I N F S Y S
R E S E A R C H
R E P O R T

INSTITUT FÜR INFORMATIONSSYSTEME

ABTEILUNG WISSENSBASIERTE SYSTEME

GAME-THEORETIC AGENT PROGRAMMING IN GOLOG UNDER PARTIAL OBSERVABILITY

ALBERTO FINZI THOMAS LUKASIEWICZ

INFSYS RESEARCH REPORT 1843-05-02

APRIL 2005

Institut für Informationssysteme
Abtg. Wissensbasierte Systeme
Technische Universität Wien
Favoritenstraße 9-11
A-1040 Wien, Austria
Tel: +43-1-58801-18405
Fax: +43-1-58801-18493
sek@kr.tuwien.ac.at
www.kr.tuwien.ac.at

TECHNISCHE UNIVERSITÄT WIEN

INFSYS RESEARCH REPORT
INFSYS RESEARCH REPORT 1843-05-02, APRIL 2005

GAME-THEORETIC AGENT PROGRAMMING IN GOLOG UNDER PARTIAL OBSERVABILITY

(PRELIMINARY VERSION, APRIL 29, 2005)

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

Abstract. In this paper, we present the agent programming language POGTGolog, which 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). It is a generalization of the agent programming language GTGolog by partial observability. 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. We illustrate the usefulness of this approach along a rugby example.

[1]Institut für Informationssysteme, Technische Universität Wien, Favoritenstraße 9-11, A-1040 Vienna, Austria; e-mail: lukasiewicz@kr.tuwien.ac.at.

[2]Dipartimento di Informatica e Sistemistica, Università di Roma "La Sapienza", Via Salaria 113, I-00198 Rome, Italy; e-mail: {finzi, lukasiewicz}@dis.uniroma1.it.

Acknowledgements: This work has been supported by the Austrian Science Fund Project P18146-N04 and by a Heisenberg Professorship of the German Research Foundation.

Copyright © 2005 by the authors

Contents

1 Introduction **1**

2 Preliminaries **2**
2.1 Situation Calculus and Golog. 2
2.2 Normal Form Games. 3
2.3 Partially Observable Stochastic Games. 4

3 Partially Observable GTGolog **5**
3.1 Domain Theory. 5
3.2 Syntax. 6
3.3 Semantics. 7
3.4 Special Cases. 9

4 Example **9**

5 Representation and Optimality Results **12**

6 Summary and Outlook **12**

1 Introduction

In the recent years, the development of controllers for autonomous agents has become increasingly important in AI. One way of designing such controllers is the programming approach, where a control program is specified through a language based on high-level actions as primitives. Another way is the planning approach, where goals or reward functions are specified and the agent is given a planning ability to achieve a goal or to maximize a reward function.

An integration of both approaches has recently been proposed through the language DTGolog (Boutilier *et al.* 2000), which integrates explicit agent programming in Golog [14] with decision-theoretic planning in (fully observable) Markov decision processes (MDPs) [13]. It allows for partially specifying a control program in a high-level language as well as for optimally filling in missing details through decision-theoretic planning. DTGolog can thus be seen as a decision-theoretic extension to Golog, where choices left to the agent are made by maximizing expected utility. From a different perspective, it can also be seen as a formalism that gives advice to a decision-theoretic planner, since it naturally constrains the search space.

Such integrations of logical formalisms for agent programming with decision-theoretic planning are also appealing from the perspective of decision-theoretic planning, since they allow for (see especially [17, 3, 5]) (i) compactly representing decision-theoretic planning problems without explicitly referring to atomic states and state transitions, (ii) exploiting such compact representations for efficiently solving large-scale problems, and (iii) nice properties such as *modularity* (parts of the specification can be easily added, removed, or modified) and *elaboration tolerance* (solutions can be easily reused for similar problems with few or no additional effort).

In DTGolog, the model of the world essentially consists of a single agent controlled by a DTGolog program and the environment summarized in "nature". But in realistic applications, one often encounters multiple agents, which may compete or cooperate with each other. Here, the optimal actions of one agent generally depend on the actions of all the other agents. In particular, there is a bidirectional dependence between the actions of two agents, which generally makes it inappropriate to model enemies and friends of the controlled agent simply as a part of "nature". This is the motivation behind GTGolog (Finzi & Lukasiewicz 2004), which is a generalization of DTGolog that integrates agent programming in Golog with multi-agent planning in (fully observable) stochastic games [11], also called Markov games [15, 8].

Example 1 Consider a rugby player a, who is deciding his next n moves and wants to cooperate with a teammate b. He has to deliberate about if and when it is worth to pass the ball. His options can be encoded by a high-level program:

$$\begin{array}{l}\mathbf{proc}(move(n), \mathbf{if}(HaveBall(a) \wedge n > 0),\\ \pi(x, \pi(y, \mathbf{choice}(a\colon moveTo(x) | pass(b)) \| \\ \qquad \mathbf{choice}(b\colon moveTo(y) | receive(a)))); move(n-1)).\end{array}$$

That is, while a is the ball-owner and $n > 0$, the two agents perform a parallel action choice in which a (resp., b) can either go somewhere or pass (resp., receive) the ball. Here, the actions' preconditions and effects are to be formally specified in a suitable action theory. Given this high-level program and the action theory for a and b, the program interpreter then fills in the best moves for a and b, reasoning about the two agents' possible interactions. □

A crucial aspect of real-world environments is that they are in general only partially observable, due to noisy and inaccurate sensors, or because some relevant parts of the environment simply cannot be sensed.

Both DT- and GTGolog, however, assume full observability, and they have not been generalized to the partially observable case so far.

In this paper, we try to fill this gap. We present the language POGTGolog, which extends GTGolog and thus also DTGolog by partial observability. It 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) [6]. More precisely, we assume that planning and control are centralized as follows. All agents transmit their local belief states and/or observations to a central agent, which then computes and returns the optimal local action for each agent. Under this assumption, finite-horizon Nash equilibria can be characterized by finite-horizon value iteration as in fully observable stochastic games. The main contributions are as follows:

- We define the language POGTGolog, which is a partially observable generalization of GTGolog. It integrates explicit agent programming in Golog with game-theoretic multi-agent planning in special POSGs.
- 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 special POSGs, and computing a Nash equilibrium.
- We show that POGTGolog generalizes its special class of POSGs. Furthermore, we show that the POGTGolog interpreter is optimal in the sense that it computes a Nash equilibrium of POGTGolog programs.
- A robotic rugby example gives evidence of the usefulness of our approach in realistic applications.

