2. Rd8 Kc4 3. Ke5 Kc5 4. Rc8 Kb4 5. Kd5 Kb5 6. Rb8 Ka4 7. Kc5 Ka3 8. Kc4 Ka2 9. Kc3 Ka1 10. Kc2 Ka2.

After the 10th move (Ka2 by black) the algorithm will work and find that mate is achievable

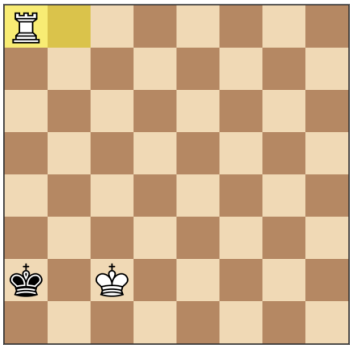


Fig. 41 The position of putting mate.

and Ra8 move will be returned. This move will be applied and the plan is achieved.

### 3.2.2 ADEQUACY BY ETUDE OF RETI

The winning plan of Reti is given in chapter 1. The plan is described as a chain of the folliwng instructions: 1. Hit opponent pawn, 2. Pass the pawn, 3. Protect own pawn. 4 Stay maximally close to pawns. 1 – 4 are goals, which are being embedded into the Solver too.

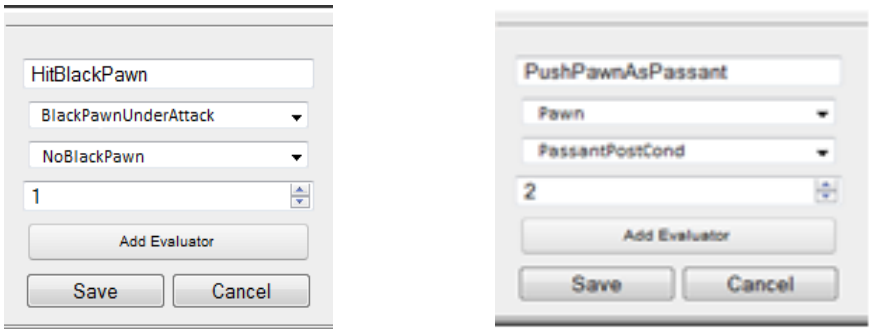



Fig. 42. HitBlackPawn and PushPawnAsPassant goals

HitPawn goal consist of precondition fieldUnderAttack, postcondition NoBlackPawn, and the depth is 1.

PushPawnAsPassant goal postcondition is a abstract consisting of two abstracts PawnPushed, which defines that Pawn coordinate is changed from postcondition situation and PawnNotAttacked abstract is defined as a virtual abstract having two specifications PawnNotUnderAttack and PawnIsProtected.

Similarly other two goals of the given plan are defined. ProtectPawn goal with precondition of Pawn on the board, postcondition of PawnIsProtected, with depth 1 and maxixmal value of "king.y"

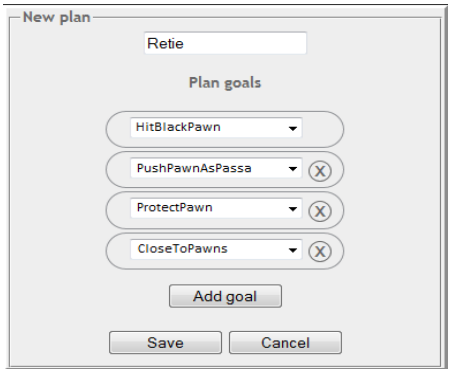



Fig. 43. Reti etude plan

criterion. CloseToPawns goal with precondition and postcondition of any situations and evaluator indicating that the distance between the king and two pawns is minimal.

Reti Plan is defined as a composition of these 4 goals in priorities.

On the given situation Reti plan is selected and processing of the situation is requested.

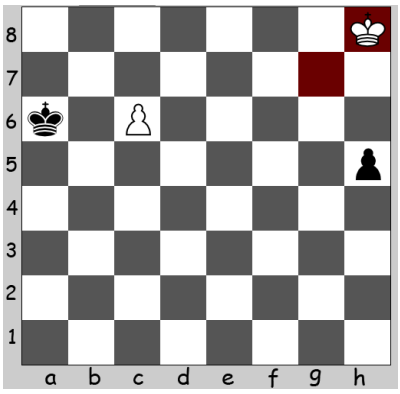
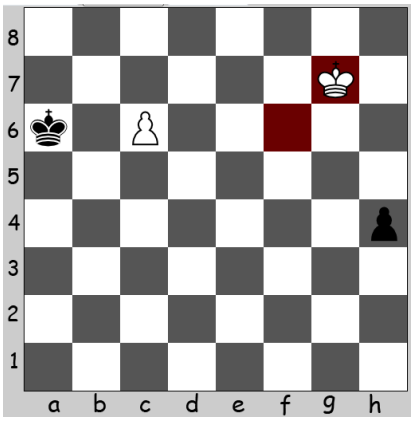


Fig. 44. Initial situation and first suggested action

Here we are going to show how the strategy search is done for a certain chain of moves done by the Solver for Reti etude. The processing algorithm works as follows. For the initial situation the algorithm searches for moves for achieving HitBlackPawn goal. In the given situation the situation does not match the precondition of the goal and thus, goal can't be achieved. The process passes to next priority goal, PushPawnAsPassant. For this goal precondition is matching to the situation, so



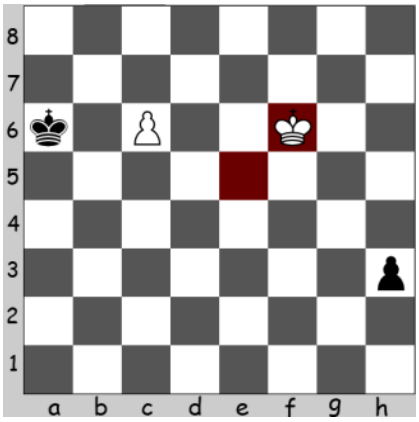


Fig. 45 Next suggested actions

the algorithm generates a game tree with depth of 2, i.e., white and black moves. For all of white moves black has moves that bring to a situation where postcondition of the goal is not matched, after c7 move Kb7 answer by black can be considered which does not satisfy postcondition of the goal, so this goal does not give any action to perform, too. Processing takes next goal ProtectPawn goal. Since there is no situation where pawn is protected, which means no move satisfies postcondition of this goal, too. The last goal to process by the algorithm is CloseToPawns goal

which has no precondition and postcondition and for each of moves it checks for distances between king and two pawns. From all of possible moves by white best value of criterion for minimal distance between king and two pawns is calculated for Kg7. So Solver suggests move Kg7.

Next we will just bring a certain game by black and suggested by the Solver strategy accordingly. After move Kg7, let's assume black plays h4.

Situation after h4 move is shown on the left image of Fig. 45. The given situation is processed in Solver similar to the initial situation and Kf6 move is suggested.

Next black h3 move is considered and situation after that move is on the right image of Fig. 45 Processing is similar to previous two steps and accordingly Ke5 move is suggested.

Let's assume now black plays Kb6 (Fig. 46, left image). HitBlackPawn and PushPawnAsPassant goals are still not achievable. ProtectPawn goal processing finds two moves satistying the postcondition – Kd5 and Kd6. Processing by the evaluator of the goal Kd6 is selected because

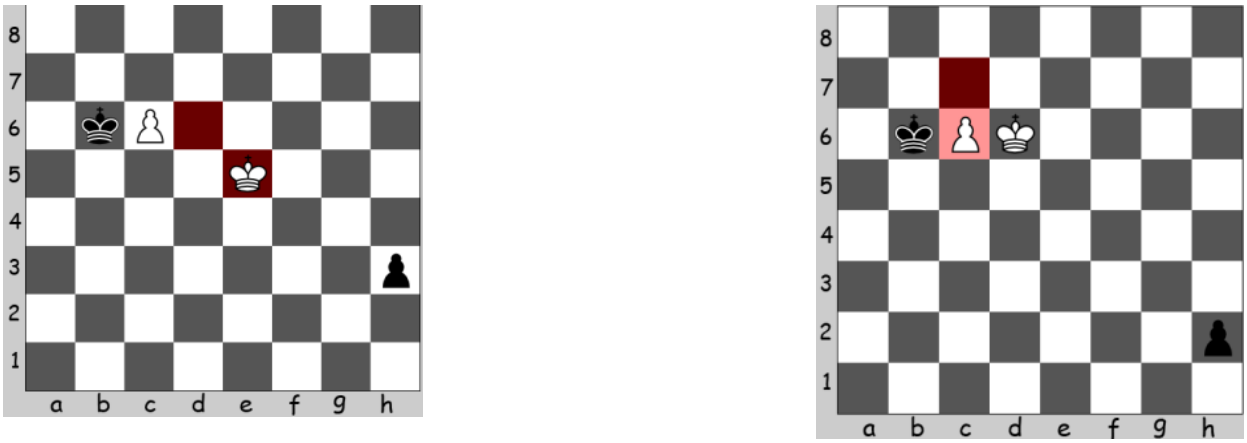



Fig 46. The pseudo code of the knowledge acquisition and integration algorithm.

critrerion "king.y" has maximal value for Kd6 move.

