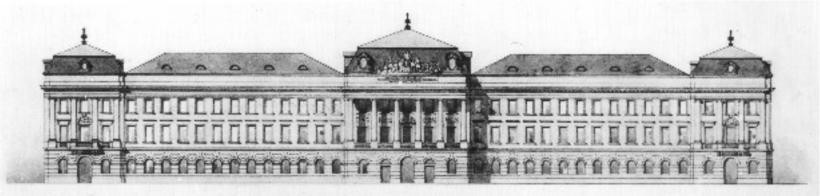
INFSYS
RESEARCH
REPORT



INSTITUT FÜR INFORMATIONSSYSTEME
ARBEITSBEREICH WISSENSBASIERTE SYSTEME

GAME-THEORETIC AGENT PROGRAMMING IN GOLOG

ALBERTO FINZI THOMAS LUKASIEWICZ

INFSYS RESEARCH REPORT 1843-04-02
APRIL 2007

Institut für Informationssysteme
AB 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

INFSYS RESEARCH REPORT

INFSYS RESEARCH REPORT 1843-04-02, APRIL 2007

GAME-THEORETIC AGENT PROGRAMMING IN GOLOG

APRIL 14, 2007

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

Abstract. We present the agent programming language GTGolog, which integrates explicit agent programming in Golog with game-theoretic multi-agent planning in stochastic games. GTGolog is a generalization of DTGolog to multi-agent systems consisting of two competing single agents or two competing teams of cooperative agents, where any two agents in the same team have the same reward, and any two agents in different teams have zero-sum rewards. In addition to being a language for programming agents in such multi-agent systems, GTGolog can also be considered as a new language for specifying games. GTGolog allows for defining a partial control program in a high-level logical language, which is then completed by an interpreter in an optimal way. To this end, we define a formal semantics of GTGolog programs in terms of Nash equilibria, and we specify a GTGolog interpreter that computes one of these Nash equilibria. We then show that the computed Nash equilibria can be freely mixed and that GTGolog programs faithfully extend (finite-horizon) stochastic games. Furthermore, we also show that under suitable assumptions, computing the Nash equilibrium specified by the GTGolog interpreter can be done in polynomial time. Finally, we also report on a first prototype implementation of a simple GTGolog interpreter.

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

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

Acknowledgements: This work was supported by the Austrian Science Fund Projects P18146-N04 and Z29-N04, by a Heisenberg Professorship of the German Research Foundation, and by the Marie Curie Individual Fellowship HPMF-CT-2001-001286 of the EU programme "Human Potential" (disclaimer: The authors are solely responsible for information communicated and the European Commission is not responsible for any views or results expressed).

Contents

1 Introduction

During the recent decades, the development of controllers for autonomous agents in real-world environments has become increasingly important in AI. One of the most crucial problems that we have to face here is uncertainty, both about the initial situation of the agent's world and about the results of the actions taken by the agent. One way of designing such controllers is based on logic-based languages and formalisms for reasoning about actions under uncertainty, where control programs and action theories are specified using high-level actions as primitives. Another way is based on approaches to classical planning under uncertainty or to decision-theoretic planning, 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. Both ways of designing controllers have certain advantages.

In particular, logic-based languages and formalisms for reasoning about actions under uncertainty (i) allow for compact representations without explicitly referring to atomic states and state transitions, (ii) allow for exploiting such compact representations for efficiently solving large-scale problems, and (iii) have the nice properties of *modularity* (which means that parts of the specification can be easily added, removed, or modified) and *elaboration tolerance* (which means that solutions can be easily reused for similar problems with few or no extra effort). The literature contains several different logic-based languages and formalisms for reasoning about actions under uncertainty, which include especially probabilistic extensions of the situation calculus (Bacchus et al., 1999; Mateus et al., 2001) and Golog (Grosskreutz & Lakemeyer, 2001), of logic programming formalisms (Poole, 1997), and of the action language $\mathcal{A}$ (Baral et al., 2002).

