Game Theoretic Golog under Partial Observability

Alberto Finzi
Institut für Informationssysteme, TU Wien
Favoritenstraße 9-11, 1040 Vienna, Austria
DIS, Università di Roma “La Sapienza”
Via Salaria 113, 00198 Rome, Italy

Thomas Lukasiewicz
Institut für Informationssysteme, TU Wien
Favoritenstraße 9-11, 1040 Vienna, Austria
DIS, Università di Roma “La Sapienza”
Via Salaria 113, 00198 Rome, Italy

ABSTRACT

We present the agent programming language POGTGolog, which combines explicit agent programming in Golog with game-theoretic multi-agent planning in a special kind of partially observable stochastic games (POSGs). The approach allows for partially specifying a high-level control program for a system of multiple agents, and for optimally filling in missing details by viewing it as a generalization of a special POSG and computing a Nash equilibrium.

Categories and Subject Descriptors

I.2 [**Computing Methodologies**]: Artificial Intelligence

General Terms

Languages, algorithms

Keywords

Game-theoretic agent programming, Golog, POSG

1. INTRODUCTION

In this paper, we present the language POGTGolog, which extends GTGolog [2] and thus also DTGolog [1] by partial observability. POGTGolog is a combination of explicit agent programming in Golog with game-theoretic multi-agent planning in a special kind of partially observable stochastic games (POSGs) [5]. POSGs are a partially observable generalization of Markov games. They also generalize normal form games, partially observable Markov decision processes (POMDPs) [6], and decentralized POMDPs (DEC-POMDPs) [4]. We consider a special kind of POSG, where at each action selection point, every agent knows what the other agents believe. By this assumption, we can characterize finite-horizon Nash equilibria by finite-horizon value iteration as in fully observable Markov games. The main contributions are as follows:

- We define the language POGTGolog, which integrates explicit agent programming in Golog with game-theoretic multi-agent planning in special POSGs. It is a generalization of GTGolog that allows for partial observability.
- The language POGTGolog allows for specifying a control program for a system of multiple agents, which is then completed in an optimal way by viewing it as a generalization of a special POSG, and computing a Nash equilibrium.
- We show that POGTGolog generalizes its special class of POSGs. Furthermore, we also show that the POGTGolog interpreter is optimal in the sense that it computes a Nash equilibrium of POGTGolog programs.

Note that further details are given in the extended paper [3].

2. PARTIALLY OBSERVABLE GTGOLOG

We now present the language POGTGolog for $n \geq 2$ agents. We first describe the domain theory and the syntax of POGTGolog programs. We then define the semantics of POGTGolog programs. To introduce the framework, we will refer to a rugby example (see Fig. 2), which is adapted from Littman’s soccer example in [7].

Example 2.1 *We assume two competing teams $A = \{a_0, \ldots, a_p\}$ and $B = \{b_0, \ldots, b_q\}$. The rugby field is a 4×5 grid. Each agent occupies a square and is able to do one of the following actions on each turn: N, S, E, W, stand, passTo(a), and receive (move up, down, right, left, no move, pass, and receive the ball, resp.). An agent is a* ball owner *iff it occupies the same square as the ball. The ball follows the moves of the ball owner. The ball owner scores when he/she steps into the adversary goal. When the ball owner goes into the square occupied by the other agent, if the other agent stands, possession of ball changes.*

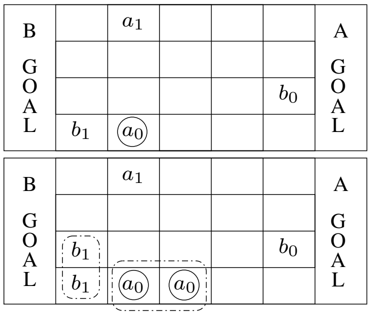

Figure 1: Rugby Example.

Domain Theory. POGTGolog programs are interpreted w.r.t. a background action theory AT and a background optimization theory OT, specified in the Situation Calculus (SC) and extending the Basic Action Theory (see [8]) to represent stochastic actions and rewards. We can illustrate this encoding by considering the

AAMAS’05, July 25-29, 2005, Utrecht, Netherlands.

rugby domain. Given two agents for each team ($A=\{a_0, a_1\}$ and $B=\{b_0, b_1\}$), to axiomatize the theory of actions AT, we introduce the deterministic actions $move(\vec{\alpha}, \vec{\beta}, \vec{m}, \vec{n})$, where $n_i, m_j \in \{N, S, E, W, stand, passTo, receive\}$ (agents α_i and β_j execute concurrently n_i and m_j, resp.) and the fluents $at(\alpha_i, x, y, s)$ (agent α_i is at (x, y) in situation s) and $haveBall(\alpha_i, s)$ (agent α_i has the ball in s) defined by the successor state axioms, e.g. for at we have:

$$\begin{array}{l}
at(\alpha, x, y, do(a, s)) \equiv (\exists x', y', m).at(\alpha, x', y', s) \wedge \\
\quad moved(\alpha, a, m) \ \wedge \ (m = stand \wedge y' = y \vee m = N \wedge \\
\quad\quad y' = y - 1 \vee m = S \wedge y' = y + 1) \wedge x = x' \vee \\
\quad (m = E \wedge x' = x - 1 \vee m = W \wedge x' = x + 1) \wedge y' = y \vee \\
\quad (\exists \beta).(m = passTo(\beta) \vee m = receive) \wedge y' = y \wedge x' = x\,.
\end{array}$$

Here, $moved(\alpha, a, m)$ is true iff m is the action of α in a.

