Adaptive Multi-Agent Programming in GTGolog

Alberto Finzi [1 2] and **Thomas Lukasiewicz** [2 1]

Abstract. We present a novel approach to adaptive multi-agent programming, which is based on an integration of the agent programming language GTGolog with adaptive dynamic programming techniques. GTGolog combines explicit agent programming in Golog with game-theoretic multi-agent planning in stochastic games. In GTGolog, the transition probabilities and reward values of the domain must be provided with the model. The adaptive generalization of GTGolog proposed here is directed towards letting the agents themselves explore and adapt these data. We use high-level programs for the generation of both abstract states and optimal policies.

1 INTRODUCTION

We present a novel approach to adaptive multi-agent programming, which is based on an integration of GTGolog [5] with multi-agent reinforcement learning as in [7]. We use high-level programs for both generating abstract states and learning policies over these abstract states. The generation of abstract states exploits the structured representation of the domain in a basic action theory in the situation calculus (SC) [9], along with the high-level control knowledge in a Golog program [9]. A learning process then incrementally adapts the model to the executive context and instantiates the partially specified behavior. To our knowledge, this is the first adaptive approach to Golog interpreting. Differently from classical Golog [9, 3], here the interpreter generates not only complex sequences of actions, but also an abstract state space for each machine state. Similarly to [2, 6], we rely on the SC machinery for state abstraction, but in our system the state generation depends on the program structure. Here, we can take advantage from the deep integration between the SC action theory and Golog programs. Similarly to hierarchical reinforcement learning [4, 1], the choice points of partially specified hierarchical programs are instantiated through reinforcement learning and dynamic programming constrained by the program structure.

2 ADAPTIVE GTGOLOG

In this section, we present AGTGolog for $n \geq 2$ agents. To introduce our framework, we refer to the following scenario inspired by [8].

Example 2.1 We have a game field which is a grid of 9×9 positions, and two agents, $\boldsymbol{a}$ and $\boldsymbol{o}$, can move one step in the N, S, E,W directions, or remain stationary. Each of them can pick up wood or gold when at a forest or goldmine, respectively, and drop these resources at its base. Each action can fail resulting in a stationary move. A reward is received whenever a unit of wood or gold is brought back to the base of an agent. Any carried object drops when two agents collide. The game is zero-sum, hence, after each move an agent receives a reward consisting of the reward of its action minus the reward of the adversary action.

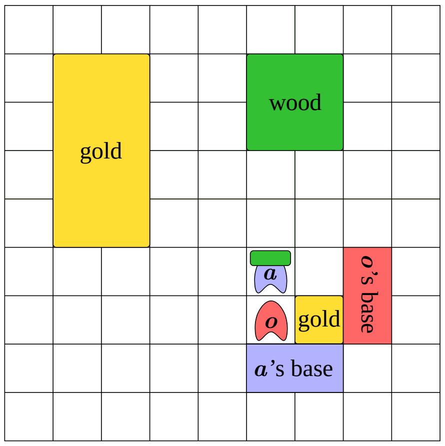

Figure 1. Example Domain

Domain Theory of AGTGolog. A *domain theory* of AGTGolog $DT = (AT, ST, OT)$ consists of a basic action theory AT, a *stochastic theory* ST, and an *optimization theory* OT, defined in the SC.

We assume two zero-sum competing agents $\boldsymbol{a}$ and $\boldsymbol{o}$ (called *agent* resp. *opponent*, where the former is under our control, while the latter is not). A *two-player action* is either an action a for agent $\boldsymbol{a}$, or an action b for agent $\boldsymbol{o}$, or two parallel actions $a \| b$, one for each agent. E.g., $move(\boldsymbol{a}, N)$, $move(\boldsymbol{o}, W)$, and $move(\boldsymbol{a}, N) \| move(\boldsymbol{o}, W)$ are two-player actions. The basic action theory AT represents a deterministic dynamic domain through fluent predicates, e.g., $at(q, x, y, s)$ and $holds(\alpha, o, s)$, which are defined by an FOL theory for the initial situation S_0 and *successor state axioms*, e.g.,

$$holds(\alpha, o, do(a, s)) \equiv holds(\alpha, o, s) \wedge \neg dropping(\alpha, a) \wedge \neg collision(a, s) \vee pickingUp(\alpha, o, a).$$