Approaches to classical planning under uncertainty and to decision-theoretic planning, on the other hand, allow especially for defining in a declarative and semantically appealing way courses of actions that achieve a goal with high probability and mappings from situations to actions of high expected utility, respectively. In particular, decision-theoretic planning deals especially with fully observable Markov decision processes (MDPs) (Puterman, 1994) or the more general partially observable Markov decision processes (POMDPs) (Kaelbling, Littman, & Cassandra, 1998).

To combine in a unified formalism the advantages of both ways of designing controllers, a seminal work by Boutilier et al. (2000) presents a generalization of Golog, called DTGolog, where agent programming in Golog relative to stochastic action theories in the situation calculus is combined with decision-theoretic planning in MDPs. The language DTGolog 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. It 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. Furthermore, DTGolog also inherits all the above nice features of logic-based languages and formalisms for reasoning about actions under uncertainty.

A limitation of DTGolog, however, is that it is designed only for the single-agent framework. That is, the model of the world essentially consists of a single agent that we control by a DTGolog program and the environment that is summarized in "nature". But there are many applications where we encounter multiple agents, which may compete against each other, or which may also cooperate with each other. For example, in *robotic soccer*, we have two competing teams of agents, where each team consists of cooperating agents. Here, the optimal actions of one agent generally depend on the actions of all the other ("enemy" and "friend") agents. In particular, there is a bidirected dependence between the actions of two different agents, which generally makes it inappropriate to model enemies and friends of the agent that we control simply as a part of "nature". As an example for an important cooperative domain, in *robotic rescue*, mobile agents may be used in the emergency area to acquire new detailed information (such as the locations of injured people in the emergency area) or to perform certain rescue operations. In general, acquiring information as well as performing rescue operations involves several and different rescue elements (agents and/or teams of agents), which cannot effectively handle the rescue situation on their own. Only the cooperative work among all the rescue elements may solve it. Since most of the rescue tasks involve a certain level of risk for humans (depending on the type of rescue situation), mobile agents can play a major role in rescue situations, especially teams of cooperating heterogeneous mobile agents.

In this paper, we overcome this limitation of DTGolog. We present the multi-agent programming language GTGolog, which combines explicit agent programming in Golog with game-theoretic multi-agent planning in (fully observable) stochastic games (Owen, 1982) (also called Markov games (van der Wal, 1981; Littman, 1994)). GTGolog allows for modeling two competing agents as well as two competing teams of cooperative agents, where any two agents in the same team have the same reward, and any two agents in different teams have zero-sum rewards. It properly generalizes DTGolog to the multi-agent setting, and thus inherits all the nice properties of DTGolog. In particular, it allows for partially specifying a control program in a high-level language as well as for optimally filling in missing details through *game-theoretic planning*. It can thus be seen as a game-theoretic extension to Golog, where choices left to the agent are made by following Nash equilibria. It can also be seen as a formalism that gives advice to a game-theoretic planner, since it naturally constrains the search space. Moreover, GTGolog also inherits from DTGolog all the above nice features of logic-based languages and formalisms for reasoning about actions under uncertainty.

The main idea behind GTGolog can be roughly described as follows for the case of two competing agents. Suppose that we want to control an agent and that, for this purpose, we write or we are already given a DTGolog program that specifies the agent's behavior in a partial way. If the agent acts alone in an environment, then the DTGolog interpreter from (Boutilier et al., 2000) replaces all action choices of our agent in the DTGolog program by some actions that are guaranteed to be optimal. However, if our agent acts in an environment with an enemy agent, then the actions produced by the DTGolog interpreter are in general no longer optimal, since the optimal actions of our agent generally depend on the actions of its enemy, and conversely the actions of the enemy also generally depend on the actions of our agent. Hence, we have to enrich the DTGolog program for our agent by all the possible action moves of its enemy. Every such enriched DTGolog program is a GTGolog program. How do we then define the notion of optimality for the possible actions of our agent? We do this by defining the notion of a Nash equilibrium for GTGolog programs (and thus also for the above DTGolog programs enriched by the actions of the enemy). Every Nash equilibrium consists of a Nash policy for our agent and a Nash policy for its enemy. Since we assume that the rewards of our agent and of its enemy are zero-sum, we then obtain the important result that our agent always behaves optimally when following such a Nash policy, and this even when the enemy follows a Nash policy of another Nash equilibrium or no Nash policy at all. More generally, our agent may also have a library of different DTGolog programs. The GTGolog interpreter then does not only allow for filling them in optimally against an enemy, but it also allows for selecting the DTGolog program of highest expected utility. The following example illustrates the above line of argumentation.

