Higher-order Logic Description of MDPs to Support Meta-cognition in Artificial Agents

Vincenzo Cannella, Antonio Chella, and Roberto Pirrone

Dipartimento di Ingegneria Chimica, Gestionale, Informatica, Meccanica,
Viale delle Scienze, Edificio 6, 90100 Palermo, Italy
{vincenzo.cannella26,roberto.pirrone}@unipa.it

Abstract. An artificial agent acting in natural environments needs meta-cognition to reconcile dynamically the goal requirements and its internal conditions, and re-use the same strategy directly when engaged in two instances of the same task and to recognize similar classes of tasks. In this work the authors start from their previous research on meta-cognitive architectures based on Markov Decision Processes (MDPs), and propose a formalism to represent factored MDPs in higher-order logic to achieve the objective stated above. The two main representation of an MDP are the numerical, and the propositional logic one. In this work we propose a mixed representation that combines both numerical and propositional formalism using first-, second- and third-order logic. In this way, the MDP description and the planning processes can be managed in a more abstract manner. The presented formalism allows manipulating structures, which describe entire MDP classes rather than a specific process.

Keywords: Markov Decision Process, ADD, Higer-order logic, u-MDP, meta-cognition

1 Introduction

An artificial agent acting in natural environments has to deal with uncertainty at different levels. In a changing environment meta-cognitive abilities can be useful to recognize also when two tasks are instances of the same problem with different parameters. The work presented in this paper tries to address some of the issues related to such an agent as expressed above. The rationale of the work derives from the previous research of the authors in the field of planning in uncertain environments [4] where the "uncertainty based MDP" (u-MDP) has been proposed. u-MDP extends plain MDP and can deal seamlessly with uncertainty expressed as probability, possibility and fuzzy logic. u-MDPs have been used as the constituents of the meta-cognitive architecture proposed in [3] where the "meta-cognitive u-MDP" perceives the external environment, and also the internal state of the "cognitive u-MDP" that is the actual planner inside the agent. The main drawbacks suffered by MDP models are both memory and computation overhead. For this reason, many efforts have been devoted to define a compact representation for MDPs aimed at reducing the need for computational

resources. The problems mentioned above are due mainly to the need of enumerating the state space repeatedly during the computation. Classical approaches to avoid enumerating the space state are based on either numerical techniques or propositional logic. The first representations for the conditional probability functions and the reward functions in MDPs were numerical, and they were based on decision trees and decision graphs. These approaches have been subsequently substituted by algebraic decision diagrams (ADD) [1][8]. Numerical descriptions are suitable to model mathematically a MDP but they fail to emphasize the underlying structure of the process, and the relations between the involved aspects. Propositional or relational representations of MDPs [6] are variants of the probabilistic STRIPS [5]; they are based on either first-order logic or situation calculus [2] [10][9][7]. In particular, a first-order logic definition of Decision Diagrams has been proposed. In this work we propose a mixed representation that combines both numerical and propositional formalisms to describe ADDs using first-, second- and third-order logic. The presented formalism allows manipulating structures, which describe entire MDP classes rather than a specific process. Besides the representation of a generic ADD as well as the implementation of the main operators as they're defined in the literature, our formalism defines *MetaADDs* (MADD) as suitable ADD abstractions. Moreover, *MetaMetaADDs* (MMADD) have been implemented that are abstractions of MADDs. The classic ADD operators have been abstracted in this respect to deal with both MADDs and MMADDs. Finally, a recursive scheme has been introduced in order to reduce both memory consumption and computational overhead.