Analogously to [1], we represent stochastic actions by means of a finite set of deterministic actions. When a stochastic action is executed, then with a certain probability nature executes exactly one of its deterministic actions and produces one of its observations. Going back to the example, we can introduce the stochastic actions $moveTo(\alpha_i, x)$ representing the agent attempt in doing x. If $moveTo(\alpha_i, x)$ succeeds, then the associated deterministic action a is executed, i.e., $moved(\alpha_i, a, x)$, otherwise it fails and no action is performed, i.e., $moved(\alpha_i, a, stand)$. We assume also that after the $moveTo$ action, the agent can observe a team member in the direction of the movement, e.g.,

$$\begin{array}{l}
prob(moveTo(\alpha, x), s, a, observe(\alpha')) = p \equiv \\
\quad (\exists y, p_1).moved(\alpha, a, y) \wedge (visible(\alpha, \alpha', a, s) \wedge \\
\quad\quad (y = stand \wedge p_1 = 0.2 \vee y = x \wedge p_1 = 0.8 \wedge \\
\quad\quad\quad p = p_1 \times 0.8) \vee (\neg visible(\alpha, \alpha', s) \wedge p = 0.0))\,.
\end{array}$$

The optimization theory OT specifies a reward and a utility function. The former associates with every situation s and multi-agent action a, a reward to each agent $i{\in}I$, denoted $reward(i, a, s)$. The utility function maps every reward and success probability to a real-valued utility $utility(v, pr)$, e.g., $utility(v, pr) = v \cdot pr$.

Belief States. To model partial observability, we introduce *belief state situations* $b = (b_i)_{i \in I}$, which represent the belief of agent i expressed as a probability distribution over ordinary situations. For example, in Fig. 2, the belief states of a_0 and a_1 are depicted, resp., in the upper and lower part. While in the belief state of a_0 there is only one situation s_1 with probability 1, the belief state of a_1 is a set of four possible situations, i.e. b_1 either at $(1, 1)$ (a) or at $(1, 2)$ (b), and a_0 either at $(2, 1)$ (c) or at $(3, 1)$ (d), with, e.g., the probability distribution: $\{(s_{a,c}, 0.5), (s_{a,d}, 0.3), (s_{b,c}, 0.1), (s_{b,d}, 0.1)\}$.

Syntax of POGTGolog. Given the multi-agent actions represented by the domain theory, *programs* p in POGTGolog are inductively built using the following constructs (where is a condition, p_1 and p_2 are programs, and $\alpha, \ldots, \beta$ are multi-agent actions): *Action sequence*: $p_1; p_2$. *Nondeterministic choice:* $\alpha| \ldots |\beta$. *Test action*: ? (testing 's truth in the current situation). *Nondeterministic choice of an argument*. *Conditionals*, *while loops*, *procedures, including recursion*. We write $\|_{j \in J}\mathbf{choice}(j : a_{j,1}| \cdots |j : a_{j,n_j})$ to denote $(j_1{:}a_{j_1,1}\| \cdots \|j_k{:}a_{j_k,1}) \mid \cdots \mid (j_1{:}a_{j_1,n_{j_1}}\| \cdots \|j_k{:}a_{j_k,n_{j_k}})$, with J a set of agents. Informally, the agents in J execute simultaneously one action each.

For example, the following high-level program (1) represents a game schema for the rugby domain:

$$\begin{array}{l}
\mathbf{proc}(schema, \\
\quad \mathbf{choice}(\boldsymbol{a}_0 \colon moveTo(\boldsymbol{a}_0, E)|stand|passTo(a_1))\| \\
\quad\quad \mathbf{choice}(\boldsymbol{a}_1 \colon moveTo(\boldsymbol{a}_1, E)|moveTo(\boldsymbol{a}_1, S)|receive)\,; \\
\quad \mathbf{choice}(\boldsymbol{a}_0 \colon moveTo(\boldsymbol{a}_0, E)|stand|passTo(a_1))\| \\
\quad\quad \mathbf{choice}(\boldsymbol{a}_1 \colon moveTo(\boldsymbol{a}_1, E)|receive)\,; \\
\quad moveTo(\boldsymbol{a}_0, E)\|moveTo(\boldsymbol{a}_1, E)\,; \\
\quad moveTo(\boldsymbol{a}_0, E)\|moveTo(\boldsymbol{a}_1, E)\,; nil).
\end{array}$$

In this schema, the agents a_0 and a_1 have two possible chances to coordinate themselves in order to pass the ball; after that, both of them have to run towards the goal.

