Dynamic Knowledge Integration into HBD Knowledge Presentation Model

Sedrak Grigoryan

Division of Computational and Cognitive networks, IPIA, NAS RA
Yerevan, Armenia
e-mail: addressforsd@gmail.com

ABSTRACT

We aim to construct adequate models of knowledge presentation. Language is one of the main spaces of knowledge transfer, thus we have developed a model based on main dimensions of verbs in English "have", "be", "do" (HBD). Below we purpose ways to enhance the existing HBD model to provide more flexible knowledge acquisition of it by integration of dynamic types.

Keywords

Knowledge, knowledge presentation, acquisition, dynamic knowledge.

1. INTRODUCTION

In [1, 2] HBD model, its extensions and knowledge acquisition are discussed. Current language-based HBD model implements only 3 main dimensions of language, particularly "have" and "be" dimensions in nuclear, primitive, composite and set abstracts, and "do" dimension in action abstracts.

The model is developed within RGT Solver, an environment aimed to solve a certain space of problems with the following requirements for them: a) there are interacting actors, b) which perform certain type of actions c) in specified situations d) for achieving identified benefits, and the space of solution for these problems is represented in a Reproducible Game Tree (RGT) [3-5]. We study knowledge acquisition, strategy construction problems for different RGT problems, particularly for chess, which is a matter of studies since Shannon's works in 1950s.

The developed HBD model covers most of RGT knowledge pieces, particularly chess, however, some concepts which imply dynamics in them still cannot be adequately acquired. Particularly, chess concept of "Mate" in [6] does not cover general "mate" concept implementation. Some kind of concepts, such as "king cannot escape" required by "mate" can be defined dynamically. "King cannot escape" in "mate" is a concept which considers all moves of king and makes sure for any possible move by king it is still under check. This implies dynamical checking of each situation change after king's move on the given situation.

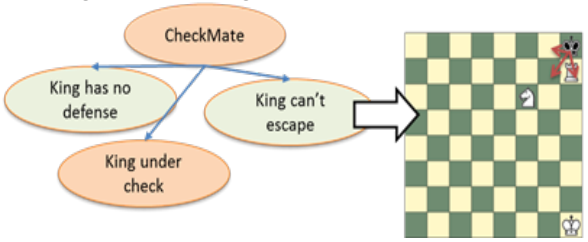

Fig. 1. "King can't escape" dynamic concept

In [7,8] dynamics is achieved at the strategy construction stage. Similar approach is suggested in [3] based on goals and plans and was integrated into RGT Solver in [5]. However, integration of dynamic knowledge into HBD model essentially improves the knowledge acquisition process. The main pending questions to answer are:

a. How to construct and integrate dynamic abstracts (DA)?

b. How does the matching apply to dynamic abstracts?

In the following sections we will try to answer these questions.

2. CONSTRUCTION OF DYNAMIC ABSTRACTS AND THEIR INTEGRAION

We aim to construct a dynamic type of abstract which will be able to define and match RGT knowledge of dynamic types, such as 'king cannot escape' and integrate them into HBD model of RGT Solver.

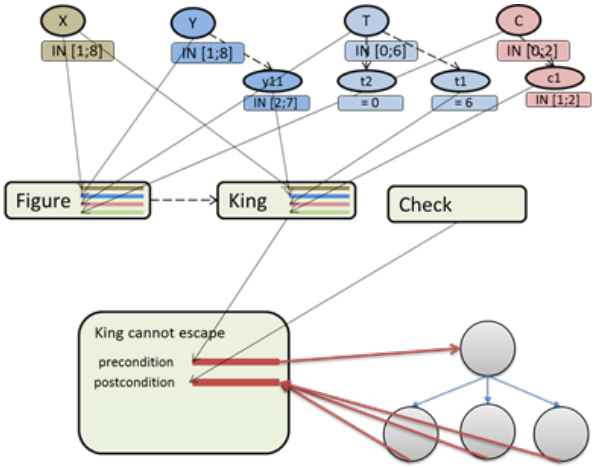

Fig. 2. Dynamic concept "King can't escape" in HBD model

As the image describes "King can't escape" concept could be defined as dynamic:

1. Since this type of knowledge implies a tree construction in it and in any final situation the same condition needs to be satisfied, thus, the abstract will have a precondition for the initial tree node and postcondition for the final situations. Thus, the main attributes of dynamic abstracts will be precondition and postcondition, both composite types [2], can include any type of precondition or postcondition concepts. The difference between the actions and dynamic abstracts is that actions suggest a situation transformation and get to new situations after their applications, while dynamic abstracts only search for certain RGT knowledge in the current situation.