Example 1.1 *(Rugby Domain)* Consider a (robotic) rugby player a, who is carrying the ball and approaching the adversary goal. Suppose that a has no team mate close and is facing only one adversary o on the way towards the goal. At each step, the two players may either (i) remain stationary, or (ii) move left, right, forward, or backward, or (iii) kick or block the ball.

Suppose that we control the player a in such a situation and that we do this by using the following simple DTGolog program, which encodes that a approaches the adversary goal, moves left or right to sidestep the adversary, and then kicks the ball towards the goal:

```
**proc** $attack$
$forward$;
$(right \mid left)$;
$kick$
**end**.
```

How do we now optimally fill in the missing details, that is, how do we determine whether $\boldsymbol{a}$ should better move left or right in the third line? In the case without adversary, the DTGolog interpreter determines an optimal action among the two. In the presence of an adversary, however, the actions filled in by the DTGolog interpreter are in general no longer optimal. In this paper, we propose to use the GTGolog interpreter for filling in optimal actions in DTGolog programs for agents with adversaries: We first enrich the DTGolog program by all the possible actions of the adversary. As a result, we obtain a GTGolog program, which looks as follows for the above DTGolog program:

```
**proc** $attack$
**choice**$(\boldsymbol{a}\colon forward) \parallel$ **choice**$(\boldsymbol{o}\colon stand \mid left \mid right \mid forward \mid backward \mid kick \mid block)$;
**choice**$(\boldsymbol{a}\colon right \mid left) \parallel$ **choice**$(\boldsymbol{o}\colon stand \mid left \mid right \mid forward \mid backward \mid kick \mid block)$;
**choice**$(\boldsymbol{a}\colon kick) \parallel$ **choice**$(\boldsymbol{o}\colon stand \mid left \mid right \mid forward \mid backward \mid kick \mid block)$
**end**.
```

The GTGolog interpreter then specifies a Nash equilibrium for such programs. Each Nash equilibrium consists of a Nash policy for the player $\boldsymbol{a}$ and a Nash policy for its adversary $\boldsymbol{o}$. The former specifies an optimal way of filling in missing actions in the original DTGolog program.

In addition to optimally filling in missing details, the GTGolog interpreter also helps to choose an optimal program from a collection of DTGolog programs for agents with adversaries. For example, suppose that we have the following second DTGolog program:

```
**proc** $attack'$
$(right \mid left)$;
$forward$;
$kick$
**end**.
```

The GTGolog interpreter then determines the Nash equilibria for the enriched GTGolog versions of the two DTGolog programs $attack$ and $attack'$ along with their expected utilities to our agent, and we can finally choose to execute the DTGolog program of maximum utility.

In addition to being a language for programming agents in multi-agent systems, GTGolog can also be considered as a new language for relational specifications of games: The background theory defines the basic structure of a game, and any action choice contained in a GTGolog program defines the points where the agents can make one move each. In this case, rather than looking from the perspective of one agent that we program, we adopt an objective view on all the agents (as usual in game theory). The following example illustrates this use of GTGolog for specifying games.