Note that detailed proofs of all results are in the extended paper.

2 Preliminaries

In this section, we recall the basic concepts of the situation calculus and Golog, normal form games, and POSGs.

2.1 Situation Calculus and Golog.

The situation calculus [9, 14] is a first-order language for representing dynamic domains. Its main ingredients are *actions*, *situations*, and *fluents*. An *action* is a first-order term of the form $a(\vec{u})$, where a is an action function symbol and $\vec{u}$ are its arguments. For example, $moveTo(pos)$ may represent the action of moving to position pos. A *situation* is a first-order term encoding a sequence of actions. It is either a constant symbol or of the form $do(a, s)$, where a is an action and s is a situation. The constant symbol S_0 is the *initial situation* and represents the empty sequence, while $do(a, s)$ encodes the sequence obtained from executing a after the sequence of s. For example, $do(moveTo(pos2), do(moveTo(pos1), S_0))$ represents the sequence of actions $moveTo(pos1)$, $moveTo(pos2)$. A *fluent* represents a world or agent property that may change when executing an action. It is a predicate symbol whose most right argument is a situation. For example, $at(pos, s)$ may express that an agent is at position pos in situation s. In the situation calculus, a dynamic domain is encoded as a *basic action theory* $BAT = (\Sigma, \mathcal{D}_{S_0}, \mathcal{D}_{ssa}, \mathcal{D}_{una}, \mathcal{D}_{ap})$, where:

- Σ is the set of foundational axioms for situations;

- $\mathcal{D}_{una}$ is the set of *unique name axioms for actions*, saying that different action terms stand for different actions;

- $\mathcal{D}_{S_0}$ is a set of first-order formulas describing the *initial state of the domain* (represented by S_0). For example, $at(a_1, 1, 2, S_0) \wedge at(a_2, 3, 4, S_0)$ may express that the agent a_1 (resp., a_2) is initially at the position $(1, 2)$ (resp., $(3, 4)$);

- $\mathcal{D}_{ssa}$ is the set of *successor state axioms* [14]. For each fluent $F(\vec{x}, s)$, it contains an axiom of the form $F(\vec{x}, do(a, s)) \equiv \Phi_F(\vec{x}, a, s)$, where $\Phi_F(\vec{x}, a, s)$ is a formula with free variables among $\vec{x}, a, s$. These axioms specify the truth of the fluent F in the next situation $do(a, s)$ in terms of the current situation s, and are a solution to the frame problem (for deterministic actions). For example,

$$\begin{array}{l} at(o, x, y, do(a, s)) \equiv a = moveTo(o, x, y) \vee \\ \quad at(o, x, y, s) \wedge \neg(\exists x', y')a = moveTo(o, x', y') \end{array} \tag{1}$$

  may express that the object o is at (x, y) in $do(a, s)$ iff it is moved there in s, or already there and not moved away in s;

- $\mathcal{D}_{ap}$ is the set of *action precondition axioms*. For each action a, it contains an axiom $Poss(a(\vec{x}), s) \equiv \Pi(\vec{x}, s)$, which characterizes the preconditions of action a. For example, $Poss(moveTo(o, x, y), s) \equiv \neg(\exists o')at(o', x, y, s)$ may express that it is possible to move the object o to (x, y) in s iff no other object o' is at (x, y) in s.

Golog is an agent programming language that is based on the situation calculus. It allows for constructing complex actions from the primitive actions defined in a basic action theory BAT, where standard (and not so-standard) Algol-like control constructs can be used, in particular, (i) action sequences: $p_1; p_2$, (ii) tests: $\phi?$, (iii) nondeterministic action choices: $p_1|p_2$, (iv) nondeterministic choices of action argument: $(\pi x).p(x)$, and (v) conditionals, while loops, and procedure calls. An example of a Golog program is

$$\textbf{while } \neg at(a_1, 1, 2) \textbf{ do } (\pi x, y)moveTo(a_1, x, y).$$

Intuitively, the nondeterministic choice $(\pi x, y)moveTo(a_1,\ x, y)$ is iterated until the agent a_1 is at the position $(1, 2)$.

The semantics of a Golog program δ is specified by a situation-calculus formula $Do(\delta, s, s')$, which encodes that s' is a situation which can be reached from s by executing δ. Thus, Do represents a macro expansion to a situation calculus formula. For example, the action sequence is defined through $Do(p_1; p_2, s, s') = \exists s''(Do(p_1, s, s'') \wedge Do(p_2, s'', s'))$. For more details on the core situation calculus and Golog, we refer the reader to [14].

2.2 Normal Form Games.

Normal form games [16] describe the possible actions of $n \geq 2$ agents and the rewards that the agents receive when they simultaneously execute one action each. For example, in *two-finger Morra*, two players E and O simultaneously show one or two fingers. Let f be the total numbers of fingers shown. If f is odd, then O gets f dollars from E, and if f is even, then E gets f dollars from O. Formally, a *normal form game* $G = (I, (A_i)_{i \in I}, (R_i)_{i \in I})$ consists of a set of *agents* $I = \{1, \ldots, n\}$, $n \geq 2$, a nonempty finite set of *actions* A_i for each agent $i \in I$, and a *reward function* $R_i \colon A \to \mathbf{R}$ for each agent $i \in I$, which associates with every *joint action* $a \in A = \times_{i \in I} A_i$ a *reward* $R_i(a)$ to i.

