Mathematical Problems of Computer Science 49, 115--122, 2018.

Object-Oriented Modeling of Matching to Systemic Classifiers

Sedrak V. Grigoryan, Nairi P. Hakobyan and Hovhannes S. Vrtanesyan

Institute for Informatics and Automation Problems of NAS RA
e-mail: addressforsd@gmail.com, hakobyannairi@gmail.com, hovhannesvrtanesyan@gmail.com

Abstract

In this paper we present a version of object-oriented implementation of models of constructive regularized mental systems, *mentals*, and *systemic classifiers* introduced in [1] as well as algorithms for matching them to situations. We experiment the adequacy of the models and algorithms for the chess representing kernels of the class of combinatorial problems, where space of solutions can be represented by Reproducible Game Trees (RGT).

Keywords: Modelling, Systemic classifiers, Mental, Matching, Chess.

1. Introduction

1.1. A number of researches ([1, 3]) search for systemic solutions of combinatorial problems (such as chess), as it is stated that those problems cannot be adequately solved by parametric methods [2]. We follow the line of approach given in [1], trying to provide a programmatic implementation of the model, suggest matching algorithms for these implementations, as well as provide evidence of their adequacy.

1.2. As it is described in [1], *mental systems* represent realities, in particular utilities, but have varying effectiveness with respect to the goals and are processed to support utilization and gaining benefits from utilities.

It is stated that a mighty way of enhancement of effectiveness of mss, and thus, cognizers, is the *regularization of classifiers* induced by mental doers and mental systems, while classifiers Cl of members x of communities C are regularized in C if accompanied by ontological in C methods, instructions allowing x regularly provide positive samples of inputs of Cl, as well as let the members of C do the same by communicating with x.

Doers, we assume, are realities having in- out- put parts and for available *inrealities,* i.e, realities at the input parts, either elaborating certain output realities or staying passive.

Classifiers of roots and utilities are identified as *root and induced goals.*

1.3. To model mentals and systemic classifiers, we consider a class of regularized competition problems, where the space of solutions can be represented by reproducible game trees (RGT).
RGT is a class of problems, which includes the following requirements:
A. there are (a) interacting actors (players, competitors, etc.) performing (b) identified types of actions in the (c) specified types of situations;
B. there are identified utilities, goals for each actor;
C. actions for each actor are defined. [4]
1.4. We consider the implementation of the given in [1] models of mentals and systemic classifiers for RGT problems, particularly for the kernel RGT problem, chess, intensively studied since Shannon's pioneer work in 1949 [9].
In what follows, we present an implementation for the models of systemic classifiers induced by mentals, provide algorithms of matching for the given models, as well as the ways to experiment their adequacy for the presentation of chess mental systems.

2. Implementation of Systemic Classifiers

2.1. Natural languages are systemic and comprehensive by their coverage of msystems, but classifiers are not constructive and model only fuzzy ones, because they determine not the positives of msystems, but only IDs of positives and IDs of rels between them [1].
OOP languages are covering mdoers as well but are more systemic with respect to algorithms since they involve attribute/parent/doing relationships corresponding to Have/Be/Do ones in natural languages [10].
We follow the line of Object-Oriented implementation of systemic classifiers based on Have/Be/Do relationships that in contrast with the previous versions of implementations [5, 6] aim to be comprehensive with respect to specification of mentals [1].

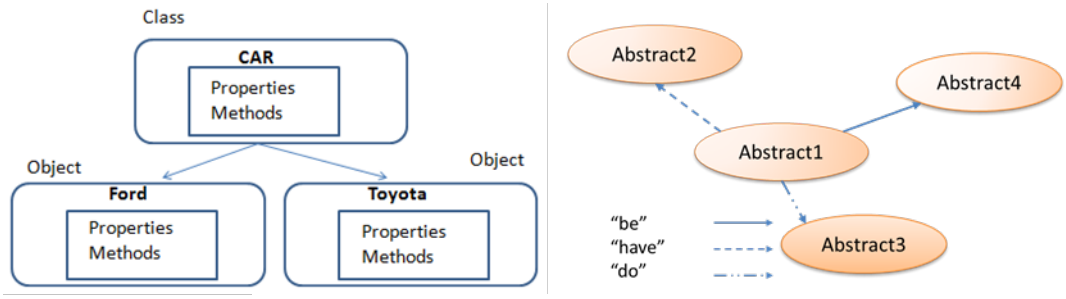

Fig. 1. OOP (left) and Have/Be/Do (right) presentations.