After Kd6 let's assume black plays h2 (Fig. 46, right image). HitBlackPawn goal is not achievable on this situation too. PushPawnAsPassant goal is achievable and c7 is suggested by Solver.

So the game was 1. Kg7, h4 2. Kf6, h3 3. Ke5 Kb6 4. Kd6 h2 5. c7… 1.Kg7, 2. Kf6, 3. Ke5 moves are selected by CloseToPawns goal. 4. Ke6 is selected by ProtectPawn goal which has higher priority and 5. c7 move is selected by PushPawnAsPassant goal.

### 3.2.3

RGT Solver searches for optimal strategies in RGT problems by the problem specification and strategy related personalized knowledge.

1. RGT Solver developed planning interface, functioning strategy searching algorithm and overall RGT Solver package adequacy is tested and proved by chess situations, particularly by rook endgames.
2. Moreover Reti etude pointed out by Botvinnik as an intensive knowledge-based analysis requiring problem was experimented and correct strategy is generated by RGT Solver.
3. RGT Solver provides generic and flexible personalized knowledge representation and acquisition structures and strategy searching algorithms relied on that knowledge. The experiment results indicate that RGT Solver is able to solve intensive knowledge-based RGT problems not restricted to chess problem, but the whole space of RGT class.

## 3.3 SUMMARY OF CHAPTER 3

PPIT algorithms are strategy searching algorithms which use personalized expert knowledge in decision making process, provides efficient solution to RGT problems by knowledge-based analysis.

In the following chapter two main problems were considered:

1. Structure and performance of ongoing PPIT algorithms.
   - PPIT algorithms were developed within RGT Solver by using plans and goals definition structures, as well as plan wrapping structures. Algorithms enhance RGT Solver in searching for optimal strategies by the problem specification and strategy related knowledge, while they are generic to RGT problems space enabling definition and strategy construction for any problem of the class.
   - The algorithm searches for strategies by plans acquired by the Solver using its knowledge representation enhancements.
     i. The best actions search algorithm for the given goals is described.
     ii. The best actions search algorithm for the given plans is described.
     iii. The best plans selection algorithm for the given situations is described.
2. Adequacy of PPIT algorithm is provided in chess, as a kernel problem in RGT problems.
   - Rook endgames and Reti etude were selected for current state of experimenting and successfully passed adequacy experiments. The last one is suggested by Botvinnik as a problem requiring intensive knowledge-based analysis.

The Solver provides with strategies by PPIT in these experiments leading to the expected results. Experimenting results let us state that RGT Solver is able to solve problems of the class not restricted to chess.

# CHAPTER 4. TUTORING BY RGT SOLVER AND STRATEGY SEARCHING ALGORITHMS

# ABSTRACT OF CHAPTER 4

RGT Solver is able to acquire personalized expert knowledge. Other than usage of the acquired knowledge in PPIT algorithms, an important execution can be tutoring for the acquired knowledge, thus we aim to provide means of tutoring RGT problems and knowledge pieces. In the following chapter we discuss tutoring approach on a particular RGT problem chess, while tutoring approach and revealed interaction mechanisms with the interface are expected to be generic to RGT Solver and similar knowledge presentation structures.

The classical chess teaching approach includes teacher and implies interaction between the students and the teacher. This approach has several shortcomings, such as teachers' effective involvement in the teaching process, not personalized. The basic shortcomings of other approaches that do not include teachers are: not interactive; bad organized feedback of learned knowledge understanding; performance of the student can't be checked.

Thus we suggest an interactive personalized tutor package based on RGT Solver and PPIT algorithms. In the following chapter two main issues are discussed:

1. Tutoring method, particularly tutoring algorithms, tutoring protocol developed during the implementation of the package, where we discuss the suggested approach and underline advantages.
2. Adequacy of the developed method, where we provide certain examples of chess endgame tutoring.

## 4.1 METHOD OF TUTORING TO CHESS

We are developing a method and software that is tutoring chess. We interpret the whole chess knowledge in a graph where each piece of chess knowledge is a node. A node represents relations with other chess knowledge pieces and defines regularities that identify that relation, e.g., "opposition" concept represented as a node identifies its relation with "king" node and saying that it contains two instances of king and one of the first king instance coordinates X or Y is different from the second king instance with 2, also kings are of different colors.

The algorithm brings up the name of the concept, relations and regularities, also certain

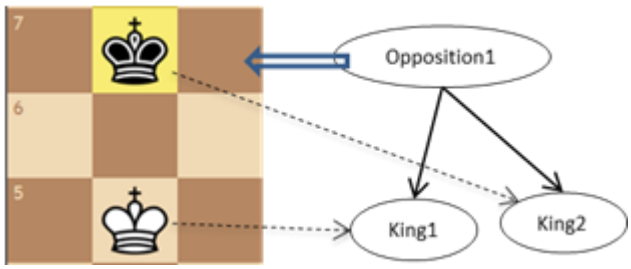



Fig. 47 Opposition chess concept node structure.

examples when tutoring.

The algorithm also has a module which solves a certain chess problem according to the knowledge it has. This module allows comparing the solution of the problem by the student and by the mentioned module for checking the solution result.

The strategy constructing knowledge is described in goals grouped in plans. Strategies are explained using those plans.

For each unit of knowledge examples (chess graph nodes, goals, plans) are added to demonstrate on the board (ideally it will generate situations by itself)

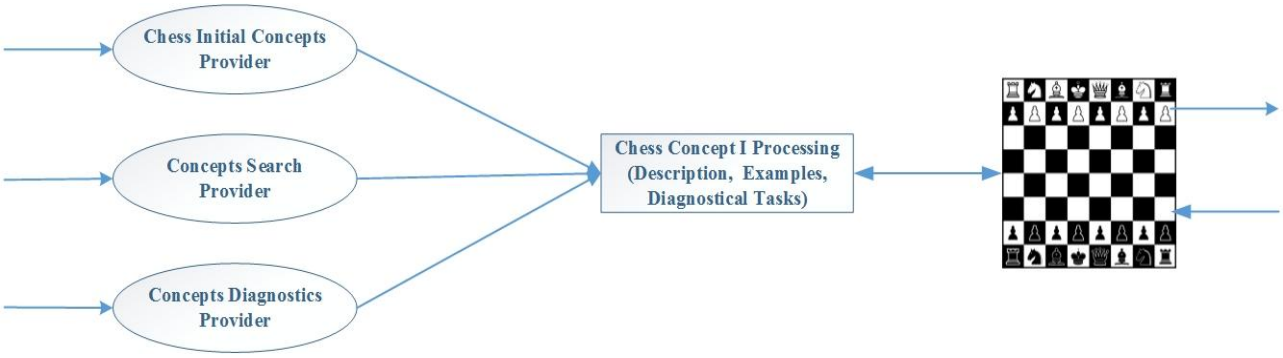



Fig. 48. Chess Concepts Teaching Software Workflow Structure.