A pure (resp., mixed) strategy specifies which action an agent should execute (resp., which actions an agent should execute with which probability). Formally, a *pure strategy* for agent $i \in I$ is any action $a_i \in A_i$. A *pure strategy profile* is any joint action $a \in A$. If the agents play a, then the *reward* to agent $i \in I$ is $R_i(a)$. A *mixed strategy* for agent $i \in I$ is any probability distribution π_i over A_i. A *mixed strategy profile* $\pi = (\pi_i)_{i \in I}$ consists of a mixed strategy π_i for each agent $i \in I$. If the agents play π, then the *expected reward* to agent $i \in I$, denoted $\mathbf{E}[R_i(a) \mid \pi]$ (or $R_i(\pi)$), is defined as $\sum_{a=(a_j)_{j \in I} \in A} R_i(a) \cdot \Pi_{j \in I} \pi_j(a_j)$.

We are especially interested in mixed strategy profiles π, called Nash equilibria, where no agent has the incentive to deviate from its part, once the other agents play their parts. Formally, a mixed strategy profile $\pi = (\pi_i)_{i \in I}$ is a *Nash equilibrium* for G iff for every agent $i \in I$, it holds that $R_i(\pi'_i \circ \pi_{-i}) \leq R_i(\pi)$ for every mixed strategy π'_i, where $\pi'_i \circ \pi_{-i}$ is obtained from π by replacing π_i by π'_i. Every normal form game G has at least one Nash equilibrium among its mixed (but not necessarily pure) strategy profiles, and many have multiple Nash equilibria. A *Nash selection function* f associates with every normal form game G a unique Nash equilibrium $f(G)$. The expected reward to agent $i \in I$ under $f(G)$ is denoted by $v_f^i(G)$.

2.3 Partially Observable Stochastic Games.

Partially observable stochastic games (POSGs) generalize normal form games, POMDPs [7], and DEC-POMDPs [4, 12, 10]. We assume that planning and control are centralized as follows. There is a central agent, which (i) knows the local belief state of every other agent, (ii) computes and sends them their optimal local actions, and (iii) thereafter receives their local observations. Hence, we assume a transmission of local belief states and local observations to a central agent from all other agents, and of the optimal local actions in the reverse direction. Under this assumption, POSGs can be translated into belief state stochastic games.

Roughly, a POSG consists of a set of states S, a normal form game for each state $s \in S$, a set of joint observations of the agents O, and a transition function that associates with every state $s \in S$ and joint action of the agents $a \in A$ a probability distribution on all combinations of next states $s' \in S$ and joint observations $o \in O$. Formally, a *partially observable stochastic game (POSG)* $G = (I, S, (A_i)_{i \in I}, (O_i)_{i \in I}, P, (R_i)_{i \in I})$ consists of a set of *agents* $I = \{1, \ldots, n\}$, $n \geq 2$, a nonempty finite set of *states* S, two nonempty finite sets of *actions* A_i and *observations* O_i for each agent $i \in I$, a transition function $P \colon S \times A \to PD(S \times O)$, which associates with every state $s \in S$ and joint action $a \in A = \times_{i \in I} A_i$ a probability distribution over $S \times O$, where $O = \times_{i \in I} O_i$, and a *reward function* $R_i \colon S \times A \to \mathbf{R}$ for each agent $i \in I$, which associates with every state $s \in S$ and joint action $a \in A$ a *reward* $R_i(s, a)$ to agent i.

Since the actual state $s \in S$ of the POSG G is not fully observable, every agent $i \in I$ has a belief state b_i that associates with every state $s \in S$ the belief of agent i about s being the actual state. A *belief state* $b = (b_i)_{i \in I}$ of G consists of a probability function b_i over S for each agent $i \in I$. The POSG G then defines probabilistic transitions between belief states as follows. The new belief state $b^{a,o} = (b_i^{a,o})_{i \in I}$ after executing the joint action $a \in A$ in the belief state $b = (b_i)_{i \in I}$ and jointly observing $o \in O$ is given by:

$$b_i^{a,o}(s') = \sum_{s \in S} P(s', o \mid s, a) \cdot b_i(s) \;/\; P_b(b_i^{a,o} \mid b_i, a), \text{ where}$$
$$P_b(b_i^{a,o} \mid b_i, a) = \sum_{s' \in S} \sum_{s \in S} P(s', o \mid s, a) \cdot b_i(s)$$

is the probability of observing o after executing a in b_i. These probabilistic transitions define the fully observable stochastic game over belief states $G' = (I, B, (A_i)_{i \in I}, P_b, (R_i)_{i \in I})$, where B is the set of all belief states of G.

We next define finite-horizon pure and mixed policies and their rewards and expected rewards, respectively, using the above fully observable stochastic game over belief states. Assuming a finite horizon $H \geq 0$, a pure (resp., mixed) time-dependent policy associates with every belief state b of G and number of steps to

go $h \in \{0, \ldots, H\}$ a pure (resp., mixed) normal form game strategy. Formally, a *pure policy* α assigns to each belief state b and number of steps to go $h \in \{0, \ldots, H\}$ a joint action from A. A *mixed policy* is of the form $\pi = (\pi_i)_{i \in I}$, where every π_i assigns to each belief state b and number of steps to go $h \in \{0, \ldots, H\}$ a probability function $\pi_i[b, h]$ over A_i. The H-step reward (resp., expected H-step reward) for pure (resp., mixed) policies can now be defined as usual. In particular, the *expected H-step reward* to agent $i \in I$ under a start belief state $b{=}(b_i)_{i \in I}$ and the mixed policy π, denoted $G_i(H, b, \pi)$, is defined as

$$\begin{cases} \sum_{a \in A}(\Pi_{j \in I}\pi_j[b,0](a_j)) \cdot \sum_{s \in S} b_i(s) R_i(s,a) & \text{if } H = 0; \\ \sum_{a \in A}(\Pi_{j \in I}\pi_j[b,H](a_j)) \cdot (\sum_{s \in S} b_i(s) R_i(s,a) + \\ \qquad \sum_{o \in O} P(b_i^{a,o} \,|b_i, a) \cdot G_i(H{-}1, b^{a,o}, \pi)) & \text{otherwise.} \end{cases}$$

The notion of a finite-horizon Nash equilibrium for a POSG G is then defined as follows. A policy π is a *Nash equilibrium* of G under a belief state b iff for every agent $i \in I$, it holds that $G_i(H, b, \pi'_i \circ \pi_{-i}) \leq G_i(H, b, \pi_i \circ \pi_{-i})$ for all policies π'_i. A policy π is a *Nash equilibrium* of G iff it is a Nash equilibrium of G under every belief state b.