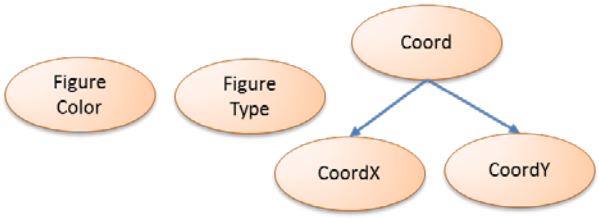

Fig. 2. Nuclears for chess.

2.1.1 We start the implementation of systemic classifiers from nuclears, which are presented as OOP classes containing only one rule. It is similar to primitive types in OOP. For instance, in chess, we define the following as nuclears: coordinates, color, and figure type.
Nuclears are similar to nuclear abstracts defined in [5].
2.1.2 To make the matching algorithms easier, we introduce basic classifiers, which we define as the minimal units to appear in the situation. In chess, fields of the chess boards, which contain or do not contain a figure are basic classifiers. These classifiers can only have nuclears as attributes. We create -have relations between Minimal classifiers and their attributes. In case of parent existence (for simplicity, the current implementation allows having only one parent for each type of classifier) we create a -be relation between Minimal classifiers and their parent.
2.1.3 More complex systemic classifiers, which include as attributes any other types of classifiers, we name CompositeClassifiers. Similar to Minimal classifiers, they can have only one parent with a -be relation and -have relations with their attributes. They are similar to OOP classes. As in OOP classes, this type of classifiers can also be virtual. In OOP this is achieved by having one undefined method, we achieved it by having undefined attributes. An attribute is called undefined if in its description there is at least one rule that is not specified (not specified rules have the value '?').
Only for CompositeClassifiers we implement negation. There are 2 types of negations: negation of general concept and negation for a specific instance. In the first case, CompositeClassifier is generally negated if there is no such concept at all. For example, if we want to check whether there is a 'check' concept on the chess board, this is a general negation. In the second case, negation refers only to the exact instance, say, to the concept 'there is no figure on e4 field'.
CompositeClassifiers are similar to complex abstracts defined in [5] but provide more flexible presentations.
2.1.4 We develop Sets similar to arrays in OOP and similar to Sets described in [6] with restriction that they can be only continuous and represent sets of classifiers. For simplicity, we assume that Sets consist only of CompositeClassifiers (since they can only represent any other type of classifier), that are connected to the Sets with -have relations. Sets have 3 rules on their instances: 1. impose restrictions on lower and upper bounds; 2. define continuity with respect to attributes instances; 3. indicate the direction.

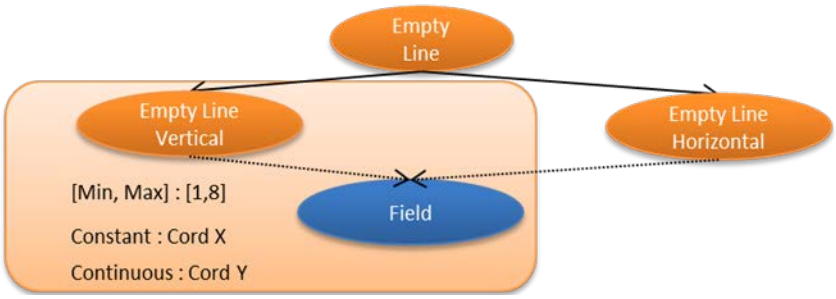

Fig. 3. Presentation of "Empty Line" Set, which is virtual, where orange colored nodes are Sets.