Example 1.2 *(Rugby Domain cont'd)* Consider a rugby player $\boldsymbol{a}_1$, who wants to cooperate with a team mate $\boldsymbol{a}_2$ towards scoring a goal against another team of two rugby players $\boldsymbol{o}_1$ and $\boldsymbol{o}_2$. Suppose the two rugby players $\boldsymbol{a}_1$ and $\boldsymbol{a}_2$ have to decide their next $n > 0$ steps. Each player may either remain stationary, change its position, pass the ball to its team mate, or receive the ball from its team mate. How should the two players $\boldsymbol{a}_1$ and $\boldsymbol{a}_2$ now best behave against $\boldsymbol{o}_1$ and $\boldsymbol{o}_2$?

The possible moves of the two rugby players $\boldsymbol{a}_1$ and $\boldsymbol{a}_2$ against $\boldsymbol{o}_1$ and $\boldsymbol{o}_2$ in such a part of a game may be encoded by the following procedure in GTGolog, which expresses that while $\boldsymbol{a}_1$ is the ball owner and $n > 0$, all the players simultaneously select one action each:

```
proc $step(n)$
if $(haveBall(\boldsymbol{a}_1) \wedge n > 0)$ then
    $\pi x, x', y, y'($choice$(\boldsymbol{a}_1\colon moveTo(x) \mid passTo(\boldsymbol{a}_2)) \parallel$ choice$(\boldsymbol{a}_2\colon moveTo(x') \mid receive(\boldsymbol{a}_1)) \parallel$
        choice$(\boldsymbol{o}_1\colon moveTo(y) \mid passTo(\boldsymbol{o}_2)) \parallel$ choice$(\boldsymbol{o}_2\colon moveTo(y') \mid receive(\boldsymbol{o}_1)))$;
$step(n{-}1)$
end.
```

Here, the preconditions and effects of the primitive actions are to be formally specified in a suitable domain theory. Given this high-level program and the domain theory, the program interpreter then fills in an optimal way of acting for all the players, reasoning about the possible interactions between the players, where the underlying decision model is a generalization of a stochastic game.

The main contributions of this paper can be summarized as follows:

- We present the multi-agent programming language GTGolog, which integrates explicit agent programming in Golog with game-theoretic multi-agent planning in stochastic games. GTGolog is a proper generalization of both Golog and stochastic games; it also properly generalizes DTGolog to the multi-agent setting. GTGolog allows for modeling two competing agents as well as two competing teams of cooperative agents, where any two agents in the same team have the same reward, and any two agents in different teams have zero-sum rewards. In addition to being a language for programming agents in multi-agent systems, GTGolog can also be considered as a new language for specifying games in game theory.

- We associate with every GTGolog program a set of (finite-horizon) policies, which are possible (finite-horizon) instantiations of the program where missing details are filled in. We then define the notion of a (finite-horizon) Nash equilibrium of a GTGolog program, which is an optimal policy (that is, an optimal instantiation) of the program. We also formally specify a GTGolog interpreter, which computes one of these Nash equilibria. GTGolog thus allows for partially specifying a control program for a single agent or a team of agents, which is then optimally completed by the interpreter against another single agent or another team of agents.

- We prove several important results about the GTGolog interpreter. First, we show that the interpreter is optimal in the sense that it computes a Nash equilibrium. Second, we prove that the single-agent components of two Nash equilibria can be freely mixed to form new Nash equilibria, and thus two competing teams of agents also behave optimally when they follow two different Nash equilibria. Third, we show that GTGolog programs faithfully extend (finite-horizon) stochastic games. That is, they can represent stochastic games, and in the special case where they syntactically model stochastic games, they also semantically behave like stochastic games. Thus, GTGolog programs show a nice semantic behavior here.

- We also show that under suitable assumptions, which include that the horizon is bounded by a constant (which is a quite reasonable assumption in many applications in practice), computing the Nash equilibrium specified by the GTGolog interpreter can be done in polynomial time. Furthermore, we report on a first prototype implementation of a simple GTGolog interpreter (for two competing agents) in constraint logic programming. Finally, we also provide several detailed examples that illustrate our approach and show its practical usefulness.