Nash equilibria of G can be characterized by finite-horizon value iteration from local Nash equilibria of normal form games as follows. Let f be an arbitrary Nash selection function for normal form games with the action sets $(A_i)_{i \in I}$. For every belief state $b = (b_i)_{i \in I}$ and number of steps to go $h \in \{0, \ldots, H\}$, let $G[b, h] = (I, (A_i)_{i \in I}, (Q_i[b, h])_{i \in I})$, where $Q_i[b, h](a)$ is defined as follows (for all $a \in A$ and $i \in I$):

$$\begin{cases} \sum_{s \in S} b_i(s) R_i(s,a) & \text{if } h = 0; \\ \sum_{s \in S} b_i(s) R_i(s,a) + \\ \qquad \sum_{o \in O} P(b_i^{a,o} \,|b_i, a) \cdot v_f^i(G[b^{a,o}, h{-}1]) & \text{otherwise.} \end{cases}$$

Let the mixed policy $\pi = (\pi_i)_{i \in I}$ for the POSG G be defined by $\pi_i(b, h) = f_i(G[b, h])$ for all agents $i \in I$, belief states b, and number of steps to go $h \in \{0, \ldots, H\}$. Then, π is a Nash equilibrium of G, and $G_i(H, b, \pi) = v_f^i(G[b, H])$ for every agent $i \in I$ and belief state b.

3 Partially Observable GTGolog

In this section, we present the language POGTGolog for $n \geq 2$ agents, which is a generalization of GTGolog that also allows for partial observability. We first describe the domain theory and the syntax of POGTGolog programs. We then define the semantics of POGTGolog programs.

3.1 Domain Theory.

POGTGolog programs are interpreted relative to a background action theory AT and a background optimization theory OT. The background action theory AT is an extension of the basic action theory BAT above, where we also allow for stochastic actions.

We assume a set of agents $I = \{1, \ldots, n\}$ with $n \geq 2$. A *single-agent action* is of form $i : a$, where $i \in I$ is an agent, and a is an ordinary action. A *multi-agent action* is of form $i_1 : a_1 \| \cdots \| i_k : a_k$, where every $i_j : a_j$ with $j \in \{1, \ldots, k\}$ is a single-agent action, and $i_1, \ldots, i_k$ are pairwise distinct. For example, $1 : move(o_1)$, $2 : move(o_2)$, $3 : move(o_3)$, and $1 : move(o_1) \parallel 2 : move(o_2) \parallel 3 : move(o_3)$ are multi-agent actions. Note that we use parallel actions as primitives to simplify the subsequent presentation. More complex parallel actions can be easily treated as well, using the concurrent version of the situation calculus [14].

Analogously to (Boutilier *et al.* 2000), we represent stochastic actions by means of a finite set of deterministic actions. Every action a is associated with a nonempty finite set of observations O_a, which are terms that represent pairwise exclusive and exhaustive properties of the world. When a stochastic action is executed, then with a certain probability "nature" executes exactly one of its deterministic actions and produces exactly one of its observations. We use the predicate $stochastic(a, s, n)$ to associate the stochastic action a with the deterministic action n in situation s, and we use the function $prob(a, n, s, o) = p$ to encode that when executing a in situation s, "nature" chooses n and produces the observation o with probability p. A stochastic action s is indirectly represented by providing a *successor state axiom* for every associated nature choice n. Hence, BAT is extended to a probabilistic setting in a minimal way. For example, consider the stochastic action $moveS(a, x, y)$ such that $stochastic(moveS(a, x, y), s, moveTo(a, x, y')) \equiv 0 \leq y' - y \leq 1$, that is, $moveS(a, x, y)$ can slide to $y + 1$. We specify the action $moveS$ by defining the precondition of $moveTo(a, x, y')$ and assuming some successor state axiom. Furthermore, to specify the probability distribution over the deterministic components, we then define

$$prob(moveS(a, x, y), moveTo(a, x, y), s, o_1) = 0.9 \text{ and}$$
$$prob(moveS(a, x, y), moveTo(a, x, y{+}1), s, o_2) = 0.1.$$

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 every agent $i \in I$, denoted $reward(i, a, s)$, for example, $reward(1, moveTo(a, x, y), s) = y$. The utility function maps every reward and success probability to a real-valued utility $utility(v, pr)$. We assume that $utility(v, 1) = v$ for all v. An example is $utility(v, pr) = v \cdot pr$. The utility function suitably mediates between the agent reward and the failure of actions due to unsatisfied preconditions.

To model partial observability, we define a new type of situations $b = (b_i)_{i \in I}$, called *belief state situations*, where every b_i is a set of pairs (s, p) consisting of an ordinary situation s and a real $p \in (0, 1]$, where all p sum up to 1, which represents the belief of agent i expressed as a probability distribution over ordinary situations. We extend concepts around actions from ordinary situations to belief state situations as follows. An action a is *executable* in a belief state $b = (b_i)_{i \in I}$, denoted $Poss(a, b)$, iff a is executable in s, denoted $Poss(a, s)$, for some $(s, p) \in b_i$ with $i \in I$ (action executability is associated with a probability value through the Golog primitive action definition; see below). We define $do(a, b) = (b'_i)_{i \in I}$ as follows. If a is a deterministic action, then $b'_i = \{(do(s, a), p/c) \mid (s, p) \in b_i, Poss(a, s)\}$, where $c = \sum_{(s,p) \in b_i : Poss(a,s)} p$, for all $i \in I$. If a is a stochastic action producing the observation $o \in O_a$, then b'_i is obtained from the union of all $\{(do(s, n), p \cdot p_n) \mid stochastic(a, s, n),\ prob(a, n, s, o) = p_n\}$ such that $(s, p) \in b_i$ and $Poss(a, s)$ by normalizing the probabilities to sum up to 1. The probability of observing $o \in O_a$ after executing the action a in b_i, denoted $prob(a, o, b_i)$, is the sum of all $prob(a, n, s, o) \cdot p$ such that $stochastic(a, s, n)$ and $(s, p) \in b_i$.