The algorithm works like described below:

**i. Tutoring chess concepts**

It is possible to start learning from the beginning or learn a certain chess concept.

a. If the student selects learning from the beginning the algorithm starts tutoring by definition of chess initial concepts which are board (we identify it by X and Y coordinates), each

figure type and its moves, playing sides (black and white), mate, stalemate and other chess rules. Initially the algorithm can start tutoring from the following concepts required by others: 1) X coordinate of the board, 2) Y coordinate of the board, 3) type of each figure 4) playing side colors.

b. If a certain concept learning is requested and this is a piece of knowledge then each relation of the knowledge piece is brought to the student and explained, if the student does not know knowledge pieces related to the tutoring concept, then all those related concepts are explained similarly until the algorithm reaches the concepts which are known by the student, for the tutoring piece also the regularities are defined, e.g., if “pawn” "concept is taught to the student, then it is identified as a related concept to “figure” as pawn is a figure, pawn moves are described, pawn Y coordinate restrictions are indicated. If a plan of a strategy is being taught, then all the required knowledge pieces are taught as described above, then all the related goals and their priorities are taught. If the student is unable to understand the description and regularities of the chess concept after several attempts, then the request is forwarded to the expert for improving the definition of that concept and explanation to the student.

**ii. Testing tutoring**

After teaching, certain chess knowledge testing is done by asking the student to solve chess problems. The student solution is compared to the problem solving module of the software which solves the problem according to the taught concept, too. The results are compared and evaluated in a tool which compares the games, measures and rates the chess players. If the solution doesn’t match the expected result suggested by the chess solving module then the wrong solution is corrected, the correct approach is explained in details. If the student still does not understand the explained plan, goal or another chess concept, then again the concept is explained iteratively as explained in b section of 1st point.

**Implementing Chess Tutors**

We use Reproducible Game Tree (RGT) Solver package and developed external tool for tutoring the student and measuring the performance. Here we will describe how the algorithm is integrated with RGT Solver and Connection tool and will demonstrate the tutoring process on a certain chess endgame tutoring example.

**i. RGT Solver**

In Solver the knowledge pieces are kept in Graph of Abstracts (GA) where each node has 3 types of English language derived relations to other nodes: be-, have- and do- [2]. Personalized Planning and Integrated Testing algorithm is developed for optimal strategy searching, where strategies are described by plans [18].

We assume an expert defined the chess game in Solver and it is ready for execution and playing. Once the chess is brought to Solver it can be also used for tutoring. We usually start definition of chess from X, Y coordinates of the board, figure color and figure type concepts. Those are the nuclear concepts which cannot be parsed in the given way of definition. If an expert wants to make deeper level of definition he will have to define lower level nuclear concepts of chess. Currently we only use the mentioned set of nuclears. Next we define figures as a composition of X, Y, figure color and figure type concepts, each action for figures, also mate, stalemate are being defined and whole the required chess strategy related knowledge. As an example node of chess concept king has the following regularities in it: 1) it is a figure, so king has a connection to figure concept (it is a "be" type of language derived connection), it has X, Y coordinates and figure color just like figure general concept, certain figure type concept value which identifies that this is a king (those are "have" type language derived connections) and king move definitions (those are "do" type connections).

### ii. Explanations in the Graph of Abstracts

The overall tutoring algorithm is brought in Fig. 47. For any concept "X" Solver node brings the relations to other nodes "Y" and "Z", and each of them is explained respectively if needed. Let's consider "Y" is known by the student and "Z" is unknown, then "Z" is parsed and its relations and regularities are brought for explanation, until we get to the nodes which cannot be parsed to more simple ones anymore.

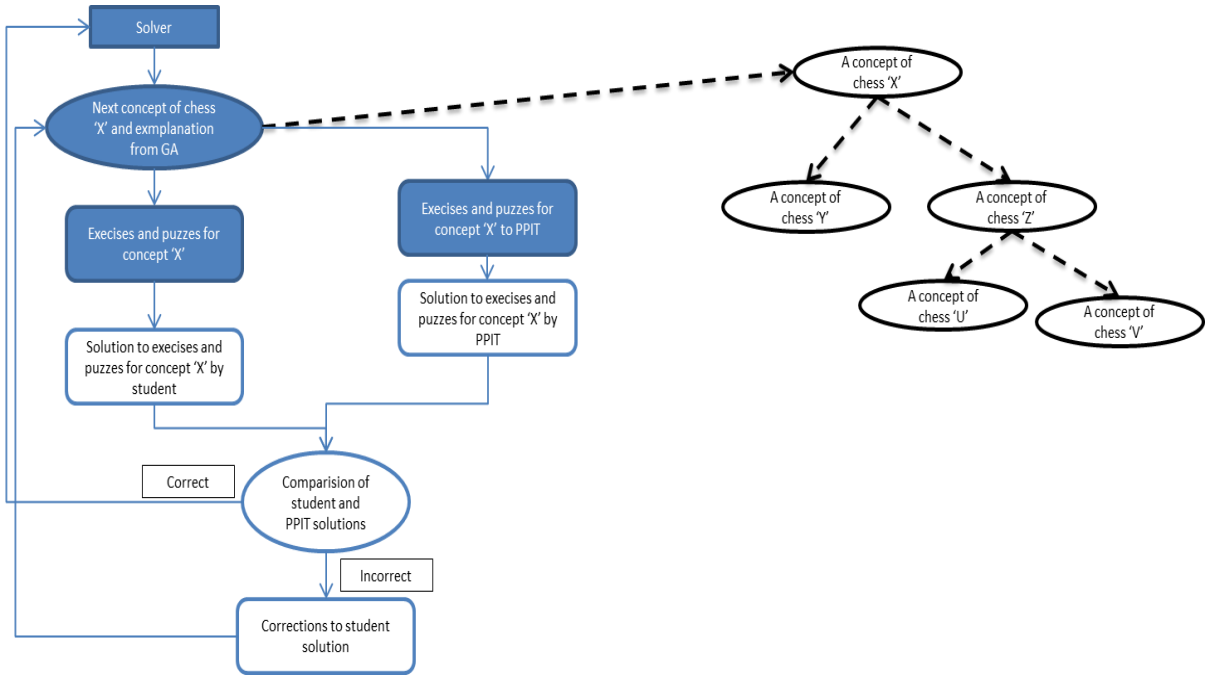



Fig. 49. Algorithm of tutoring for a chess concept.

Generally explanation of any node of GA consists of a) name of the node concept, b) its relations to other concepts (any of "be", "have", "do" relations which are in the definition of that node) and related concept names, c) regularities between related nodes for the certain node, e.g., opposition concept has two kings as instances in it king1, king2 and king1's color is different from king2's color, king1's X or Y coordinate is different from king2's coordinate by 2 and the other coordinate of king1 equals king2's respective one. Those regularities are also brought in the explanation panel of Solver when tutoring, d) certain situations containing the described concept.

### iii. Explanation of strategies

Other than GA nodes strategy searching plans can be also explained to the student. Plan is a set of goals sorted by their priorities, where each goal is a composition of chess concepts definition precondition and postcondition of the goal, the depth of tree to search for the goal from the initial situation and criteria to evaluate how good the goal is achieved. Description of plan is done by explaining each goal of plan and stating its priority, for each goal its preconditional and postconditional chess concept node in GA is explained as described above. The depth of search tree is brought and criteria are explained, where each criterion is also a regularity, which may need calculation.

For each taught planning algorithm and related chess knowledge Solver suggests testing chess problems. Solution to the problem suggested by the student is being compared with the solution

suggested by the PPIT algorithm generated by the same chess knowledge or the same plan. Moves comparison is done by the Connection Tool and if they do not match, that means the student did not understand the explained plan or concept correctly, the correct solution of the problem is demonstrated to the student, the tutoring knowledge is explained again and a new problem for the same knowledge is suggested to the student to solve.

**iv. Improving explanations**

If the explanation is bad and the student is not able to understand the concept, the problem is forwarded to the expert who improves the definition of the expected chess knowledge to make it understandable and clear and explains the student if needed.

Regular improvement of Solver and PPIT are achieved by Connection Tool. The tool provides an ability to execute chess powerful engines, which allow checking how well the strategy plan is and request for correction if needed. Also Connection Tool enables rating the student, also creating certain games and suggesting more chess problems by the requests.

To improve provide methodology of performance measurement for RGT Solvers, PPITs and students for variety of RGT problems we need to provide certain assessment approaches. In the future we are going to rely on the performance assessments methodology discussed in [91] where the base of a methodology of assessment of performance of systems combining the strengths of methods measuring performance of competing systems or their constituents by original, used in practice criteria of effectiveness and ones evaluating systems by expert attributes followed by their aggregation into a global quality indicators called the global preferences are presented. The basics of those methods and the requirements to combine them for more effective assessment of performances are discussed. An approach of using of absolute, i.e. real practice, scales of competing systems for assessing constituents of those systems are presented and argumentation of the adequacy of the approach is provided.