2.1.5 Actions are similar to OOP functions that describe some algorithm. They are connected with -do relation with an Actor that makes an Action (precondition).
In contrast to the actions described in [5], we represent actions as a composition of the main precondition classifier, which is the actor, other precondition classifiers and rules describing the algorithm, how to change the precondition and provide an instance of classifier as an output.

Fig. 4. Presentation of Actions, which perform similar to methods in OOP.

2.1.6 Dynamic classifiers describe the transition from one state to another similar to [7]. For example, in chess it is impossible to describe the concept ‘King cannot escape’ with classifiers that describe static chess concepts. These types have 2 attributes – precondition and postcondition, both of them are CompositeClassifiers and are connected with -have relations. DynamicClassifiers, the same as previously described classifiers, can have only one parent with -be relation.

2.1.7 Goals also have 2 attributes – precondition and postcondition with -have relations. They can also have one parent with -be relation. In addition, there is also an evaluating function, which is defined by rules and maximal depth of tree to evaluate.

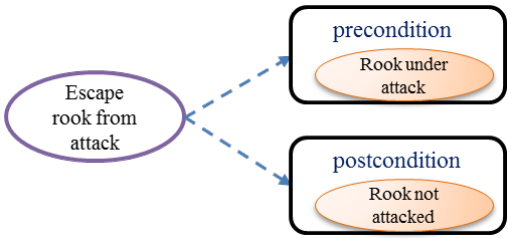

Fig. 5 A goal from plan of "mate by rook".

2.1.8 Plans include Goals as attributes connected by -have relations, and the Goals are ordered by priorities.

2.1.9 Corresponders have a CompositeClassifier as an attribute with a -have relation and Actions with a -do relation. Some real life concepts, like factories, cars implementation require both functionality and attributes existence. For instance, the concept ‘car’ should have 4 wheels (attribute) and must be able to be driven.

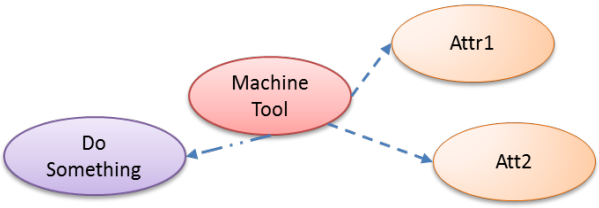

Fig. 6. Presentation of a Machine Tool, which has both actions to perform and attributes.

2.2 Experimenting Models for Chess

We will demonstrate the algorithm on the example of the chess concept ‘Check’ (shown in Figure 3). ‘Check’ is a CompositeClassifier. Attributes of ‘Check’ are ‘King’ and ‘Field is under attack’. ‘King’ is a MinClassifier and asfollows from its definition, there are nuclears as attributes – 2 ‘Coordinates’, ‘Figure type’ and ‘Figure color’. ‘King’ has a parent ‘Figure’. ‘Field is under attack’ is a virtual CompositeClassifier, where attributes are ‘Field’, ‘Attacker’, and it has children ‘Field is under attack of pawn’, ‘Field is under attack of bishop’ etc. ‘Field’ is a MinClassifier with all the described nuclears with attributes. ‘Field’ is a parent for ‘Figure’. ‘Attacker’ is a ‘Figure’. We’ll define one of ‘Field is under attack’’s children, the others are defined equally.