2. The depth of tree is usually small, but can vary from one concept to another, e.g., "king cannot escape" concept requires only one move depth checking, while other dynamic concepts, such as "perpetual check" may require a

deeper search. Thus, we also need to specify the max tree depth in the dynamic abstract.

3. Precondition and postcondition abstracts can depend on each other, but cannot be unspecified, in other words dynamic abstracts cannot be virtual [2]. Some dynamic abstracts may require checking of values against initial situations, thus, dependencies between precondition and postcondition are allowed, but virtuality are not allowed, because concepts cannot be virtual, but at the same time both precondition and postcondition abstracts can be usages of virtual abstracts.

4. Based on previous requirements and rules, we reveal that a) dynamic abstracts will contain 'have' type of relation to their attributes and have no parent, b) dynamic abstracts can be attributes for other composite abstracts.

Basically the above described rules are complete for definition of dynamic abstracts. Below pseudocode for their creation is brought:

```
function createDynamicConcept() {
  input: abstractData
  output DynamicAbstract
  name = getNameFromData(abstractData);
  Abstract duplicate = getAbstractFromLibrary(name);
  Boolean isUpdate = false;
  if(duplicate != null) {
    isUpdate = true;
    if(duplicate.TYPE != DYNAMIC){
      throw NameDuplicationException(name);
    }
  }
  String preconditionName =
getPreconditionName(abstractData);
  Composite precondition = getAbstract(preconditionName);
  String postconditionName =
getPostconditionName(abstractData);
  Composite postcondition =
getAbstract(postconditionName);
  Integer depth = getTreeDepth(abstractData);
  DynamicAbstract abs = createDynamicConceptInternal(
        preconditionName, postConditionName, depth);
  addDynamicAbstractToLibrary(abs);
  return abs;
}
```

3. MATCHING OF DYNAMIC ABSTRACTS

Dynamic abstracts are different from other abstracts in their form, thus, their matching requires a different approach [9]. Thus, dynamic abstracts cannot be matched with a regular pattern matching approach as it is done for other types.

In [5] matching of goals and plans is discussed. As goals are similar to DAs their matching will be also done that way, with some differences. Overall matching of abstracts is done with several phases.

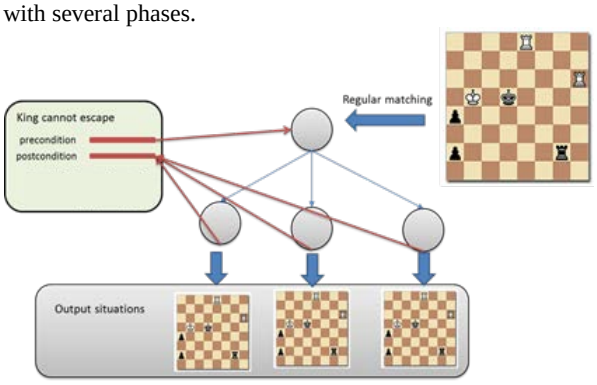

Fig. 3. Matching of dynamic abstracts

a. First all the regular matching is done as described in [9].

b. All of DAs which have active precondition abstracts in phase a.) (in the case of "king cannot escape" activation of 'king' is expected), a tree is generated based on max allowed depth for the given DAs similar to construction of tree for goals. All the end nodes of the tree, i.e., all the output situations have to match the postcondition. In the case of 'king cannot escape' in all of the ending situations king must be under check. This means abstract is matched.

c. After the above step requirements are satisfied and dependencies in the dynamic abstract are satisfied (if there are any), abstract is being activated. Activation of DAs triggers activation of other abstracts connected to the DA, which could not be activated at phase a. because of the missing attribute activated in b. which can itself trigger activation of new DAs.

d. Above a., b., c., steps are performed sequentially until no new abstract is being activated. At the end of this step we have the complete matching process for the given input situation.

4. EXPERIMENTING THE UPDATED HBD MODEL IN SOLVER

For experimenting the extended with dynamic abstracts HBD model adequacy we consider "rook against king" endgames, where "mate" concept is used, thus, also dynamic abstracts "king cannot escape" and "king has no defense" are required. In contrast to [6], "mate" is defined as a composition "king under check", "king cannot escape" and "king has no defense", where "check" concept definition is the same, while "king cannot escape" and "king has no defense" abstracts are defined as dynamic ones. "King cannot escape" is defined as a dynamic abstract where precondition is "king" and postcondition is "check" for the king identified in precondition and its position is changed to identify that this was a move by king with the depth of only one. "King has no defense" has a precondition of "check" and postcodition is "check" where king's position is not