3.2 Syntax.

Given the multi-agent actions represented by the domain theory, *programs* p in POGTGolog are inductively built using the following constructs (where ϕ is a condition, p_1, p_2 are programs, and $\alpha, \ldots, \beta$ are multi-agent actions):

1. *Deterministic or stochastic action:* α.
2. *Nondeterministic choice:* $\alpha | \ldots | \beta$. Do α or $\ldots$ or β.
3. *Test action*: $\phi?$. Test ϕ's truth in the current situation.

4. *Nondeterministic choice of an argument.*
5. *Action sequence*: $p_1; p_2$. Do p_1 followed by p_2.
6. *Conditionals*: $\mathbf{if}\ \phi\ \mathbf{then}\ p_1\ \mathbf{else}\ p_2$.
7. *While loops*: $\mathbf{while}\ \phi\ \mathbf{do}\ p_1$.
8. *Nondeterministic iteration*: ${p_1}^*$. Do p_1 $n \geq 0$ times.
9. *Procedures, including recursion.*

In order to explicitly specify the choices of actions of the agents, for $J = \{j_1, \ldots, j_k\} \subseteq I$, we write $\mathbf{choice}(j_1 : a_{j_1,1} | \cdots |\ j_1 : a_{j_1,n_{j_1}}) \parallel \cdots \parallel \mathbf{choice}(j_k : a_{j_k,1} | \cdots | j_k : a_{j_k,n_{j_k}})$, which we also abbreviate as $\|_{j \in J}\mathbf{choice}(j : a_{j,1} | \cdots | j : a_{j,n_j})$, to denote $(j_1{:}a_{j_1,1} \| \cdots \| j_k{:}a_{j_k,1}) \mid \cdots \mid (j_1{:}a_{j_1,n_{j_1}} \| \cdots \| j_k{:}a_{j_k,n_{j_k}})$. Informally, the agents in J execute simultaneously one action each, namely, every $j \in J$ one from $\{a_{j,1}, \ldots, a_{j,n_j}\}$.

3.3 Semantics.

The semantics of a POGTGolog program p relative to AT and OT for a set of agents $I = \{1, \ldots, n\}$ is defined through the macro $DoG(p, b, h, \pi, v, pr)$. Here, we have as input the program p, a belief state $b = (b_i)_{i \in I}$, and a finite horizon $h \geq 0$. The macro DoG then determines a joint strategy π for all agents, along with its reward and success probability vectors $v = (v_i)_{i \in I}$ and $pr = (pr_i)_{i \in I}$, respectively, where v_i is the reward of π to agent i, and $pr_i \in [0, 1]$ is the success probability of π relative to agent i. Note that if p fails to terminate before the horizon h is reached, then it is stopped, and the best partial strategy is returned. Intuitively, our aim is to control the agents in I, which are provided with the optimal joint strategy π computed by DoG for the program p, and then execute their part of π. We define $DoG(p, b, h, \pi, v, pr)$ by induction as follows:

1. Zero horizon and null program:

$$DoG(p, b, 0, \pi, v, pr) =_{def} \pi{=}Nil \wedge \textstyle\bigwedge_{i \in I}(v_i{=}0 \wedge pr_i{=}1)$$
$$DoG(Nil, b, h, \pi, v, pr) =_{def} \pi{=}Nil \wedge \textstyle\bigwedge_{i \in I}(v_i{=}0 \wedge pr_i{=}1)$$

   Intuitively, p ends when it is null or at the horizon end.

2. Deterministic first program action:

$$\begin{array}{l} DoG(a; p, b, h, \pi, v, pr) =_{def} \\ \quad \neg Poss(a, b) \wedge \pi = Stop \wedge \textstyle\bigwedge_{i \in I} v_i = 0 \wedge \bigwedge_{i \in I} pr_i = 0 \vee \\ \quad \exists \pi', v', pr' \colon Poss(a, b) \wedge DoG(p, do(a, b), h{-}1, \pi', v', pr') \wedge \\ \quad \pi = a; \pi' \wedge v = v' + (reward(i, a, b_i))_{i \in I} \wedge \\ \quad pr = pr' \cdot (\sum_{(s,p) \in b_i \colon Poss(a,s)} p)_{i \in I} \end{array}$$

   Here, let $(s_i)_{i \in I}\ op\ (t_i)_{i \in I} = (s_i\ op\ t_i)_{i \in I}$ for $op \in \{+, \cdot\}$, and $reward(i, a, b_i) = \sum_{(s,p) \in b_i} p \cdot reward(i, a, s)$. Informally, if a is not executable, then p stops with success probability 0. As in [1], $Stop$ is a fictitious action of zero-cost, which stops the program execution. If a is executable, then the optimal execution of $a; p$ in the belief state b depends on that one of p in the belief state $do(a, b)$. Observe that a is executable with the probability $(\sum_{(s,p) \in b_i \colon Poss(a,s)} p)_{i \in I}$, which affects the overall program success probability pr.

3. Stochastic first program action (nature choice):

$$
\begin{array}{l}
DoG(a;p,b,h,\pi,v,pr) =_{def} \\
\quad \neg Poss(a,b) \wedge \pi = Stop \wedge \bigwedge_{i\in I} v_i = 0 \wedge \bigwedge_{i\in I} pr_i = 0 \vee \\
\quad \exists \pi_q, v_q, pr_q \colon Poss(a,b) \wedge \bigwedge_{q=1}^{k} DoG(n_q;p,b,h,n_q;\pi_q,v_q,pr_q) \wedge \\
\quad \pi = a; \textbf{for } q=1 \textbf{ to } k \textbf{ do if } o_q \textbf{ then } \pi_q \wedge \\
\quad v = \sum_{q=1}^{k} v_q \cdot (prob(a,o_q,b_i))_{i\in I} \wedge \\
\quad pr = \sum_{q=1}^{k} pr_q \cdot (prob(a,o_q,b_i) \sum_{(s,p)\in b_i : Poss(a,s)} p)_{i\in I}
\end{array}
$$

Here, n_q is the action associated with the stochastic action a under the observation o_q. The generated strategy is a conditional plan, where every possible stochastic action execution (i.e., every possible observation) is considered.