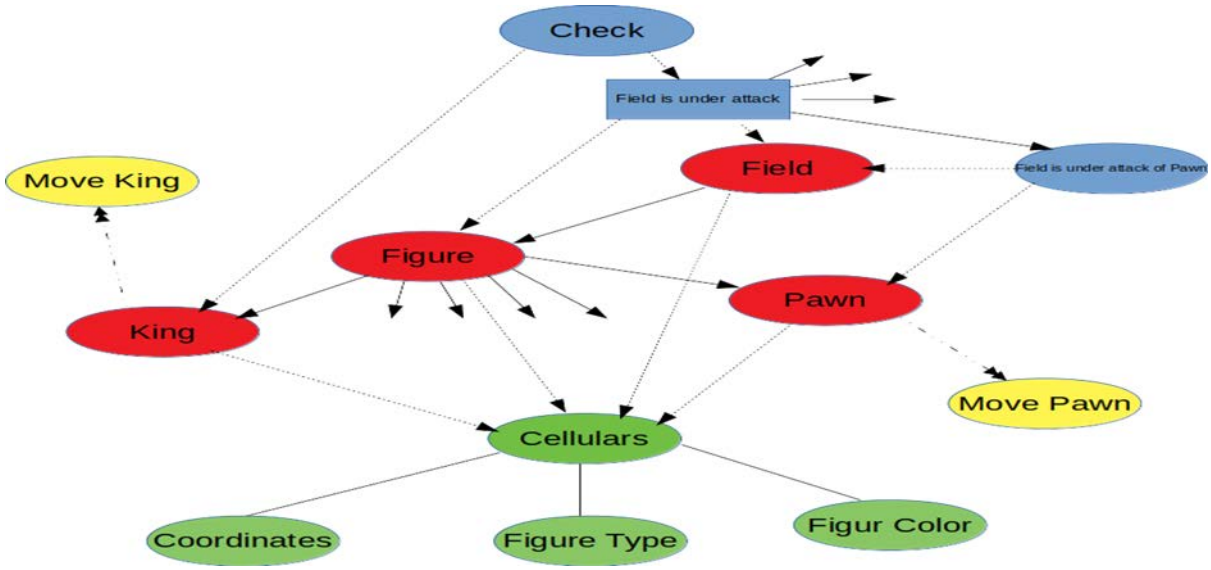

Fig. 7. Systemic presentation of "Check" chess concept in Solver18. Blue nodes present composites (including virtual "Field is under attack" concept), red nodes present MinDoers, green nodes present nuclears and blue nodes present actions.

'Field is under attack of pawn' is a CompositeClassifier with attributes 'Field' and 'Attacker', where 'Attacker' is a 'Pawn'.

3. Matching Mental System to Situations

Fig. 8. Presentation of situations.

We represent situations as instances of nuclears (for example, for chess it has a figure color: "white, black, no color", figure type: "pawn, bishop, knight, rook, queen, king, no figure", coordinates nuclear has two children – x and y both having 1-8 coordinates). Nuclears instances are specified in groups with groupIDs. A set of nuclear instances with group ids describes the situation as described in [8], but, on the contrary, here we describe the side that will act in the given situation.

3.1 General Matching

3.1.1 Matching Nuclears

Matching nuclears is performed by checking its rule, if the rule is satisfied it's being matched, and an instance of active nuclear with the given value is sent forward through the system.

3.1.2 Matching Minimal Classifiers

Matching of minclassifiers is done by matching its attributes.

3.1.3 Matching Composite Classifiers

Matching composites is done by matching its attributes and local rules checking. If all the attributes are matched and rules are satisfied, then the composite is matched.

Virtual composites are created and matched as described in [8], usages, which are the uses of virtually specified composite classifiers, are matched by their virtual specification matching and additional rules checking, while specifications matched as regular composites trigger matching of their parent pure virtual composite classifiers.

3.1.4 Matching Sets is done by the checking of each instance of its composite element. If the given numbers of instances that satisfy all the rules defined in set are received, the Set instance is considered as matched. Basically the set activation is described in [8].

3.1.5 Matching actors and actions

Actions are activated with the following steps: 1. Actor is activated and other precondition attributes are activated, 2: Action is activated and can be applied to the situation and modify it.

3.1.6 Matching Dynamic Classifiers

Matching dynamic classifiers basically described in [6], dynamics are matched when precondition composite is matched, and postcondition for all of the final situations is satisfied.

3.1.7 Matching Goals

Goals, as described above, have a precondition, which is composite, this shall be matched in order to consider the goal classifier. When the precondition is matched, the given depth of the tree is generated and final situations are evaluated with the given evaluation function. All the instances of goals that match the postcondition are considered as matched, and their evaluation value is assigned.

3.1.8 Matching Corresponders

Matching Corresponders is performed by both checking its attributes matching and by checking its action, i.e., comparing the expected postcondition with the output for the given precondition.