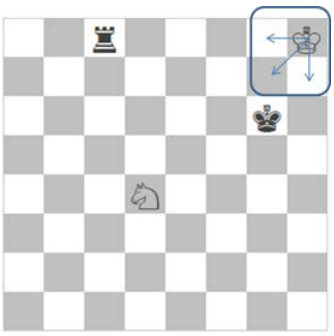

Fig. 4. "King cannot escape" in a rook endgame situation

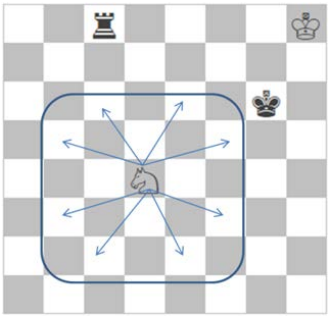

Fig. 5. "King has no defense" in a rook endgame situation

Fig. 6. "Rook against king" endgame

changed, which means other pieces are moved, not king and king is still under check.

The plan for `rook against king`, as described in [5], is as follows: 1) Put mate, 2) Avoid stalemate, 3) Escape rook from attack, 4) Push king to the edge, 5) Make a waiting move, 6) Bring own king closer to the opponent king.

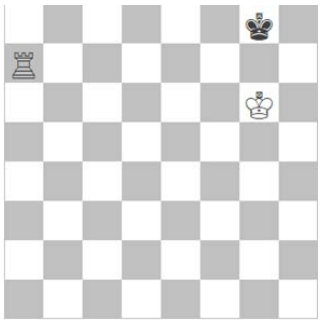

Fig. 7. Situation where "mate" concept with dynamic attributes is detected

Let's consider the situation shown in Fig. 6. The plan performs as described in [5], and after several moves we get to the position shown in Fig. 7. In this situation "put mate" goal can be achieved, as the "mate" after rook's move a8 is detected: for black king "check", "king cannot escape" and "king has no defense" abstracts are activated and "mate" is matched.

4. CONCLUSION

1. Structure of dynamic abstract is defined, where abstracts consist of precondition, which is a composite abstract describing the initial situation to match the knowledge, postcondition, which is a composite abstract describing the final situations for the knowledge (e.g., postcondition for "king cannot escape" is that king is still under check after any move by king) and depth of tree. This kind of knowledge is similar to goals with the difference that it is integrated into the network of abstracts, while goals just refer to abstracts.
2. Algorithm to match these abstracts to the given situations is developed. The algorithm works with several iterations with two main steps, where first step is the regular matching process and the second step is dynamic abstracts matching. DA is considered as matched if precondition is matched and for all of final situations in the generated game tree of that abstract postcondition is matched.
3. Adequacy of these types of abstracts is experimented with "rook against king" situations, where 'mate' concept is defined using dynamic abstract 'king cannot escape'. Based on the defined knowledge Solver is able to solve the given 'rook against king' endgame situations adequately.

The future development of HBD model requires enhancement of the model to unite all the existing knowledge types, including plans and goals. Also generation of goals from different abstracts, including dynamic ones and generation of plans is in the coming steps of development.

5. ACKNOWLEDGEMENTS

Author expresses his deep gratitude to Professor Edward Pogossian for supervising this work and Vanand Mkrtchyan for support in development of Dynamic Abstracts definition interface in Solver.

REFERENCES

[1] E. Pogossian, "On Modeling Cognition" *Computer Science and Information Technologies (CSIT11)*, pp. 194-198, Yerevan, Sept.26-30, 2011.

[2] K. Khachatryan and S. Grigoryan, "Java programs for presentation and acquisition of meanings in SSRGT games", *Proceedings of SEUA Annual conference*, pp. 127-135, Yerevan, Armenia, 2013.

[3] 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*, pp. 229-241, St. Petersburg, Russia, June 6-7, 2007.

[4] E. Pogossian, "Specifying Personalized Expertise" *International Association for Development of the Information Society (IADIS): International Conference Cognition and Exploratory Learning in Digital Age (CELDA 2006)*, pp 151-159, Barcelona, Spain Dec 8-10, 2006

[5] S. Grigoryan, "Structuring of Goals and Plans for Personalized Planning and Integrated Testing of Plans", *Mathematical Problems of Computer Science*, vol. 43, pp. 62-75, 2015.

[6] K. Khachatryan and V. Vahradyan, "Graphical Language Interpreter Unified for SSRGT Problems and Relevant Complex Knowledge", *International Conference in Computer Sciences and Information Technologies*, pp. 178-182, Yerevan, Armenia, 2011.

[7] J. Laird, "The Soar Cognitive Architecture", *MIT press*, England, 2012.

[8] M. Veloso, "PDDL by Example", *Carnegie Mellon University*. 2015.

[9] K. Khachatryan and S. Grigoryan, "Java programs for matching situations to the meanings of SSRGT games", *Proceedings of SEUA Annual conference*, pp. 135-141 Yerevan, Armenia, 2013.