It also encodes *action preconditions*, one for each action, e.g.,

$$Poss(pickUpS(\alpha), s) \equiv \neg \exists x\, (holds(\alpha, x, s)).$$

Given AT, we can specify ST. As usual in the SC [3], stochastic actions are expressed by a finite set of deterministic actions represented by AT. When a stochastic action is executed, then "nature" chooses and executes with a certain probability exactly one of its deterministic actions. E.g., we introduce the stochastic action $moveS(\alpha, m)$ (agent α executes $m \in \{N, S, E, W, stand\}$), associated with the deterministic actions $move(\alpha, m)$ and $move(\alpha, stand)$ representing success and failure, respectively. To encode probabilities in ST (and rewards in OT), we use *state formulas* and *state partitions*. A *state formula* over $\vec{x}, s$ is an SC formula $\phi(\vec{x}, s)$ in which all predicate symbols are fluents, and the only free variables are the non-situation variables $\vec{x}$ and the situation variable s. A *state partition* over $\vec{x}, s$ is a nonempty set of state formulas $P(\vec{x}, s) = \{\phi_i(\vec{x}, s) \mid i \in \{1, \ldots, m\}\}$ such that (i) $\forall \vec{x}, s\, (\phi_i(\vec{x}, s) \Rightarrow \neg \phi_j(\vec{x}, s))$ is valid

[1] Inst. für Informationssysteme, TU Wien, Favoritenstr. 9-11, A-1040 Wien.
[2] DIS, Università di Roma "La Sapienza", Via Salaria 113, I-00198 Roma.

for all $i, j \in \{1, \ldots, m\}$ with $j > i$, (ii) $\forall \vec{x}, s \bigvee_{i=1}^{m} \phi_i(\vec{x}, s)$ is valid, and (iii) every $\exists \vec{x}, s\, \phi_i(\vec{x}, s)$ is satisfiable. For state partitions P_1 and P_2, we define $P_1 \otimes P_2 = \{\psi_1 \wedge \psi_2 \mid \psi_1 {\in} P_1,\ \psi_2 {\in} P_2,\ \psi_1 \wedge \psi_2 \neq \bot\}$.

Given a stochastic action a and an associated component n, we specify a state partition $P_{pr}^{a,n}(\vec{x}, s) = \{\phi_j^{a,n}(\vec{x}, s) \mid j \in \{1, \ldots, m\}\}$ to group together situations s with common p such that "nature" chooses n in s with probability p, denoted $prob(a(\vec{x}), n(\vec{x}), s) = p$:

$$\exists p_1, \ldots, p_m \,(\textstyle\bigwedge_{j=1}^{m} (\phi_j^{a,n}(\vec{x}, s) \Leftrightarrow prob(a(\vec{x}), n(\vec{x}), s) = p_j))\,.$$

In our example, we assume $P_{pr}^{a,n} = \{\top\}$ for each action-component pair, e.g., $\exists p\,(prob(pickUpS(\alpha), pickUp(\alpha), s) = p)$.

As for the optimization theory OT, for every two-player action a, we specify a state partition $P_{rw}^{a}(\vec{x}, s) = \{\phi_k^{a}(\vec{x}, s) \mid k \in \{1, \ldots, q\}\}$ to group together situations s with common r such that $a(\vec{x})$ and s assign the reward r to agent $\boldsymbol{a}$, denoted $reward(a(\vec{x}), s) = r$:

$$\exists r_1, \ldots, r_q \,(\textstyle\bigwedge_{k=1}^{q} (\phi_k^{a}(\vec{x}, s) \Leftrightarrow reward(a(\vec{x}), s) = r_k)\,.$$

In our domain, we define a zero-sum reward function as follows:

$$\begin{aligned} reward(\alpha, a, s) = r \equiv \exists r_\alpha\, (rewAg(\alpha, a, s) = r_\alpha \wedge \\ \exists \beta, r_\beta\, (rewAg(\beta, a, s) = r_\beta \wedge r = r_\alpha - r_\beta)), \\ \exists r_1, \ldots, r_m \,(\textstyle\bigwedge_{j=1}^{m} \phi_j^{\alpha,a}(s) \Leftrightarrow rewAg(\alpha, a, s) = r_j). \end{aligned}$$

Here, $\phi_j^{\alpha,a}(\vec{x}, s)$ belongs to $P_{rw}^{\alpha,a}$. Moreover, a utility function associates with every reward v and success probability pr a real-valued utility $utility(v, pr)$. We assume that $utility(v, 1) = v$ and $utility(v, 0) = 0$ for all v. E.g., $utility(v, pr) = v \cdot pr$.