2 Algebraic Decision Diagrams

A Binary Decision Diagram (BDD) is a directed acyclic graph intended for representing boolean functions. It represents a compressed decision tree is. Given a variable ordering, any path from the root to a leaf node in the tree can contain a variable just once. An Algebraic Decision Diagram (ADD) [8][1] generalizes BDD for representing real-valued functions $f : \{0, 1\}^n \rightarrow \mathbb{R}$ (see figure 1). When used to model MDPs, ADDs describe probability distributions. The literature in this field reports the definition of the most common operators for manipulating ADDs, such as addition, multiplication, and maximization. A homomorphism exists between ADDs and matrices. Sum and multiplication of matrices can be expressed with corresponding operators on ADDS, and suitable binary operators have been defined purposely in the past. ADDs have been very used to represent matrices and functions in MDPs. SPUDD is the most famous example of applying ADDs to MDPs[8].

3 Representing ADDs in Higher-order logic

In our work ADDs have been described in Prolog using first-order logic as a fact in a knowledge base to exploit the Prolog capabilities of managing higher-order logic. An ADD can be regarded as a couple $\langle S, \mathbf{v} \rangle$ where S is the tree's structure,

which is made up by nodes, arcs, and labels, and **v** is the vector containing the values in terminal nodes of the ADD. Let's consider the ADD described in the previous section. Figure 1 shows its decomposition in the structure-vector pair. Terminal nodes are substituted with variables, and the ADD is transformed into its structure. Each element of the vector **v** is a couple made up by a proper variable inserted into the i-th leaf node of the structure, and a probability value that was stored originally into the i-th leaf node.

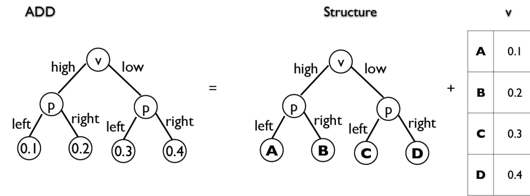

Fig. 1. The decomposition of an ADD in the corresponding structure-vector pair $\langle S, \mathbf{v} \rangle$.

In general, ADDs can be represented compactly through a recursive definition due to the presence of isomorphic sub-graphs. At the same manner, a structure can be defined recursively. The figure 2 shows an example.

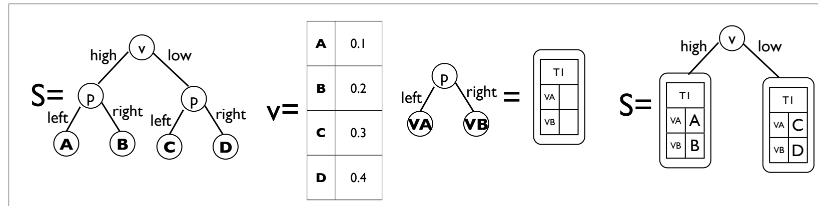

Fig. 2. Each structure can be defined recursively as composed by its substructures. A structure can be decomposed into a collection of substructures. Each substructure can be defined separately, and the original structure can be defined as a combination of substructures. Each (sub)structure is described by the nodes, the labels and the variables in its terminal nodes. Such variables can be unified seamlessly with either another substructure or another variable.

Following this logic, we can introduce the concept of *MetaADD* (MADD), which is a structure-vector pair $\langle S, \mathbf{v} \rangle$ where S is the plain ADD structure, while **v** is an array of variables that are unified with no value (see figure 3). A MADD expresses the class of all the different instances of the same function, which involve the same variables but can produce different results.

An operator op can be applied to MADDs just like in the case of ADDs. In this way, the definition of the operator is implicitly extended. The actual

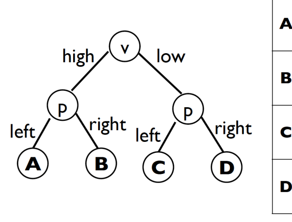

Fig. 3. The MetaADD corresponding to the ADD introduced in the figure 1.

implementation of an operator op applied to MADDs can be derived by the corresponding operator defined for ADDs. Given three ADDs, add_1, add_2, and add_3, and their corresponding MADDs $madd_1$, $madd_2$, and $madd_3$, then $add_1 \,\mathrm{op}\, add_2 = add_3 \Rightarrow madd_1 \,\mathrm{op}\, madd_2 = madd_3$.