Semantics of POGTGolog. The semantics of a POGTGolog program p w.r.t. AT and OT for two agents 1 and 2 is defined through the macro $DoG(p, b, h, \pi, v, pr)$, where $b = (b_1, b_2)$, $v = (v_1, v_2)$, and $pr = (pr_1, pr_2)$. Here, we have as input the program p, a belief state b, and a finite horizon $h \geq 0$. The predicate DoG then determines a strategy π for both agents 1 and 2, its rewards v_1 and v_2 to 1 and 2, and its success probabilities pr_1 and pr_2 from $[0, 1]$, respectively. We define $DoG(p, b, h, \pi, v, pr)$ by induction on the program structure. For example, the semantics of a two-agents parallel choice is defined as follows:

$$\begin{array}{l}
DoG(\mathbf{choice}(1\colon a_1| \ldots |a_n)\|\mathbf{choice}(2\colon o_1| \ldots |o_m); \\
p, b, h, \pi, v, pr) \ =_{def} \exists \pi_{i,j}, v_{i,j}, pr_{i,j}, \pi_1, \pi_2 : \\
\quad \bigwedge_{i=1}^{n}\bigwedge_{j=1}^{m} DoG(1{:}a_i\|2{:}b_j; p, b, h, 1{:}a_i\|2{:}b_j; \pi_{i,j}, v_{i,j}, pr_{i,j}) \wedge \\
\quad (\pi_1, \pi_2) = selectNash(\{r_{i,j} = utility(v_{i,j}, pr_{i,j}) \mid i, j\}) \wedge \\
\quad \pi = \pi_1\|\pi_2; \mathbf{if}\ \phi_1 \wedge \psi_1\ \mathbf{then}\ \pi_{1,1}\ \mathbf{else\ if}\ \phi_2 \wedge \psi_1\ \mathbf{then}\ \pi_{2,1}\ \ldots \\
\quad\quad\quad\quad \mathbf{else\ if}\ \phi_n \wedge \psi_m\ \mathbf{then}\ \pi_{n,m} \wedge \\
\quad v = \sum_{i=1}^{n} \sum_{j=1}^{m} v_{i,j} \cdot \pi_1(a_i) \cdot \pi_2(o_j) \wedge \\
\quad pr = \sum_{i=1}^{n} \sum_{j=1}^{m} pr_{i,j} \cdot \pi_1(a_i) \cdot \pi_2(o_j)\,.
\end{array}$$

Intuitively, we compute a Nash strategy by finite horizon value iteration for POSGs. For each possible pair of action choices, the optimal strategy is calculated. Then, a Nash strategy is locally extracted from a matrix game by the function *selectNash*.

Strategy Generation. Suppose our aim is to control agent a_1, which executes its part of the strategy π that is obtained from the DoG formula associated with the program p. For example, assuming a 4-steps horizon, an optimal instantiation of the schema (1) is the strategy π such that $AT \cup OT \models DoG(schema, (b_{a_0}, b_{a_1}), 4, \pi, (v_1, v_2), (pr_1, pr_1))$, where v_i and pr_i are the associated values and probabilities, respectively. Given the initial belief state in Fig. 2, a possible strategy could be, e.g.:

$$\begin{array}{l}
passTo(a_1)\|receive\ ;\ moveTo(\boldsymbol{a}_0, stand)\|moveTo(\boldsymbol{a}_1, E)\ ; \\
\quad moveTo(\boldsymbol{a}_0, E)\|moveTo(\boldsymbol{a}_1, E)\ ; \\
\quad moveTo(\boldsymbol{a}_0, E)\|moveTo(\boldsymbol{a}_1, E)\,,
\end{array}$$

which gives to agent a_1 three $moveTo(\boldsymbol{a}_1, E)$ attempts to achieve the touch-line.

Acknowledgments. This work was supported by the Austrian Science Fund Project P18146-N04 and by a Heisenberg Professorship of the German Research Foundation. We thank the reviewers for their constructive comments, which helped to improve our work.

3. REFERENCES

[1] C. Boutilier, R. Reiter, M. Soutchanski, and S. Thrun. Decision-theoretic, high-level agent programming in the situation calculus. In *Proc. AAAI-2000*, pp. 355 362.

[2] A. Finzi and T. Lukasiewicz. Game-theoretic agent programming in Golog. In *Proc. ECAI-2004*, pp. 23 27.

[3] A. Finzi and T. Lukasiewicz, 'Game-theoretic Golog under partial observability', Technical Report INFSYS RR-1843-05-02, Institut für Informationssysteme, TU Wien, 2005.

[4] C. V. Goldman and S. Zilberstein. Decentralized control of cooperative systems: Categorization and complexity analysis. *J. Artif. Intell. Res.*, 22:143 174, 2004.

[5] E. A. Hansen, D. S. Bernstein, and S. Zilberstein. Dynamic programming for partially observable stochastic games. In *Proc. AAAI-2004*, pp. 709 715. AAAI Press.

[6] L. P. Kaelbling, M. L. Littman, and A. R. Cassandra. Planning and acting in partially observable stochastic domains. *Artif. Intell.*, 101(1 2):99 134, 1998.

[7] M. L. Littman. Markov games as a framework for multi-agent reinforcement learning. In *Proc. ICML-1994*, pp. 157 163.

[8] R. Reiter. *Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems*. 2001.