The rest of this paper is organized as follows. In Section 2, we recall the basic concepts of the situation calculus, Golog, normal form games, and stochastic games. In Section 3, we define the domain theory, syntax, and semantics of GTGolog programs for the case of two competing agents. In Section 4, we formally specify a GTGolog interpreter, we provide optimality, faithfulness, and complexity results for the interpreter, and we describe an implementation of the interpreter. In Section 5, we give an additional extensive example for GTGolog programs. Section 6 then generalizes GTGolog programs to the case of two competing teams of cooperative agents. In Sections 7 and 8, we discuss related work, summarize our results, and give an outlook on future research.

Notice that detailed proofs of all results of this paper as well as excerpts of the implementation of the GTGolog interpreter along with a sample domain are given in Appendices A to C.

2 Preliminaries

In this section, we first recall the main concepts of the situation calculus (in its standard and concurrent version) and of the agent programming language Golog; for further details and background see especially (Reiter, 2001). We then recall the basics of normal form games and stochastic games.

2.1 The Situation Calculus

The situation calculus (McCarthy & Hayes, 1969; Reiter, 2001) is a first-order language for representing dynamically changing worlds. Its main ingredients are *actions*, *situations*, and *fluents*. An *action* is a first-order term of the form $a(u_1, \ldots, u_n)$, where the function symbol a is its *name* and the u_i's are its *arguments*. All changes to the world are the result of actions. For example, the action $moveTo(r, x, y)$ may stand for moving the agent r to the position (x, y). 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 do is a distinguished binary function symbol, 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, the situation $do(moveTo(r, 1, 2), do(moveTo(r, 3, 4), S_0))$ stands for executing $moveTo(r, 1, 2)$ after executing $moveTo(r, 3, 4)$ in the initial situation S_0. We write $Poss(a, s)$, where $Poss$ is a distinguished binary predicate symbol, to denote that the action a is possible to execute in the situation s. A *(relational) 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, $at(r, x, y, s)$ may express that the agent r is at the position (x, y) in the situation s. A situation calculus formula is *uniform* in a situation s iff (i) it does not mention the predicates $Poss$ and $\sqsubset$ (which denotes the proper subsequence relationship on situations), (ii) it does not quantify over situation variables, (iii) it does not mention equality on situations, and (iv) every situation in the situation argument of a fluent coincides with s (cf. (Reiter, 2001)). In the situation calculus, a dynamic domain is represented by a *basic action theory* $AT = (\Sigma, \mathcal{D}_{una}, \mathcal{D}_{S_0}, \mathcal{D}_{ssa}, \mathcal{D}_{ap})$, where:

- Σ is the set of (domain-independent) foundational axioms for situations (Reiter, 2001).
- $\mathcal{D}_{una}$ is the set of unique names axioms for actions, which express that different actions are interpreted in a different way. That is, (i) actions with different names have a different meaning, and (ii) actions

with the same name but different arguments have a different meaning: for all action names a and a', it holds that (i) $a(x_1,\ldots,x_n) \neq a'(y_1,\ldots,y_m)$ if $a \neq a'$, and (ii) $a(x_1,\ldots,x_n) \neq a(y_1,\ldots,y_n)$ if $x_i \neq y_i$ for some $i \in \{1,\ldots,n\}$.

- $\mathcal{D}_{S_0}$ is a set of first-order formulas that are uniform in S_0 describing the initial state of the domain (represented by S_0). For example, the formula $at(r,1,2,S_0) \wedge at(r',3,4,S_0)$ may express that the agents r and r' are initially at the positions $(1,2)$ and $(3,4)$, respectively.

- $\mathcal{D}_{ssa}$ is the set of *successor state axioms* (Reiter, 1991, 2001). 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 that is uniform in s 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, the axiom $at(r,x,y,do(a,s)) \equiv a = moveTo(r,x,y) \vee (at(r,x,y,s) \wedge \neg\exists x',y'\,(a = moveTo(r,x',y')))$ may express that the agent r is at the position (x,y) in the situation $do(a,s)$ iff either r moves to (x,y) in the situation s, or r is already at the position (x,y) and does not move away in s.