**Handling the interactions between the student and tutors**

During the study a new software tool was also designed, that allows establishing a connection between chess engine, for example Solver, and player, to handle chess game and fix its results. The tool has a modular construction. Basically this is the interface between tutoring methods of RGT Solver and the student.

Engines Interaction Tool allows to run chess engines, it creates a session for started engines and controls their lifecycle. It also creates a session for regular player and provides a simple input/output interface for interaction with the chess engine using text commands. Designed tool has possibility to start chess games between the player and the chess engine from scratch or from

defined board position to the end of game or for defined count of game moves. It allows getting info about game states and game results. It also has an ability to connect to board GUI programs to visualize the defined cases. The tool has a mechanism of communication between the chess player and the engine based on using of communication protocol, the rules of which allow sending different types of data.

In context of tutoring concept defined in this work Engines Interaction Tool is able to show information about requested for learning chess regularity, show examples of the regularity using chess board visualization. Then the tool allows to check the understanding of the material using the requested tasks concerning the regularity by organizing chess game or some part of it and comparing the expected results for solving the tasks and the results of chess player. Afterwards some chess games can be handled between learning player and some chess engines to check advantages of learning the regularity for playing chess game as whole from start to finish.

To implement the listed functionality Engines Interaction Tool has the following structure, shown in Fig 48.

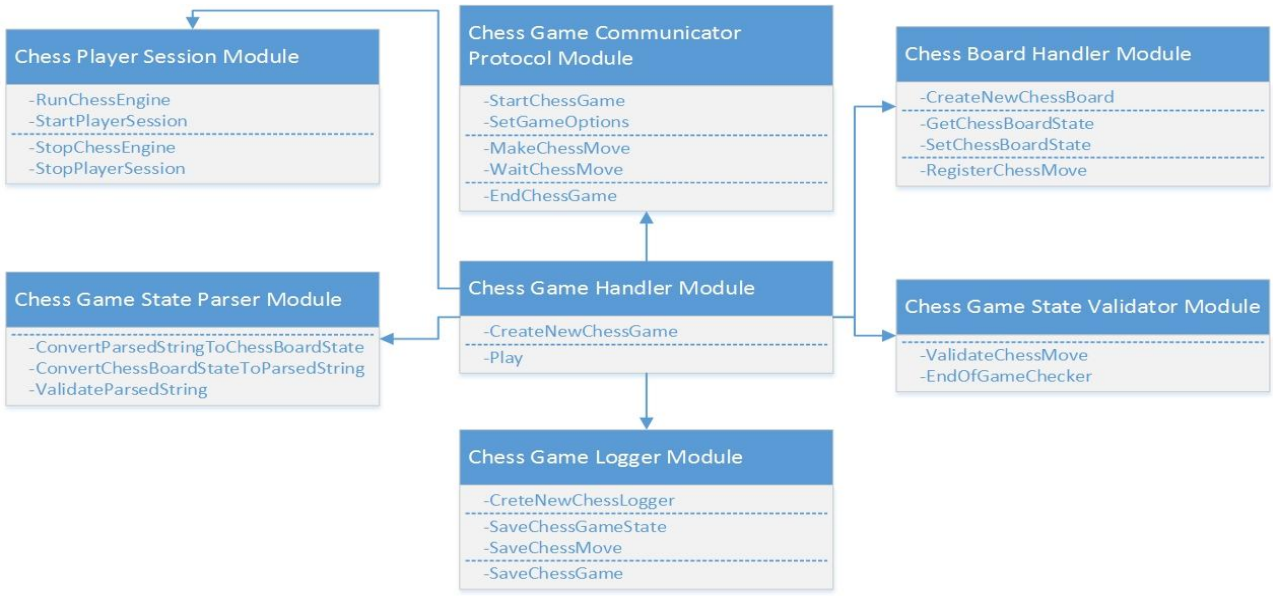



Fig. 50. Interface tool modular construction.

ChessPlayerSession module allows creating a new session for chess player. Chess player can be represented by chess engine or a person. Using this module the chess engine can be started as a new process. The module also handles running chess engine process destroying.

Sending game data between the chess engine and the player can be performed using one of chess game communication protocols. As an example of one of the most used such protocols UCI

[92] was implemented as a part of corresponding tool module named ChessGameCommunicationProtocol.

ChessBoardHandler module is designed to represent chess board. It allows creating new chess boards for game, to set the chess board to some state or to get chess board current state. To change the state of board also registering new moves method can be used.

ChessGameStateParser module is used to save any intermediate game state using some notation, restore it specific condition, described using that notation. One of the most used chess game state notations is Forsyth-Edwards Notation (FEN) [93]. FEN record describes chess game particular board position. ChessGameStateParser module contains FEN parser implementation, so a board state can be converted to FEN string and vice versa.

ChessGameStateValidator module is designed to check if the specified state of game is valid in terms of chess. It also checks the meaning of state for game, so it can detect the end of game.

Designed tool also has a module, which allows conducting chess games named ChessGameHandler. So using this module a new chess game can be started and run until the end or being stopped. The game can be also started from the defined intermediate board position.

ChessGameLogger module allows saving different conditions of game, moves, results and a game as a whole.

To fix the player's results ChessGameLogger module is used to retrieve game data, analyze its contents and provide some progress measurement details.

The designed tool is flexible, so other chess game protocols, other ways of engine running or connected, other parsing notations, etc., can also be implemented and used if necessary.

## 4.2 ADEQUACY OF TUTORING IN CHESS ENGAMES

For the adequacy testing of Solver development of the given tutoring algorithm and development of the Interaction Tool we discuss an example of tutoring a student to a chess endgame “mate by one rook” assuming that the student already knows at least the basic rules of chess, i.e., figures, their moves, mate, stalemate, etc.

A winning strategy in Rock against King endgames:

1. Put mate
2. Avoid stalemate
3. Escape rook from attack
4. Push king to the edge (without putting rook under attack)
5. Make a waiting move when preOpposition appears

6. Bring white king closer to the opponent king

Each step defines a goal and its priority. Tutoring starts with showing the plan to the student. Then each of plan goals is explained and an example is demonstrated on the board.

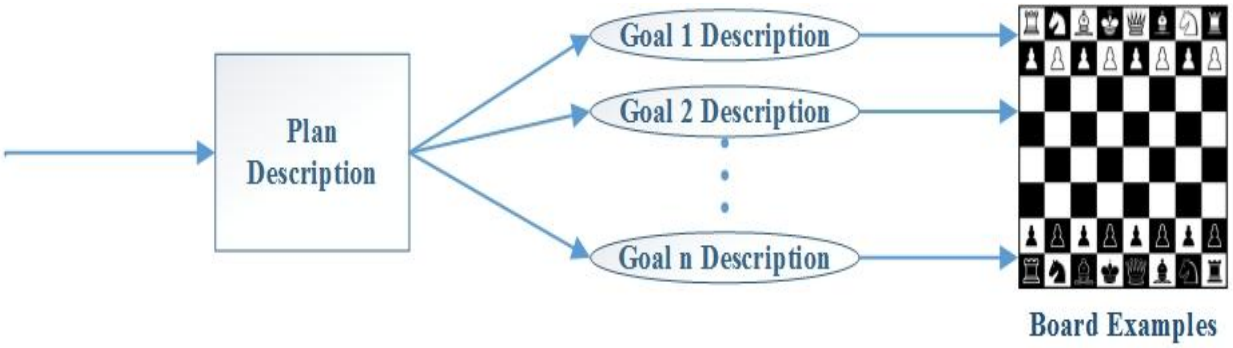



Fig. 51. Chess Concept Tutoring Process.

1. First goal is “put mate” for which preCondition is any situation, and postCondition is a situation where mate exists, the depth is 1. Since we consider a student who already knows chess rules, then explanation of this goal does not need to go deep, it just shows the goal, its tree depth, preCondition and postCondition patterns (in general cases the program will explain “mate” and each of its component concepts if needed). For the mentioned goal a situation is suggested to the student to solve. The student solves the problem and if that is correct the next goal explanation is started, if the student makes a wrong move comparing to Solver execution of the goal, then the goal is explained again and again a situation is suggested to solve. The comparison of student to Solver suggested moves, as well as chess game playing ability are provided by Interaction Tool.