Syntax of AGTGolog. AGTGolog has the same syntax as standard GTGolog. Given the actions in the domain theory DT, a *program* p has one of the following forms:

- *Deterministic or stochastic action:* α. Do α.
- *Nondeterministic action choice of $\boldsymbol{a}$:* **choice**$(\boldsymbol{a}\colon a_1 | \ldots | a_n)$. Do an optimal action (for agent $\boldsymbol{a}$) among $a_1, \ldots, a_n$.
- *Nondeterministic action choice of $\boldsymbol{o}$:* **choice**$(\boldsymbol{o}\colon o_1 | \ldots | o_m)$. Do an optimal action (for agent $\boldsymbol{o}$) among $o_1, \ldots, o_m$.
- *Nondeterministic joint action choice:* **choice**$(\boldsymbol{a}\colon a_1 | \ldots | a_n) \parallel$ **choice**$(\boldsymbol{o}\colon o_1 | \ldots | o_m)$. Do any action $a_i \| o_j$ with an optimal probability $\pi_{i,j}$.
- *Test action*: $\phi?$. Test the truth of ϕ in the current situation.
- *Action sequence*: $p_1; p_2$. Do p_1 followed by p_2.
- *Nondeterministic choice of two programs:* $(p_1 \mid p_2)$. Do p_1 or p_2.
- *Nondeterministic choice of an argument:* $\pi x\,(p(x))$. Do any $p(x)$.
- *Conditionals*, *while-loops*, and *procedures, including recursion*.

E.g., in our example, we define the procedure $tryToPickUp$:

$$\begin{aligned} &\mathbf{proc}(tryToPickUp, \mathbf{choice}(\boldsymbol{a}\colon pickUpS(\boldsymbol{a}) | moveS(\boldsymbol{a}, stand)) \| \\ &\quad \mathbf{choice}(\boldsymbol{o}\colon pickUpS(\boldsymbol{o}) | moveS(\boldsymbol{o}, stand))). \end{aligned}$$

State Partition Generation. Given an AGTGolog program p, a *machine state* consists of a subprogram p' of p and a number of steps to go h. A *joint state* (ϕ, p, h) consists of a state formula ϕ and a machine state (p, h). Every machine state (p, h) is associated with a state partition, denoted $SF(p, h) = \{\phi_1(\vec{x}, s), \ldots, \phi_m(\vec{x}, s)\}$. E.g., for *null program* or *zero horizon* (base case):

$$SF(Nil, h) = SF(p', 0) = \{\top\}.$$

For *deterministic first program action*, we have:

$$\begin{aligned} SF(a; p', h) = P_{rw}^{a}(\vec{x}, s) \otimes \{Regr(\phi(\vec{x}, do(a, s)) \wedge Poss(a, s) \mid \\ \phi(\vec{x}, s) \in SF(p', h{-}1)\} \cup \{\neg Poss(a, s)\} - \{\bot\}. \end{aligned}$$

For $p = tryToPickUp\,; carryToBase$ and $h = 3$, each program state (e.g., $(p, 3)$ or $(p_1, 2)$, with $p_1 = carryToBase$) is associated with a suitable state partition, e.g., $SF(p, 3)$ or $SF(p_1, 2)$.