In addition, matching of any type of classifiers to situations can be done by matching of their children, i.e., any child matching triggers matching its parent (e.g. “king”, “queen” or any other type of figure matching means matching “figure” as well).
General matching process is done through the system for the given situation. At the first step, it finds matching some classifiers, at the next step it continues doing the same until there are no more matched classifiers. After each step matching negated classifiers, dynamics and goals are performed, as described below:

3.2 Matching one Systemic Classifier

In some cases, it is just needed to find a certain concept on the situation, say, to see if there’s a check on the chess board or not. Check classifier matching is searching exactly for it. In this case, all other classifiers, which are not relevant to the Check classifier, are ignored in the system when processing the situation.

3.2.1 Negated Classifiers Matching

Negated classifiers are composite ones, there are two types of negations in the system, the regular negation shows only a flag in the composite classifier, and when this flag is active and there is no such concept in the situation, then the appropriate classifier instance with all empty values is fired as it is The other case of negation is having a certain negated concept in the situation, then in contrast to the general negation, this needs the main attribute of negated composite to be activated first. Then if after finishing the processing of the situation positive presentation of the concept is not matched, the negation instance with the given main attribute is considered as matched. E.g., if “e4 field is not attacked” concept is being searched, then the field with the value e4, which can be defined as a main attribute there is being checked if under attack. If no “field under attack”, where the field is “e4” found, then e4 field is not under attack, which is what we needed.

2.3 Matching Algorithm by the example of chess

Let’s discuss an example of Matching on an example of ‘check’ described in the previous chapter. On field c3 white King is matched, on field c8 black rook is matched and on field h8 black king is matched. On fields c1-c7, a8, b8, d8-g8 matches ‘field is under attack of rook’, which triggers matching its parent ‘field is under attack’. Rook on c8, the free line between rook and white king activates ‘field under attack of rook’, in its turn its triggers matching ‘field under attack’ the ‘field’ attribute of which is the same as ‘King’, consequently, the rule in chess ‘check’ CompositeClassifier is also satisfied and ‘check’ is matched.

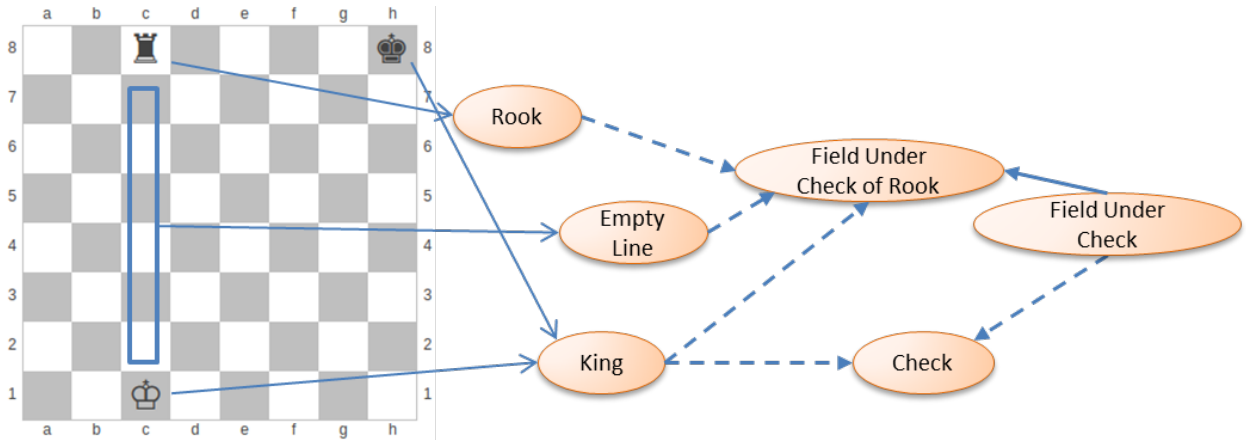

Fig. 9. Situation where "Check" concept is matched.