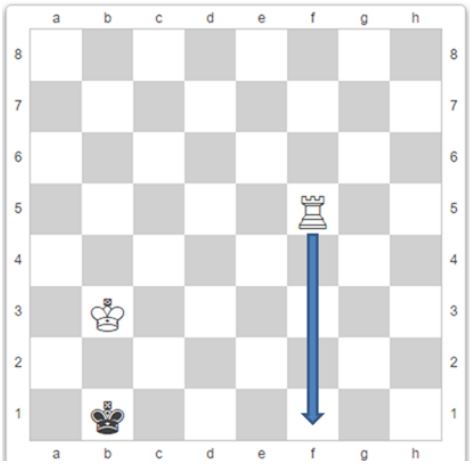


Fig. 52 Chess goal of “put mate”.

2. “Avoid stalemate” goal is explained similar to 1st goal since there is no much difference in knowledge levels used in those goals, preCondition of this goal is again any situation and the postCondition is a situation where no stalemate appears. The depth of search is 1. Again stalemate is known from chess rules, and if no, then it is explained. We consider a student who knows, then Solver just brings the info in the goal and suggests a situation to make a move or moves which are good for this goal. Again this is done using Interaction Tool.

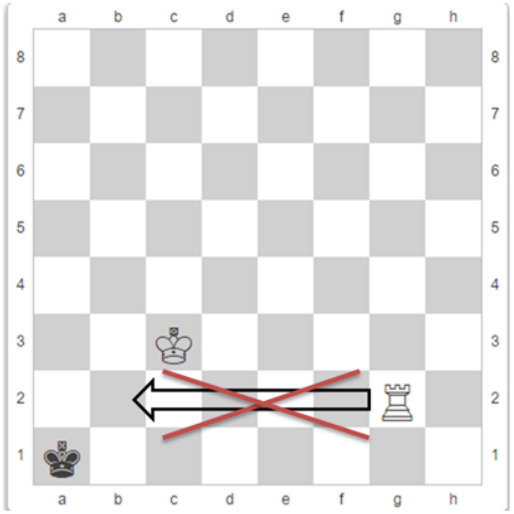



Fig. 53. Avoid stalemate concept

3. “Escape rook from attack” is a goal which preCondition is “rook under attack” abstract, indicating the goal is applicable for situations where the rook is under the attack (in the example we consider it under opponent king’s attack). The postCondition is a situation

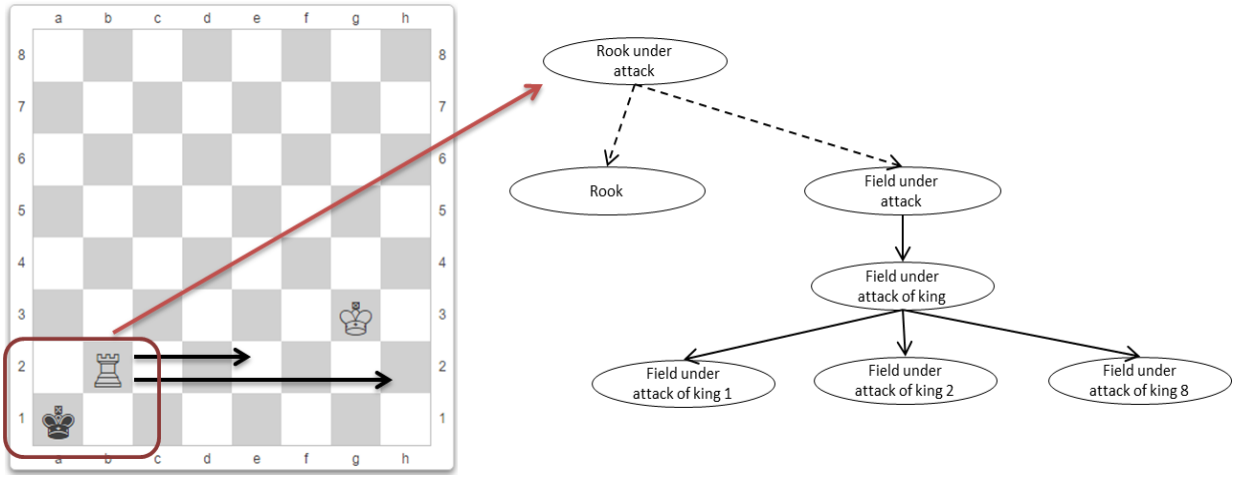



Fig. 54. Chess concept “rook attacked” and goal “escape rook attack”.

where rook is not under attack and the vertical or horizontal coordinate of the rook is not changed depending on what plan we execute: pushing king by vertical or by horizontal. It has a search depth of 1 and the evaluator will have one criterion defined which calculates the distance of the rook and opponent king by vertical/horizontal direction. The explanation is started from overall definition of the goal, where we can consider the student does not know the concept “rook under attack” which is a concept derived from “field under attack” concept indicating that there is a rook instead of field, which is shown to the student. If the student does not know “field under attack” concept, it is explained by its specifications, where there are different types of attacks, such as “field under attack of king” which is needed in this very case and other similar specifications. “Field under attack” concept is a virtual abstract in GA and “field under attack of king” is its specification (GA node types are described in chapter 2). “Field under attack of king” is also a virtual abstract which needs to be explained by its all specifications where each possible attack of king is being demonstrated, overall there are 8 specifications of this concept to explain. We consider the student already knows field is under attack of king abstract, as this is a part of basic chess rules defining king moves and attacks. After explanation of “rook under attack” an example for this concept is demonstrated on the board. Next postCondition is explained, where “rook under attack” concept is used again. For the goal depth is indicated and unchanged horizontal or vertical coordinate of the rook is shown as regularity. The criterion is also named to the student indicating that goal achievement is considered better when the distance of the rook from opponent king is maximal by showing the name of the criterion, its regularity and demonstration of two different examples mentioning which is better for this criterion. Again a situation is suggested to solve and solution is compared with Solver execution of the same goal.

4. “Push king to the edge (without putting rook under attack)”, where preCondition can

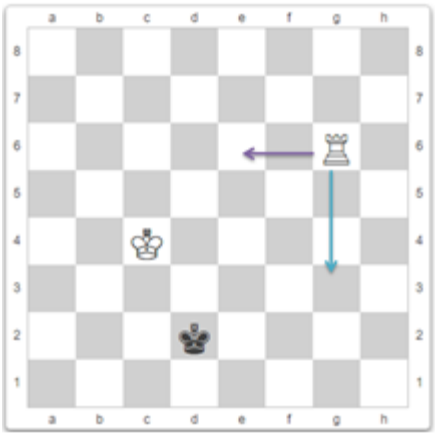


Fig. 55. Goal “push king to the edge”.

be any situation, so nothing to explain and postCondition is “rook is not under attack” which is already explained in 3rd goal, search depth is 2. The evaluator has two criteria which are brought to the student. First is: moves of opponent king are closer to the edge are better, where Solver uses definition of “edge” concept here which is explained to the student as lines of board (defined as set in GA) where either x = 1 or x = 8 or y = 1 or y = 8. The second criterion for this goal evaluator is the number of actions opponent king can do, and the better action is the action which allows fewer number of actions by opponent king. King moves are known by the student and the criterion just shows that minimal number of opponent king moves are better. One more time a situation to solve the goal is suggested and compared to the Solver solution for the same situation.

5. “Make a waiting move when perOpposition appears” goal is defined: preCondition is preOpposition situation. preOpposition is a concept to be taught. In Solver we defined it as a concept that contains two kings, king1 and king2 and is a virtual abstract in GA. Its explanation is done by tutoring of each specification of the given virtual abstract, where each specification identifies specific relations between two kings where opponent kings appear, e.g., one of perOpposition specifications, we call it perOppositionByVertical1 identifies the following relations for the kings which are shown during the explanation process of preOpposition virtual abstract king1.x = king2.x + 2, king1.y = king2.y - 1. Similarly another preOpposition situation is explained to the student perOppositionByVertical2, where regularties shown to the user are king1.x = king2.x + 2, king1.y = king2.y + 1. Other preOpposition specifications are explained to the student, too and for each of them certain situations are braught as examples. The postCondition of this goal is a situation where the own king position is not changed and vertical/horizontal (depending on the direction of pushing king) coordinate of the rook is not changed, so only regularities defining unchanged king position and rook vertical/horizontal coordinate and explaining situations are shown to the student. Searching depth of goal is 1. The evaluator again has one criterion, indicating the distance between opponent king and own rook shall be maximal, which is shown to the student, too. Usually a chess knowing student would not need to be taught for the regularity of “king position is not changed” and many similar concepts at all, but if he/she needs it, the regularity will be shown and a certain example is brought.