Learning Algorithm. For each joint state σ composed of a machine state (p, h) and a state formula $\phi(\vec{x}, s) \in SF(p, h)$, the learning algorithm generates the best policy $\pi(\sigma)$ for the agent $\boldsymbol{a}$. The policy is obtained from (p, h) by replacing every single-agent choice in p by a single action, and every multi-agent choice by a collection of probability distributions $\pi(\sigma)$, one over the actions of each agent. We deploy a hierarchical version of Q-learning described in Figure 2.

```
Algorithm $Learn(p, h)$
Input: GTGolog program $p$ and finite horizon $h$.
Output: optimal policy $\pi(\phi, p, h)$ for all $\phi \in SF(p, h)$.
  1. $\alpha := 1$;
  2. for each joint state $\sigma$ do $\langle v, pr \rangle(\sigma) := \langle 1, 1 \rangle$;
  3. repeat
  4.    estimate $\phi \in SF(p, h)$;
  5.    $Update(\phi, p, h)$;
  6.    $\alpha := \alpha \cdot decay$
  7. until $\alpha < \varepsilon$;
  8. return $(\pi(\phi, p, h))_{\phi \in SF(p,h)}$.
```

Figure 2. Algorithm $Learn(p, h)$

We initialize the learning rate α (decaying at each cycle according to $decay$) and the variables $\langle v, pr \rangle(\cdot)$ representing the current value of the v-function in a joint state. At each cycle, the current state $\phi \in SF(p, h)$ is estimated. Then, from the joint state $\sigma = (\phi, (p, h))$, the procedure $Update(\phi, p, h)$ updates the values $\langle v, pr \rangle(\sigma)$ during an execution of the program p with horizon h. $Update(\phi, p, h)$ is inductively defined w.r.t. the structure of the program. E.g., if $p = a\,;\, p'$, where a is deterministic and executable in ϕ, then $Update(\phi, p, h)$ executes a and gets $reward$; upon the $Update(do(a, \phi), p', h{-}1)$ execution, we have the updates $\langle v, pr \rangle(\sigma) := \langle (1 - \alpha) \cdot v(\sigma) + \alpha \cdot (reward + \gamma \cdot v(do(a, \phi), p', h{-}1)), pr(do(a, \phi), p', h{-}1) \rangle$ and $\pi(\sigma) := a\,; \pi'(do(a, \phi), p', h{-}1)$. If $p =$ **choice**$(\boldsymbol{a}\colon a_1 | \ldots | a_n) \parallel$ **choice**$(\boldsymbol{o}\colon o_1 | \ldots | o_m); p'$, then the update step is associated with a Nash equilibrium (as in [7]): $(\pi_{\boldsymbol{a}}, \pi_{\boldsymbol{o}}) := selectNash(\{r_{i,j} = utility(\langle v, pr \rangle(\phi, \boldsymbol{a}{:}a_i \| \boldsymbol{o}{:}o_j; p', h)) \mid i, j\}$; where $\langle v, pr \rangle(\sigma) := \sum_{i=1}^{n} \sum_{j=1}^{m} \pi_{\boldsymbol{a}}(a_i) \cdot \pi_{\boldsymbol{o}}(o_j) \cdot \langle v, pr \rangle(\phi, \boldsymbol{a}{:}a_i \| \boldsymbol{o}{:}o_j; p', h))$. E.g., in our domain, we run the learning algorithm to instantiate a policy for the program $p = tryToPickUp; carryToBase$, with horizon 3, i.e., $Learn(p, 3)$. The agent runs several times p (with horizon 3) playing against the opponent until the learning ends and the variables $\langle v, pr \rangle$ are stabilized for each joint state of the program p.

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

REFERENCES

[1] D. Andre and S. J. Russell, 'State abstraction for programmable reinforcement learning agents', in *Proc. AAAI-2002*, pp. 119–125.

[2] C. Boutilier, R. Reiter, and B. Price, 'Symbolic dynamic programming for first-order MDPs', in *Proc. IJCAI-2001*, pp. 690–700.

[3] 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.

[4] T. G. Dietterich, 'The MAXQ method for hierarchical reinforcement learning', in *Proc. ML-1998*, pp. 118–126.

[5] A. Finzi and T. Lukasiewicz, 'Game-theoretic agent programming in Golog', in *Proc. ECAI-2004*, pp. 23–27.

[6] C. Gretton and S. Thiebaux, 'Exploiting first-order regression in inductive policy selection', in *Proc. UAI-2004*, pp. 217–225.

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

[8] B. Marthi, S. J. Russell, D. Latham, and C. Guestrin, 'Concurrent hierarchical reinforcement learning', in *Proc. IJCAI-2005*, pp. 779–785.

[9] R. Reiter, *Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems*, MIT Press, 2001.