- $\mathcal{D}_{ap}$ is the set of *action precondition axioms*. For each action a, it contains an axiom of the form $Poss(a(\vec{x}),s) \equiv \Pi(\vec{x},s)$, where Π is a formula that is uniform in s with free variables among $\vec{x},s$. This axiom characterizes the preconditions of the action a. For example, $Poss(moveTo(r,x,y),s) \equiv \neg\exists r'\, at(r',x,y,s)$ may express that it is possible to move the agent r to the position (x,y) in the situation s iff no agent r' is at (x,y) in s (note that this also includes that the agent r is not at (x,y) in s).

In this paper, we use the concurrent version of the situation calculus (Reiter, 2001), which is an extension of the above standard situation calculus by concurrent actions. A *concurrent action* c is a set of standard actions, which are concurrently executed when c is executed. A situation is then a sequence of concurrent actions of the form $do(c_m,\ldots,do(c_0,S_0))$, where $do(c,s)$ states that all the simple actions a in c are executed at the same time in the situation s.

To encode concurrent actions, some slight modifications to standard basic action theories are necessary. In particular, the successor state axioms in $\mathcal{D}_{ssa}$ are now defined relative to concurrent actions. For example, the above axiom $at(r,x,y,do(a,s)) \equiv a = moveTo(r,x,y) \vee (at(r,x,y,s) \wedge \neg\exists x',y'\,(a = moveTo(r, x',y')))$ in the standard situation calculus is now replaced by the axiom $at(r,x,y,do(c,s)) \equiv moveTo(r, x,y) \in c \vee (at(r,x,y,s) \wedge \neg\exists x',y'(moveTo(r,x',y') \in c))$. Furthermore, the action preconditions in $\mathcal{D}_{ap}$ are extended by further axioms expressing (i) that a singleton concurrent action $c = \{a\}$ is executable if its standard action a is executable, (ii) that if a concurrent action is executable, then it is nonempty and all its standard actions are executable, and (iii) preconditions for concurrent actions. Note that precondition axioms for standard actions are in general not sufficient, since two standard actions may each be executable, but their concurrent execution may not be permitted. This *precondition interaction problem* (Reiter, 2001) (see also (Pinto, 1998) for a discussion) requires some domain-dependent extra precondition axioms.

2.2 Golog

Golog is an agent programming language that is based on the situation calculus. It allows for constructing complex actions (also called *programs*) from (standard or concurrent) primitive actions that are defined in a basic action theory AT, where standard (and not so-standard) Algol-like control constructs can be used. More precisely, *programs* p in Golog have one of the following forms (where c is a (standard or concurrent) primitive action, ϕ is a *condition*, which is obtained from a situation calculus formula that is uniform in s by suppressing the situation argument, $p, p_1, p_2, \ldots, p_n$ are programs, $P_1, \ldots, P_n$ are procedure names, and $x, \vec{x}_1, \ldots, \vec{x}_n$ are arguments):

(1) *Primitive action*: c. Do c.

(2) *Test action*: $\phi?$. Test the truth of ϕ in the current situation.

(3) *Sequence*: $[p_1; p_2]$. Do p_1 followed by p_2.

(4) *Nondeterministic choice of two programs*: $(p_1 \,|\, p_2)$. Do either p_1 or p_2.

(5) *Nondeterministic choice of program argument*: $\pi x\,(p(x))$. Do any $p(x)$.

(6) *Nondeterministic iteration*: $p^\star$. Do p zero or more times.

(7) *Conditional*: **if** ϕ **then** p_1 **else** p_2. If ϕ is true in the current situation, then do p_1 else do p_2.

(8) *While-loop*: **while** ϕ **do** p. While ϕ is true in the current situation, do p.