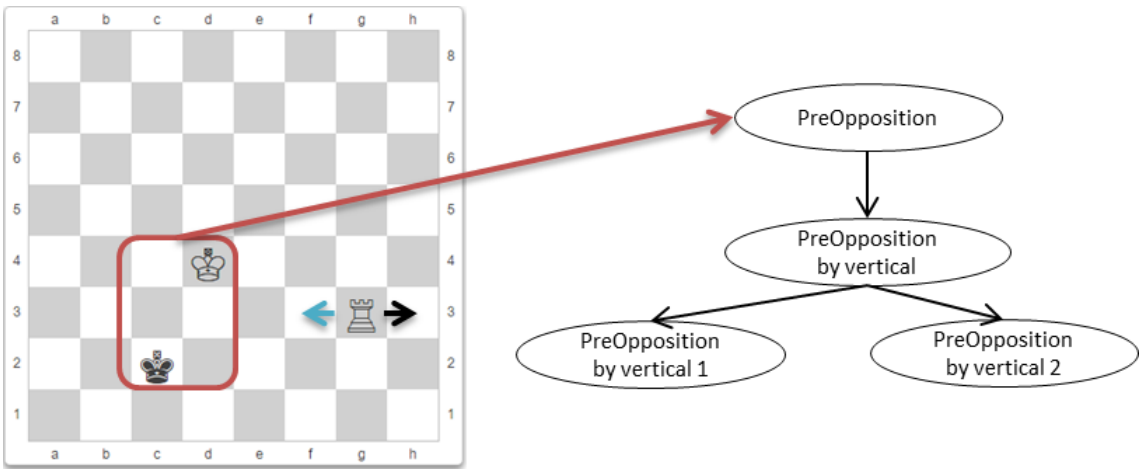



Fig. 56. ”preOpposition” chess concept and “make waiting move” goal.

6. “Bring white king closer to the opponent king, but avoid opposition” goal does not have any related concept for explanation in preCondition, while for postCondition “opposition” concept is being explained. Opposiotn explanation is similar to preOpposition concept with the difference that specifications of Opposition virtual abstract are two, oppositionByVertical and oppositionByHorizontal, searching depth is 1. The evaluator has one criterion, which defines the distance of the king from the opponent king is minimal. We can calculate this by the following formula “(king.cordX-opponentKing.cordX)2 + (king.cordY-opponentKing.cordY)2”, and the criteria are named to the student. The formula is shown as the regularity calculating the minimal distance if needed (here also usually the student would not need this regularity for calculation of two kings distance).

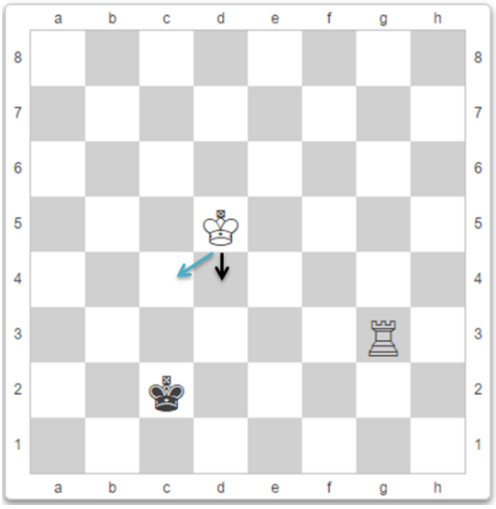


Fig. 57. “bring king closer to opponent king” goal.

**Examining tutoring Rock against King**

After tutoring the student for the "mate by one rook" strategy this is being tested on a situation, where each move of student is compared with each move of PPIT executed in Solver. After each move Interaction Tool provides opponent's move after student making correct move according to the plan. If the student suggested move does not match PPIT suggested move, then the PPIT action is explained in details for the PPIT plan execution algorithm. The correctness of PPIT execution of "mate by one rook" is demonstrated in chapter 2. If the student is able to solve the situation as PPIT does, then the concept is counted as learned. If the student was not able to learn the plan, then explanation process is done again. If the same problem with understanding appears several times (the number is configurable depending on the student performance and abilities) a request is sent to the expert that Solver was unable to explain. Meantime while PPIT and student are solving "mate by one rook" situations powerful chess engines are executed by the Interaction tool to measure the performance of PPIT and student, too. If PPIT solution to the same problem is bad, then a note is sent to the expert about the notices issue and situation where it appears.

Similar to the described example more endgame plans and relevant chess knowledge pieces can be taught to the students in the interactive manner described above without effective input by the expert during explanation process. The demonstrated example proves that the developed algorithm and software tools are able to tutor students to chess problems and concepts, particularly chess endgames can be taught and executed and PPIT solution and concepts understanding by the student can be measured to improve the tutoring performance.

## 4.3 SUMMARY OF CHAPTER 4

The main problems considered in this chapter are:

1. Algorithms, methods and software for tutoring. Algorithms of tutoring to students for RGT problems within RGT Solver with certain implementation and integration of chess interface and explanation mechanisms are designed and developed, measuring the performance and providing interface for the student is developed. The developed approach has several advantages:
   a. The mechanism of tutoring is personalized for each student by their levels, including genius and autistic students.
   b. Interactive environment for making level by level tutoring, testing, feedback provision and correction by detailed description are included, where powerful chess engines are included in the process.
   c. Students' performance measurement means are provided in the developed external tool.

d. It is generic to RGT problems and can be adopted any other RGT problems where interface integration is still required to describe concepts, measure performances and tutoring strategies.

2. The designed algorithms and software adequacy is proved. Adequacy is experimented by tests provided in chess endgames tutoring, particularly rook endgame tutoring is tested.

# CONCLUSION

Knowledge-based strategies were studied in Computing Center of Academy of Sciences (now Institute for Informatics and Automation Problems at the Academy of Sciences of Armenia problems) since 1957 and National Polytechnic University of Armenia.

A large class of unsolved and practically important combinatorial problems of competitions, defense and communications, where the spaces of solutions can be presented by reproducible game trees (RGT), were identified, including certain management and marketing, intrusion protections problems, various military problems, chess and chess-like combinatorial games.

In RGT problems interacting actors perform identified types of actions in specified types of situations some of which are identified as goals. Actors try to achieve those goals by actions performed in the situations and the aim of problems is to construct acceptable strategies.

Exhaustive search methods are useless for RGT problems due to enormous sizes of game trees that represent them.

We search for human-like solutions for these problems due to the high effectiveness and productivity of expert approaches in searching RGT solutions and due to the importance of bringing human-computer communication to the level of human interactions in problem solving.

The dissertation is devoted to development of

1. models and programs for presentation of RGT expert knowledge including plans and goals
2. algorithms and programs matching situations to the knowledge
3. knowledge-based strategy search algorithms for RGT class of problems
4. personalized interactive chess tutoring

as well as provision of

5. experiments for ensuring the adequacy of developed models and algorithms

The main results of the work are:

1. The models for presentation of RGT knowledge based on “have-, be-, do-” categories of English grammar and the algorithms for matching situations to the models of RGT knowledge including goals and plans [28] [27].
2. The proof of the adequacy of the models of RGT knowledge and matching algorithms is provided by the experiments in chess, intrusion protection and management problems [29].
3. RGT plans and goals based strategy search algorithms within RGT Solver [25].

4. The experiments proving the adequacy of developed PPIT strategy search algorithms for RGT problems, particularly, for the Bottvinik's test etude of Reti [25] [24].
5. A model of personalized interactive tutoring in chess based on our models of RGT expert knowledge and strategy search algorithms as well as the evidence of their adequacy to the expert ones based on the chess experiments [26].

# REFERENCES


[1] E. Pogossian, Adaptation of Combinatorial Algorithms, Yerevan: Academy of Sciences of Armenia, 1983, pp. 1-.293.

[2] E. Pogossian, "On Modeling Cognition," in *Computer Science and Information Technologies (CSIT11)*, Yerevan, Sept.26-30, 2011.