4. Nondeterministic first program action (choice of j):

$$
\begin{array}{l}
DoG(\textbf{choice}(j\colon a_1 | \cdots | a_k);p,b,h,\pi,v,pr) =_{def} \\
\quad \exists \pi_q, v_q, pr_q \colon \bigwedge_{q=1}^{k} DoG(i{:}a_q;p,b,h,\pi_q,v_q,pr_q) \wedge \\
\quad \pi = \textbf{for } q=1 \textbf{ to } k \textbf{ do if } \psi_q \textbf{ then } \pi_q \wedge \\
\quad utility(v_{m,j},pr_{m,j}) = \max_{q=1}^{k} utility(v_{q,j},pr_{q,j}) \wedge \\
\quad v = v_m \wedge pr = pr_m
\end{array}
$$

Here, $pr_q = (pr_{q,i})_{i\in I}$ and $v_q = (v_{q,i})_{i\in I}$. Informally, given several possible actions $a_1,\ldots,a_k$ for agent $j \in I$, the best action is the one with the best utility. The ψ_q's denote conditions (defined by *observability axioms*) that the other agents in I have to test to observe j's choice.

5. Nondeterministic first program action (choice of J):

$$
\begin{array}{l}
DoG(\|_{j\in J}\textbf{choice}(j\colon a_{j,1} | \ldots | a_{j,n_j});p,b,h,\pi,v,pr) =_{def} \\
\quad \exists \pi_a, v_a, pr_a, \pi\colon \\
\quad \bigwedge_{a\in A} DoG(\|_{j\in J} j{:}a_j;p,b,h,\|_{j\in J} j{:}a_j;\pi_a,v_a,pr_a) \wedge \\
\quad (\pi_j)_{j\in J} = selectNash(\{utility(v_a,pr_a)|_J \mid a \in A\}) \wedge \\
\quad \pi = \|_{j\in J}\pi_j; \textbf{for each } a \in A \textbf{ do if } \phi_a \textbf{ then } \pi_a \wedge \\
\quad v = \sum_{a\in A} v_a \cdot \Pi_{j\in J}\pi_j(a_j) \wedge \\
\quad pr = \sum_{a\in A} pr_a \cdot \Pi_{j\in J}\pi_j(a_j)
\end{array}
$$

Here, $A = \times_{j\in J}\{a_{j,1},\ldots,a_{j,n_j}\}$, $utility((s_i)_{i\in I},(t_i)_{i\in I}) = (utility(s_i,t_i))_{i\in I}$, and for $s = (s_i)_{i\in I}$ and $J \subseteq I$, denote by $s|_J$ the restriction of s to J. Informally, we first compute the optimal joint strategy for each possible combination of actions $a \in A$. Then, we compute a local Nash equilibrium $(\pi_j)_{j\in J}$ from a normal form game by the function $selectNash$. Here, each π_j is a probability distribution over $\{a_{j,1},\ldots,a_{j,n_j}\}$. The conditions ϕ_a with $a \in A$ are to observe what the agents have actually executed.

6. Test action:

$$
\begin{array}{l}
DoG(\phi?;p,b,h,\pi,v,pr) =_{def} \\
\quad \exists pr' \colon \phi[b] \wedge DoG(p,b,h,\pi,v,pr') \wedge pr = pr' \cdot prob(\phi[b]) \vee \\
\quad \neg\phi[b] \wedge \pi = Stop \wedge \bigwedge_{i\in I} v_i = 0 \wedge \bigwedge_{i\in I} pr_i = 0
\end{array}
$$

Here, $\phi[b]$ is true iff $\phi[s]$ is true for some $(s,p) \in b_i$ and $i \in I$, but it is associated with the probability $prob(\phi[b]) = (\sum_{(s,p)\,\in\, b_i\colon \phi[s]} p)_{i\in I}$, which is accounted into the overall program success probability.

7. The semantics of nondeterministic iterations, conditionals, while loops, procedures, argument selection, and associate sequential composition is defined as usual.

3.4 Special Cases.

POGTGolog is defined for the very general case of any $n \geq 2$ agents with arbitrary (i.e., not necessarily the same or zero-sum) rewards. These are two special cases:

- *One team of cooperative agents*: All agents have the same reward. If they all also have the same belief state, then POGTGolog models a POMDP. Otherwise, POGTGolog models a special kind of DEC-POMDP.
- *Two competing teams of cooperative agents*: Two agents of the same team have the same reward, while two agents of different teams have zero-sum rewards.

Observe that even if two agents have originally the same (resp., zero-sum) rewards, they may have different (resp., non-zero-sum) rewards in the local Nash computations of POGTGolog, because of different belief states.

4 Example

In this section, we consider a rugby example (see Fig. 1), which is adapted from Littman's soccer example in [8]. The rugby field is a 4×5 grid. We assume a team of agents $A = \{a_0, \ldots, a_p\}$ against a team $B = \{b_0, \ldots, b_q\}$, with the *captains* a_0 and b_0, respectively. 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, move down, move right, move left, no move, pass, and receive the ball, resp.). The ball is represented by an oval and also occupies a square. An agent is a *ball owner* iff it occupies the same square as the ball. The ball follows the moves of the ball owner, and we have a goal when the ball owner steps into the adversary goal. An agent can also pass the ball to another agent of the same team, but this is possible only if the receiving agent is not closer to the opposing end of the field than the ball, otherwise, an offside fault is called by the referee, and the ball possession goes to the captain of the opposing team. When the ball owner goes into the square occupied by the other agent, if the other agent stands, possession of ball changes. Thus, a good defensive maneuver is to stand where the other agent wants to go.