The definition of the variables in $madd_3$ depends on the operator. We will start explaining the implementation of a generic operator for ADDs. Assume that the structure-vector pairs for two ADD's are given: $add_1 = \langle S_1, \mathbf{v}_1 \rangle$, $add_2 = \langle S_2, \mathbf{v}_2 \rangle$. Running the operator will give the following result:

$$add_1 \,\mathrm{op}\, add_2 = \langle S_3, \mathbf{v}_3 \rangle$$

Actual execution is split into two phases. At first, the operator is applied to both structures and vectors of the input ADDs separately, then the resulting temporary ADD is simplified.

$$\begin{aligned} S_{temp} &= S_1 \quad \overline{\mathrm{op}} \quad S_2 \\ \mathbf{v}_{temp} &= \mathbf{v}_1 \quad \overline{\mathrm{op}} \quad \mathbf{v}_2 \\ &\mathrm{simplify}(S_{temp}, \mathbf{v}_{temp}) \rightarrow \langle S_3, \mathbf{v}_3 \rangle \end{aligned}$$

Here $\overline{\mathrm{op}}$ is the "expanded" form of the operator where the structure-vector pair is computed plainly. The equations above show that S_{temp} depends only on S_1 and S_2, and it is the same for $\mathbf{v}_{temp}$ with respect to $\mathbf{v}_1$ and $\mathbf{v}_2$.The simplify$(\cdot, \cdot)$ function represents the pruning process, which takes place when all the leaf nodes with the same parent share the same value. In this case, such leaves can be pruned, and their value is assigned to the parent itself. This process is repeated until there are no leaves with the same value and the same parent in any location of the tree (see figure 4). Such a general formulation of the effects produced by an operator on a couple of MADDs can be stored in memory as *Abstract Result* (ABR). ABRs are defined recursively to save both memory and computation too. An ABR is a t-uple $\langle M_1, M_2, Op, M_3, F \rangle$, where M_1, M_2 and M_3 are MADDs, Op is the operator that combines M_1, M_2 and returns M_3, while F is a list of relationships between the variables in M_1, M_2 and M_3, which in turn depend on Op. We applied the abstraction process described so far, to MADDs also by replacing its labels with non unified variables. The resulting structure-vector pairs have been called *MetaMetaADDs* (MMADD) (see figure

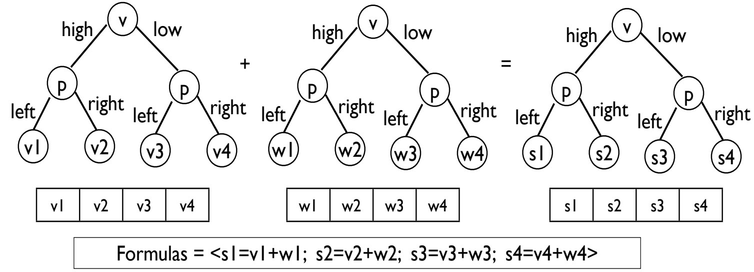

Fig. 4. Two MADDs are added, producing a third MADD. Results are computed according to the formulas described inside the box.

5). We called such computational entity *meta-structure* MS. It has neither values nor labels: all its elements are variables. MS is coupled with a corresponding vector $\mathbf{v}$, which contains variable-label couples (see Figure 5). The definition of

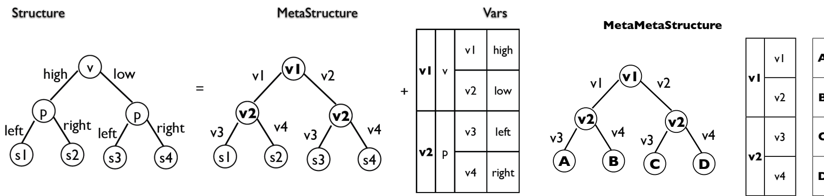

Fig. 5. The decomposition of a MADD structure into the pair $\langle MS, \mathbf{v} \rangle$, and the corresponding MMADD

operators, their abstraction, and the concept of ABR remain unchanged also at this level of abstraction.