4. Conclusions

In the following work models for constructive regularized mental systems were described and their implementations were discussed.
1. Object-Oriented models for systemic classifiers discussed in [1] were developed, particularly nuclears', goals', corresponders' and other types of classifiers' models were given, and their adequacy evidence is demonstrated for the example of chess concepts.
2. Algorithms for matching situations to each type of systemic classifiers are provided, where there are two types of matching algorithms, a. matching situations to the whole system of classifiers, where all the matched classifiers are activated and b. matching situations to certain classifiers, where only instances of expected classifiers are searched. Adequacy of matching algorithms is experimented for the example of chess concepts.

Acknowledgements
Authors express their deep gratitude to Professor Edward Pogossian for supervising the work.

References

[1] E. Pogossian, "Towards Adequate Constructive Models of Mental Systems", *International Conference in Computer Sciences and Information Technologies*, Yerevan, Armenia, pp. 6, 2017.
[2] M. Botvinnik, "Computers in chess: solving in exact search problems", *Springer Series, in Symbolic Computation, with Appendixes, Springer-Verlag*, New York, 1984.
[3] M. Botvinnik, *About Solving Approximate Problems*, (in Russian), S. Radio, Moscow, 1979.
[4] E. Pogossian, V. Vahradyan and A. Grigoryan, "On competing agents consistent with expert knowledge", *Lecture Notes in Computer Science, AIS-ADM-07: The International Workshop on Autonomous Intelligent Systems - Agents and Data Mining*, St. Petersburg, Russia, June 6-7, pp. 229-241, 2007.
[5] K. Khachatryan and S. Grigoryan, "Java programs for presentation and acquisition of meanings in SSRGT games", *Proceedings of SEUA Annual conference*, Yerevan, Armenia, pp. 127-135, 2013.
[6] S. Grigoryan, *Research and Development of Algorithms and Programs of Knowledge Acquisition and Their Effective Application to Resistance Problems*, pp. 111, Yerevan, Armenia, 2016.
[7] S. Grigoryan, *"Dynamic Knowledge Integration into HBD Knowledge"*, *International Conference in Computer Sciences and Information Technologies*, Yerevan, Armenia, pp. 3, 2017.
[8] K. Khachatryan and S. Grigoryan, "Java programs for matching situations to the meanings of SSRGT games", *Proceedings of SEUA Annual conference*, Yerevan, Armenia, pp. 135-141, 2013.
[9] C. Shannon, "Programming a Computer for Playing Chess", *Philosophical Magazine*, vol. 41, no. 314, 1950.
[10] E. Pogossian, "On modeling cognition", *International Conference in Computer Sciences and Information Technologies*, Yerevan, Armenia, pp 194-198, 2011.

Submitted 11.09.2017, accepted 14.02.2018.

Սիստեմիկ դասակարգիչների համապատասխանեցման օբյեկտ-կողմնորոշված մոդելավորում

Ս. Գրիգորյան, Ն. Հակոբյան և Հ. Վրթանեսյան

Ամփոփում

Աշխատանքում ներկայացնում ենք կառուցողական կանոնակարգված մտավոր համակարգերի, մտավորների (mentals) և սիստեմիկ դասակարգիչների օբյեկտ-կողմնորոշված մոդելների մի իրականացում, որոնք ներկայացված են [1]-ում, մշակված են ալգորիթմներ դրանք իրավիճակներին համապատասխանեցնելու համար: Մոդելների և ալգորիթմների համապատասխանությունը մենք փորձարկում ենք շախմատի համար, որը ներկայացնում է միջուկ կոմբինատոր խնդիրների մի դասի համար, որոնց լուծումների բազմությունը վերարտադրելի ծառ է (RGT):

Объектно-ориентированное моделирование соответствия системным классификаторам

С. Григорян, Н. Акобян и О. Вртанесян

Аннотация

В даной работе представлена версия оъектно-ориентированной реализации моделей конструктивно регуляризованных ментальных систем, mentals и системных классификаторов, представленных в [1], а также алгоритмов их сопоставления с ситуациями. Адекватность моделей и алгоритмов мы экспериментурием для шахмат, представляющее собой ядро класса комбинаторных задач, где пространство решений представляет собой воспроизводимое дерево.