[3] E. Pogossian, "Focusing Management Strategy Provision Simulation.," in *Proceedings of the CSIT2001, 3d Inter. Conf. in Comp. Sci. and Inf. Technologies*, Yerevan, 2001.

[4] E. Pogossian, A. Djavadyan, "A Game Model For Effective Counteraction Against Computer Attacks In Intrusion Detection Systems," in *NATO ASI, Data Fusion for Situation Monitoring, Incident Detection, Alert and Response Management"*, Tsahkadzor, Armenia, August 19-30, 2003, pp. 30.

[5] E. Pogossian, A. Grigoryan, "Anomalies dynamic analysis and correction software," *Proceedings of IPIA in Mathematics and Computer Sciences*, 2008.

[6] E. Pogossian, A. Javadyan, E. Ivanyan "Effective Discovery of Intrusion Protection Strategies.," in Workshop on Agents and Data Mining, St. Petersburg, Russia, 2005, pp. 263-274.

[7] E. Pogossian, D. Dionne, A. Grigoryan, J. Couture, E. Shahbazian, "Developing Goals Directed Search Models Empowering Strategies Against Single Ownership Air Threats," in *International Conference in Computer Sciences and Information Technologies (CSIT2009)*, Yerevan, 2009, pp. 155-163.

[8] A. Grigoryan, Z. Naghashyan, M. Gurgiyan, E. Pogossian, "Strategy Knowledge Acquisition Interface for Computation Anomalies Dynamic Analysis and Correction," in *International Workshop in Security*, 2010.

[9] E. Pogossian, "Combinatorial Game Models For Security Systems.," in *NATO ARW on "Security and Embedded Systems*, Porto Rio, Patras, Greece, 2005.

[10] E. Pogossian, "Effectiveness Enhancing Knowledge Based Strategies for SSRGT Class of Defense Problems," in *NATO ASI 2011 Prediction and Recognition of Piracy Efforts Using Collaborative Human-Centric Information Systems*, Salamanca, Spain, 2011.

[11] C. Shannon, "Programming a Computer for Playing Chess," *Philosophical Magazine*, vol. 41, no. 314, 1950.

[12] [Online]. Web Resource for Stockfish Chess Engine Available: https://stockfishchess.org/.

[13] [Online]. Web Resource for Komodo Chess Engine Available: https://komodochess.com/.

[14] M. Botvinnik, About Solving Approximate Problems, Moscow: Sov. Radio, 1979.

[15] E. Pogossian, M. Hambartsumyan, Y. Harutyunyan, "A Repository of Units of Chess Vocabulary Ordered by Complexity of their Interpretations," *National Academy of Sciences of Armenia, IPIA*, 1974-1980, pp. 1-55.

[16] E. Pogossian, "Specifying Personalized Expertise," in *International Conference Cognition and Exploratory Learning in Digital Age*, Barcelona, Spain, 2006, pp.151-159.

[17] M. Botvinnik, Computers in Chess: Solving Inexact Search Problems. Springer Series in Symbolic Computation, with Appendixe, New York: Springer-Verlag, 1984.

[18] E. Pogossian, V. Vahradyan, A. Grigoryan, "On Competing Agents Consistent with Expert Knowledge," in *Workshop on Autonomous Intelligent Systems - Agents and Data Minin*, St. Petersburg, 2007.

[19] T. Bagdasaryan, A. Grigoryan, Z. Naghashyan, "Developing of a Scripting Language Interpreter for Regular Acquisition of Expert Knowledge," in *CSIT2009 : International Conference in Computer Sciences and Information Technologie*, Yerevan, 2009.

[20] Z. Naghashyan, E. Pogossian, "Developing Java Software for Representation, Acquisition and Management of Strategy Knowledge," *Mathematical Problems of Computer Sciences, Proceedings of IPIA*, 2010, pp. 10.

[21] Z. Naghashyan, "Developing software for formation and acquisition lexical units in certain limited languages," PhD Thesis, Yerevan, 2010.

[22] K. Khachatryan, V. Vahradyan, "Graphical Language Interpreter Unified for SSRGT Problems and Relevant Complex Knowledge," in *Computer Science and Information Technologies (CSIT11) Proc. Of the Conf.*, Yerevan, 2011.

[23] K. Khachatryan, "Developing algorithms and programs for formation of and search in knowledge bases of competition and combating problems," PhD Thesis, Yerevan, 2013.

[24] S. Grigoryan, "On Validity of Personalized Planning and Integrated Testing Algorithms in Reproducible Games," in *Computer Science and Information Technologies (CSIT15) Proc. Of the Conf.*, Yerevan, Armenia, 2015, pp. 317-321.

[25] S. Grigoryan, "Structuring of Goals and Plans for Personalized Planning and Integrated Testing of Plans," *Mathematical Problems of Computer Science,* vol. 43, 2015.

[26] S. Grigoryan, L. Berberyan, "Developing Interactive Personalized Tutors in Chess," *Mathematical Problems of Computer Science,* vol. 44, 2015, pp. 116-132.

[27] K. Khachatryan, S. Grigoryan, "Java Programs for Matching Situations to the Meanings of SSRGT Games," in *Proceedings of SEUA Annual conference*, Yerevan, 2013, pp. 127-135.

[28] K. Khachatryan and S. Grigoryan, "Java Programs for Presentation and Acquisition of Meanings in SSRGT Games," in *Proceedings of SEUA Annual conference*, Yerevan, 2013, pp. 135-141.

[29] K. Khachatryan, S. Grigoryan, T. Baghdasaryan, "Experiments Validating the Be-Have-Do Meaning Presentation Model and Matching Algorithm for Competing and Combating Problems," Yerevan, Armenia, 2013, pp. 155-159.

[30] B. Stilman, M. Aldossary, "Revisiting Major Discoveries in Linguistic Geometry with Mosaic Reasoning," 2015.

[31] T. Winograd and F. Flores, Understanding Computers and Cognition, Norwood, NJ: Ablex Corporation, 1987.

[32] J. Laird, A. Newell and P. Rosenblom, "SOAR: An Architecture for General Intelligence" *Artificial Intelligence 33*, 1987, p. 1–64.

[33] J. Pitrat, "Meta-Explanation in a Constraint Satisfaction Solver," 2006, pp. 1118-1125.

[34] D. Wilkins, “Practical Planning: Extending the Classical AI Planning Paradigm”, vol. 205, San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1988, pp. 205.

[35] J. Laird, “The Soar Cognitive Architecture”, England: MIT Press, 2012.

[36] D. Scales, "Efficient Matching Algorithms for the SOAR/OPSS Production System," California, 1986.

[37] M. Minsky, “The Society of Mind, New York, London, Toronto, Sydney, Tokyo, Singapore: Symon & Schuster, 1988.

[38] D. Roy, "Grounding Language in the World: Signs, Schemas, and Meaning," in *Cognitive Machines Group, The Media Laboratory, MIT http://www.media.mit.edu/cogmac / projects. html*, Massachusetts, 2005.

[39] E. Pogossian, "Modeling of Meaning Processing and its Applications in Competition and Combating Games," in *Proceedings of SEUA Annual conference*, Yerevan, Armenia, 2013.

[40] [Online] Nato Grants Web resource Available: http://www.nato.int/science/e/newinitiative.htm.

[41] M. Chussil, D. Reibstein, "Putting the Lessons before the Test. In Wharton on to Analyse & Develop Competitive Strategies," in *Warton on Dynamic Competitive Strategy*, New Jersey, John Wesley & Sons, 1997, pp. 343-368.

[42] E. Pogossian, "Focusing Management Strategy Provision Simulation.," in *Proceedings of the CSIT2001, 3d Inter. Conf. in Comp. Sci. and Inf. Technologies*, Yerevan, 2001.

[43] E. Pogossian, "Business Measurements By On-The-Job Competition Scales," in *International Conference "Management of Small Business: Problems, Teaching, Future"*, Sevastopol, Sept. 20th -24th 2004.

[44] J. Furnkranz, "Machine Learning in Games: A Survey in "Machines that Learn to Play Games"" Nova Scientific, 2001.

[45] C. Donninger, "A Graphical Language for Expressing Chess Knowledge.," *International Computer Chess Association Journal 19(4),* p. 234–241, 1996.