4 Discussion of the Presented Formalism and Conclusions

The generalizations of ADDs to second- and third-order logic that were introduced in the previous section, allow managing MDPs in a more efficient way than a plain first-order logic approach. Most part of MDPs used currently, share many regularities in either transition or reward function. As said before, such functions can be described by ADDs. If these regularities appear, ADDs are made up by sub-ADDs sharing the same structures. In these case, computing a plan involves many times the same structure with the same elaboration in different steps of the process. Our formalism allows to compute them only once and save it in a second- and third-order ABR. Every time the agent has to compute structures

that have been used already, it can retrieve the proper ABR to make the whole computation faster. Computation can be reduced also by comparing results in different MDPs. In many cases, two MDPs can share common descriptions of the world, similar actions, or goals, so the results found for a MDP could be suitable for the other one. A second-order description allows comparing MDPs that manage problems defined in similar domains, with the same structure but different values. Finally, a third-order description allows to compare MDPSs, which manage problems defined in different domains but own homomorphic structures. In this case, every second- and third- order ABR computed for the first MDP can be useful to the other one. This knowledge can be shared by different agents. Adding ABRs to the knowledge base enlarges the knowledge of the agent, and reduces the computational effort but implies a memory overhead. Our future investigation will be devoted to devise more efficient ways for storing and retrieving ABRs thus improving the overall performances of the agent.

References

1. R.I. Bahar, E.A. Frohm, C.M. Gaona, G.D. Hachtel, E. Macii, A. Pardo, and F. Somenzi. Algebraic decision diagrams and their applications. In *Computer-Aided Design, 1993. ICCAD-93. Digest of Technical Papers., 1993 IEEE/ACM International Conference on*, pages 188–191, 1993.
2. Craig Boutilier, Raymond Reiter, and Bob Price. Symbolic Dynamic Programming for First-Order MDPs. In *IJCAI*, pages 690–700, 2001.
3. Vincenzo Cannella, Antonio Chella, and Roberto Pirrone. A meta-cognitive architecture for planning in uncertain environments. *Biologically Inspired Cognitive Architectures*, 5:1 – 9, 2013.
4. Vincenzo Cannella, Roberto Pirrone, and Antonio Chella. Comprehensive Uncertainty Management in MDPs. In Antonio Chella, Roberto Pirrone, Rosario Sorbello, and Kamilla R. Johannsdottir, editors, *BICA*, volume 196 of *Advances in Intelligent Systems and Computing*, pages 89–94. Springer, 2012.
5. Richard Dearden and Craig Boutilier. Abstraction and approximate decision-theoretic planning. *Artif. Intell.*, 89(1-2):219–283, January 1997.
6. Charles Gretton and Sylvie Thiébaux. Exploiting first-order regression in inductive policy selection. In *Proceedings of the 20th UAI Conference*, UAI '04, pages 217–225, Arlington, Virginia, United States, 2004. AUAI Press.
7. Jan Friso Groote and Olga Tveretina. Binary decision diagrams for first-order predicate logic. *J. Log. Algebr. Program.*, 57(1–2):1 – 22, 2003.
8. Jesse Hoey, Robert St-aubin, Alan Hu, and Craig Boutilier. Spudd: Stochastic planning using decision diagrams. In *In Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence*, pages 279–288. Morgan Kaufmann, 1999.
9. Saket Joshi, Kristian Kersting, and Roni Khardon. Generalized first order decision diagrams for first order markov decision processes. In Craig Boutilier, editor, *IJCAI*, pages 1916–1921, 2009.
10. Saket Joshi and Roni Khardon. Stochastic planning with first order decision diagrams. In Jussi Rintanen, Bernhard Nebel, J. Christopher Beck, and Eric A. Hansen, editors, *ICAPS*, pages 156–163. AAAI, 2008.