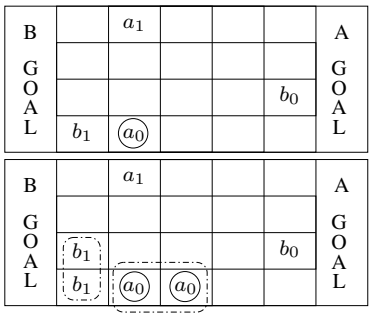

Figure 1: Rugby Example

We assume two agents for each team, that is, $A = \{a_0, a_1\}$ and $B = \{b_0, b_1\}$. To axiomatize the action theory 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, respectively) 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 situation s) defined by the following successor state axioms:

$$
\begin{aligned}
&at(\alpha, x, y, do(a, s)) \equiv (\exists x', y', m).at(\alpha, x', y', s) \wedge \\
&\quad moved(\alpha, a, m) \wedge (m = stand \wedge y' = y \vee m = N \wedge \\
&\qquad y' = y - 1 \vee m = S \wedge y' = y + 1) \wedge x = x' \vee \\
&\quad (m = E \wedge x' = x - 1 \vee m = W \wedge x' = x + 1) \wedge y' = y \vee \\
&\quad (\exists \beta).(m = passTo(\beta) \vee m = receive) \wedge y' = y \wedge x' = x\,; \\
&haveBall(\alpha, do(a, s)) \equiv (\exists \beta).haveBall(\beta, s) \wedge \\
&\quad (\alpha = \beta \wedge \neg(\exists \alpha').\neg cngBall(\alpha', a, s) \wedge rcvBall(\alpha', a, s) \vee \\
&\qquad \alpha_i \neq \beta \wedge (cngBall(\alpha_i, a, s) \vee rcvBall(\alpha_i, a, s))\,.
\end{aligned}
$$

Here, $moved(\alpha, a, m)$ is true iff m is the α action in a; $cngBall(\alpha, a, s)$ is true iff the ball possession changes to α after an action a in s (either in the case of an adversary block or in the case of an offside ball passage). The predicate $rcvBall(\alpha, a, s)$ is true iff the agent α receives the ball from the ball owner and it is in offside.

Once we have defined the deterministic components, we can introduce the stochastic actions $moveTo(\alpha_i, x)$ representing the agent attempt in doing x. We suppose that any agent (or opponent) α_i can succeed $moveTo(\alpha_i, x)$, and then the associated deterministic action a is executed, i.e., $moved(a, \alpha_i, x)$, or it can fail, and then no action is performed, i.e., $moved(a, \alpha_i, stand)$. After each $moveTo$ action, the agent can observe the presence of a team member in the direction of the movement, a player is visible only if not covered by another agent, e.g.,

$$
\begin{aligned}
&prob(moveTo(\alpha, x), s, a, observe(\alpha')) = p \equiv \\
&\quad (\exists y, p_1).moved(\alpha, a, y) \wedge (visible(\alpha, \alpha', a, s) \wedge \\
&\qquad (y = stand \wedge p_1 = 0.2 \vee y = x \wedge p_1 = 0.8 \wedge \\
&\qquad\quad p = p_1 \times 0.8) \vee (\neg visible(\alpha, \alpha', s) \wedge p = 0.0))\,; \\
&prob(moveTo(\alpha, x), s, a, observe(none)) = p \equiv \\
&\quad (\exists y, p_1).moved(\alpha, a, y) \wedge (y = stand \wedge p_1 = 0.2 \vee \\
&\qquad y = x \wedge p_1 = 0.8 \wedge (\exists \alpha').(visible(\alpha, \alpha', a, s) \wedge \\
&\qquad\quad p = p_1 \times 0.2) \vee (\forall \alpha').\neg visible(\alpha, \alpha', s) \wedge p = p_1))\,.
\end{aligned}
$$

Here, $visible$ is true if α can observe α' after the a execution from s. The observation set associated with the action $moveTo(\alpha_i, x)$ is $\{observe(\alpha_j), observe(none)\}$.

We now define the axioms in R. The zero-sum reward function among the two teams is represented by:

$$
\begin{aligned}
&reward(\alpha, s){=}r \equiv \\
&(\exists \alpha')goal(\alpha', s) \wedge (sameTeam(\alpha', \alpha) \wedge r{=}M \vee \\
&\quad \neg sameTeam(\alpha', \alpha) \wedge r{=}-M) \vee \\
&(\exists \alpha')\neg goal(\alpha', s) \wedge haveBall(\alpha', s) \wedge at(\alpha', x, y, s) \wedge \\
&(\exists r')evalPos(x, y, r') \wedge (sameTeam(\alpha, \alpha') \wedge r{=}r' \vee \\
&\quad \neg sameTeam(\alpha, \alpha) \wedge r{=}-r')\,.
\end{aligned}
$$

Here, the reward is high (M stands for a "big" integer), if a player of the same team scores a goal, and the reward depends on $evalPos(x, y, r)$, that is, the ball-owner position (roughly, r is high if the ball-owner belongs to the same team and is close to the adversary goal), otherwise.

Once we have defined the rules of the game, we can use POGTGolog to manage the different game situations. We can consider the scenario depicted in Fig. 1. We suppose a cooperative setting where the two opponents in B are static, and a_0 and a_1 have to coordinate themselves in order to score a goal. We assume that the captain a_0 has a complete view of the situation, and its belief state b_{a_0} coincides with the state illustrated in Fig. 1, upper part: There is only one situation s_1 with probability 1 such that $at(a_0, 2, 1, s_1)$, $at(a_1, 2, 4, s_1)$, $at(b_1, 1, 1, s_1)$, $at(b_0, 5, 2, s_1)$ (i.e., the captain of team B is very close to the goal of team A), and $haveBall(a_0, s_1)$. From the perspective of a_0, the goal seems quite done: a_0 can pass to a_1, which has a paved way towards the goal. Unfortunately, a_0 has to cooperate with a_1, whose vision of the game state is more confused (see Fig. 1, lower part): From a_1's point of view (i.e., belief state b_{a_1}), it could be either at $(1, 1)$ (a) or at $(1, 2)$ (b), and a_0 could be either at $(2, 1)$ (c) or at $(3, 1)$ (d). Hence, a_1 has a belief state b_{a_1} composed of four possible states with, e.g., the following probability distribution: $\{(s_{a,c}, 0.5)$, $(s_{a,d}, 0.3)$, $(s_{b,c}, 0.1)$, $(s_{b,d}, 0.1)\}$. Both a_0 and a_1 have to decide when (and if) it is worth to pass the ball, considering that if a_0 tries to pass while a_1 is in offside (e.g., in $s_{a,d}$ or $s_{b,d}$), then the ball goes to the captain of the adversary team (i.e., b_0) which is in a very good position. Given this scenario, the following high-level program represents a game schema which can be compiled to find a possible strategy:

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

Here, 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 (with or without the ball). Assuming a 4-steps horizon, an optimal instantiation of this schema is the strategy π such that $AT \cup OT \models DoG(schema, (b_{a_0}, b_{a_1}), 4, \pi, (v_1, v_2), (pr_1, pr_1))$. Assuming suitable values of M and k, more than one solution is possible. For a_0, the best strategy is to pass the ball as soon as possible, and the following

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

gives to a_1 three $moveTo(\boldsymbol{a}_1, E)$ attempts to achieve the touch-line. This is also a possible Nash equilibrium. From the standpoint of $\boldsymbol{a}_1$, instead, it is worth to spend a $moveTo(\boldsymbol{a}_1, S)$ to observe if the agent $\boldsymbol{a}_0$ is aligned trying to minimize the likelihood of a wrong passage; in this case, $\boldsymbol{a}_0$ is to delay the passage waiting for the move of $\boldsymbol{a}_1$. The associated strategy is another possible Nash equilibrium more favorable to the belief state of $\boldsymbol{a}_1$:

$$
\begin{array}{l}
stand \| moveTo(\boldsymbol{a}_0, S); \\
\quad [\mathbf{if}\ observe(\boldsymbol{a}_0)\ \mathbf{then}\ passTo(a_1) \| receive\,; \\
\qquad moveTo(\boldsymbol{a}_0, E) \| moveTo(\boldsymbol{a}_1, E)] \\
\quad [\mathbf{if}\ observe(none)\ \mathbf{then}\ moveTo(\boldsymbol{a}_0, E) \| moveTo(\boldsymbol{a}_1, E)]; \\
\qquad moveTo(\boldsymbol{a}_0, E) \| moveTo(\boldsymbol{a}_1, E); \\
\qquad moveTo(\boldsymbol{a}_0, E) \| moveTo(\boldsymbol{a}_1, E)\,.
\end{array}
$$

Since, in this context, there is more than one Nash equilibrium, the choice of the common strategy depends on the selection function embedded in the compiler algorithm.

5 Representation and Optimality Results

The following theorem shows that every POSG G can be encoded as a program p in POGTGolog such that DoG characterizes one of the finite-horizon Nash equilibria π of G along with the expected finite-step reward of π.

Theorem 2 *Let $G = (I, S, (A_i)_{i \in I}, (O_i)_{i \in I}, P, (R_i)_{i \in I})$ be a POSG, and let $H \geq 0$ be a horizon. Then, there exists an action theory AT, an optimization theory OT, and a POGTGolog program p relative to them such that π is a Nash equilibrium for G, where $\pi(b, h) = (\pi_1, \pi_2)$ is given by $DoG(p, b, h{+}1, \pi_1 \| \pi_2; \pi', v, pr)$ for every belief state b and steps to go $h \in \{0, \ldots, H\}$. Furthermore, for every $H \geq 0$ and belief state b, it holds that $G(H, b, \pi) = v$ is given by $DoG(p, b, H{+}1, \pi_1 \| \pi_2; \pi', v, pr)$.*

We next show that DoG produces optimal results. Given a finite horizon $H \geq 0$, a *strategy* π for a POGTGolog program p is obtained from the H-horizon part of p by replacing single-agent choices by single actions, and multi-agent choices by probability distributions over their actions. The notions of an *expected H-step reward $G(p, b, H, \pi)$*, with a belief state b, and of a finite-horizon *Nash equilibrium under a belief state b* can then be defined in a straightforward way as for POSGs. The next theorem shows that DoG is optimal in that it computes a Nash equilibrium under a given belief state b and its expected finite-step reward.

Theorem 3 *Let AT be an action theory, OT be an optimization theory, and p be a POGTGolog program relative to them. Let $DoG(p, b, h{+}1, \pi, v, pr)$ for a belief state b and $h \geq 0$. Then, π is a Nash equilibrium under the belief state b, and $utility(v, pr)$ is its expected h-step reward.*

6 Summary and Outlook

We have presented POGTGolog, which combines explicit agent programming in Golog with game-theoretic multi-agent planning in special POSGs. It 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. We have illustrated the usefulness of this approach along a rugby example.

An interesting topic of future work is to explore if similar languages can be defined for other classes of special (or even for general) POSGs or DEC-POMDPs.

References

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

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

[3] N. H. Gardiol and L. P. Kaelbling. Envelope-based planning in relational MDPs. In *Proceedings NIPS-2003*.

[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] C. Guestrin, D. Koller, C. Gearhart, and N. Kanodia. Generalizing plans to new environments in relational MDPs. In *Proceedings IJCAI-2003*, pp. 1003–1010.

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

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

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

[9] J. McCarthy and P. J. Hayes. Some philosophical problems from the standpoint of artificial intelligence. In *Machine Intelligence*, volume 4, pp. 463–502. Edinburgh University Press, 1969.

[10] R. Nair, M. Tambe, M. Yokoo, D. V. Pynadath, and S. Marsella. Taming decentralized POMDPs: Towards efficient policy computation for multiagent settings. In *Proceedings IJCAI-2003*, pp. 705–711. 2003.

[11] G. Owen. *Game Theory: Second Edition*. Academic Press, 1982.

[12] L. Peshkin, K.-E. Kim, N. Meuleau, and L. Pack Kaelbling. Learning to cooperate via policy search. In *Proceedings UAI-2000*, pp. 489–496.

[13] M. L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley, 1994.

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

[15] J. van der Wal. *Stochastic Dynamic Programming*, volume 139 of *Mathematical Centre Tracts*. Morgan Kaufmann, 1981.

[16] J. von Neumann and O. Morgenstern. *The Theory of Games and Economic Behavior*. Princeton University Press, 1947.

[17] S. W. Yoon, A. Fern, and B. Givan. Inductive policy selection for first-order MDPs. In *Proceedings UAI-2002*, pp. 569–576.