[46] T. Mitchell, P. Utgoff, R. Banerji, "Learning by Experimentation: Acquiring and Refining Problem-Solving Heuristics.," in *Machine Learning: An Artificial Intelligence Approach*, San Mateo, CA, Morgan Kaufmann, 1983, Chapter 6.

[47] M. Buro, "ProbCut: An effective selective extension of the alpha-beta algorithm" *International Computer Chess Association Journal 18(2),* 1995, pp. 71–76.

[48] M. Buro, "Experiments with Multi-ProbCut and a new high-quality evaluation function for Othello" in *Games in AI Research*, Maastricht, The Netherlands, 1999.

[49] P. Utgoff, D. Precup, "Constructive Function Approximation," in *Feature Extraction, Construction and Selection: A Data Mining Perspective, Volume 453 of The Kluwer International Series in Engineering and Computer Science*, Kluwer Academic Publishers., 1998, p. Chapter 14.

[50] T. Nakano, N. Inuzuka, H. Seki, H. Itoh, "Inducing Shogi Heuristics Using Inductive Logic Programming," in *Proceedings of the 8th International Conference on Inductive Logic Programming (ILP-98)*, Madison, WI, 1998.

[51] D. Abdi, "Analysis of pruned minimax trees," vol. 24, 2013.

[52] P. Skowronski, Y. Bjornsson, M. Winands, "Automated Discovery of Search-Extension Features," vol. 12.

[53] C. Garbay, "Knowledge Acquisition and Representation," in *The Biomedical Engineering Handbook*, Second Edittion ed., CRC Press LLC, 2000.

[54] B. Buchanan, M. Shotcliffe, "Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project," Addison-Wesley, Reading, Massachusetts, 1984.

[55] B. Buchanan, D. Barstow, "Construction of an Expert System," in *Building Expert Systems*, Addison-Wesley Publishing Company, Inc, 1983, pp. 127-167.

[56] R. Davis, D. Lenat, "Knowledge-Based System in Artificial Intelligence", New Yourk: MacGraw-Hill, 1982.

[57] J. Boose, "Knowledge Acquisition Program for Expert Systems Based on Personal Construct Psychology," *International Journal of Man-Machine Studies*, vol. 23, 1985, pp. 495-525.

[58] J. Breuker, B. Wielinga, "Use of Models in Interpreting Verbal Data," in *Knowledge Elicitation for Expert Systems: a Practical Handbook*, Plenum Press, 1987, pp. 17-44.

[59] D. Rumelhart, G. Hinton, R. Williams, "Learning Internal Representations by Error Propagation," in *Parallel and Distributed Processing*, vol. 1, Cambridge, Massachusetts, MIT Press, 1986, p. Chapter 8.

[60] F. Taroni, C. Aitken, P. Garbolino, A. Biedermann, "Bayesian Networks and Probabilistic Inference in Forensic Science", John Wiley & Sons, 2006.

[61] J. Quinlan, "Induction of Decision Trees," *Machine Learning*, vol. 1, no. 1, 1986, pp. 81 - 106.

[62] I. Witten, E. Frank, "Data Mining: Practical Machine Learning Tools and Techniques", Second Edition ed., Morgan Kaufman, 2005.

[63] G. Paliouras, V. Karkaletsis, "Machine Learning and Its Applications", Springer, 2001.

[64] M. Kearns, U. Vazirani, "An Introduction to Computational Learning Theory", MIT Press, 1994.

[65] C. Brutyan, I. Zaslavski, L. Mkrtchyan, "Certain Methods of Automated Synthesis of Positional Strategies in Games," *Problemi Kibernetiki*, 1967, p. 50.

[66] [Online]. UML® Web Resource Page Available: http://www.uml.org/.

[67] I. Craig, "Object-Oriented Programming Languages: Interpretation Undergraduate Topics in Computer Science Series", London: Springer, 2007, p. 252.

[68] Z. Naghashyan, "Developing Graphical Language Interpreter for Representing

Communicable Knowledge of the concepts," in *Proceedings of SEUA Annual Conference*, Yerevan, 2010.

[69] D. Poo, D. Kiong, S. Ashok, “Object-Oriented Programming and Java”, London: Springer Science+Business Media, 2008.

[70] G. Hjálmtÿsson, R. Gray, "Dynamic C++ Classes, A lightweight mechanism to update code in a running program," June 1998. [Online]. Available: http://www.usenix.org/publications/library/proceedings/usenix98/full_papers/hjalmtysson/hjalmtysson.pdf. [Accessed 2010].

[71] J. Gosling, B. Joy and G. Steele, "Java Language Specification(Second Edition): Execution," July 1996. [Online]. Available: http://java.sun.com/docs/books/jls/second_edition/html/execution.doc.html#44459.

[72] G. Levenfish, “Book For a Beginner Chess Player”, Moscow, 1957.

[73] "Chesscademy chess learning website," 2015. [Online]. Available: https://www.chesscademy.com.

[74] T. Ogneva, "Some Psychological Aspects of Teaching Children The Chess Game," London, 2004.

[75] V. Anisheva, “Methodical Aspects of Individualized Chess Initial Training for Primary School Children”, Moscow, 2002.

[76] D. Martinovic, I. Markovic, "Piece Values and Piece Exchange in Chess: Individualized Instruction of Schoolchildren Using Tests With Three Levels of Difficulty," *Mathematics. Computing Education* , 2011.

[77] A. Kirsanov, "Individualization of Educational Activity As a Pedagogical Problem," *Kazan University Publishing*, 1982.

[78] I. Unt, "Individualization and Differentiation of Teaching," *Education Publishing*, 1990.

[79] A. Granickaya, "Learn to think and act. Adaptive learning in school," *Education Publishing*, 1991.

[80] E. Pogossian, E. Arakelova, "Tools for Testing and Correction of The Completeness of Knowledge Acquisition by Autistic Children," Yerevan, Armenia, 2011.

[81] D. Gadwal, Greer J., McCalla G. "Tutoring bishop-pawns endgames: an experiment of using knowledge-based chess as a domain for intelligent tutoring," 1992.

[82] Y. Wilks, "An Artificial Intelligence Approach to Machine Translation," in *Machine*

*Translation*, Springer US, 2009, pp. 27-63.

[83] R. Hausser, “Foundations of Computational Linguistics, Human-Computer Communication in Natural Language”, 2nd Edition ed., Berlin, New York: Springer, 2001.

[84] H. Uchida, M. Zhu, T. Della Senta, “The UNL, a Gift for a Millennium”, Tokyo, Japan: UNU/IAS, 1999.

[85] E. Pogossian, "On a Transparent Presentation of Written English Syntax," in *5th International Cognitive Linguistics Conference*, Vrije Universiteit, Amsterdam, 1997.

[86] C. Forgy, "Rete: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem," *Artificial Intelligence 19*, pp. 17-37, 1982.

[87] T. Baghdasaryan, "Adapting RGT Solver Interface to Management Strategy Search Problems," *Mathematical Problems of Computer Science*, vol. 39, pp. 135-145, 2013.

[88] "TAC SCM web page," [Online]. Available: http://tac.sics.se/page.php?id=1.

[89] B. Karapetyan, "High level strategy description language in games," *Mathematical Problems of Computer Science (in Russian),* vol. 16, 1986. .

[90] E. Pogossian, "On Assessment of Performance of Systems by Combining On-the-Job and Expert Attributes Scales," Yerevan, 2015.

[91] S. Meyer-Kahle, R. Huber, "UCI (universal chess interface)," November 28, 2000.

[92] S. Edwards, "Forsyth-Edwards Notation, Portable Game Notation Specification and Implementation Guide," 1994.

# APPENDIX A. GLOSSARY OF ACRONYMS

ADS – Anomalies Detection Systems

API – Application Programming Interface

CPMU - Choosing Plans with Max Utility

FEN – Forsyth–Edwards Notation

GMP - Generating Moves by a Plan

GUI – Graphical User Interface

IGAF – Intermediate Goals at First

JSON - JavaScript Object Notation

KBMS – Knowledge Base Management System

OO – Object Oriented

OOP – Object Oriented Programming

OSP - Optimal Strategy Provision

PPIT – Personalized Planning and Integrated Testing

RHP - Reducing Hopeless Plans

RGT –problems where Space of Solutions can be represented by **Reproducible Game Trees**

UCR – Units of Chess Repository

TZT - Trajectory-Zones based Technique

UML – Unified Modeling Language

UNL – Universal Networking Language