(9) *Procedures*: **proc** $P_1(\vec{x}_1)$ p_1 **end** ; $\ldots$; **proc** $P_n(\vec{x}_n)$ p_n **end** ; p.

For example, the Golog program **while** $\neg at(r,1,2)$ **do** $\pi x, y\,(moveTo(r,x,y))$ stands for "while the agent r is not at the position $(1,2)$, move r to a nondeterministically chosen position (x,y)".

Golog has a declarative formal semantics, which is defined in the situation calculus. Given a Golog program p, its execution is represented by a situation calculus formula $Do(p,s,s')$, which encodes that the situation s' can be reached by executing the program p in the situation s. The formal semantics of the above constructs in (1)–(9) is then defined as follows:

(1) Primitive action: $Do(c,s,s') \overset{def}{=} Poss(c,s) \wedge s' = do(c,s)$. The situation s' can be reached by executing c in the situation s iff c is executable in s, and s' coincides with $do(c,s)$.

(2) Test action: $Do(\phi?,s,s') \overset{def}{=} \phi[s] \wedge s = s'$. Successfully testing the truth of ϕ in s means that ϕ holds in s and that s' equals to s (testing does not affect the state of the world). Here, $\phi[s]$ is the situation calculus formula obtained from ϕ by restoring s as the suppressed situation argument for all the fluents in ϕ. For example, if $\phi = at(r,1,2)$, then $\phi[s] = at(r,1,2,s)$.

(3) Sequence: $Do([p_1;p_2],s,s') \overset{def}{=} \exists s''(Do(p_1,s,s'') \wedge Do(p_2,s'',s'))$. The situation s' can be reached by executing $[p_1;p_2]$ in the situation s iff there exists a situation s'' such that s'' can be reached by executing p_1 in s, and s' can be reached by executing p_2 in s''.

(4) Nondeterministic choice of two programs: $Do((p_1|p_2),s,s') \overset{def}{=} Do(p_1,s,s') \vee Do(p_2,s,s')$. The situation s' can be reached by executing $(p_1|p_2)$ in the situation s iff s' can be reached either by executing p_1 in s or by executing p_2 in s.

(5) Nondeterministic choice of program argument: $Do(\pi x\,(p(x)),s,s') \overset{def}{=} \exists x\, Do(p(x),s,s')$. The situation s' can be reached by executing $\pi x\,(p(x))$ in the situation s iff there exists an argument x such that s' can be reached by executing $p(x)$ in s.

(6) Nondeterministic iteration: $Do(p^\star,s,s') \overset{def}{=} \forall P\{\forall s_1\, P(s_1,s_1) \wedge \forall s_1,s_2,s_3[P(s_1,s_2) \wedge Do(p,s_2,s_3) \rightarrow P(s_1,s_3)]\} \rightarrow P(s,s')$. The situation s' can be reached by executing $p^\star$ in the situation s iff either (i) s' is equal to s or (ii) there exists a situation s'' such that s'' can be reached by executing $p^\star$ in s, and s' can be reached by executing p in s''. Note that this includes the standard definition of transitive closure, which requires second-order logic.

(7) Conditional: $Do(\textbf{if}\ \phi\ \textbf{then}\ p_1\ \textbf{else}\ p_2, s, s') \overset{def}{=} Do(([\phi?; p_1] \mid [\neg\phi?; p_2]), s, s')$. The conditional is reduced to test action, sequence, and nondeterministic choice of two programs.

(8) While-loop: $Do(\textbf{while}\ \phi\ \textbf{do}\ p, s, s') \overset{def}{=} Do([[\phi?; p]^\star; \neg\phi?], s, s')$. The while-loop is reduced to test action, sequence, and nondeterministic iteration.

(9) Procedures: Do(**proc** $P_1(\vec{x}_1)\ p_1$ **end** ; $\ldots$; **proc** $P_n(\vec{x}_n)\ p_n$ **end** ; $p,\ s,\ s'$) $\overset{def}{=} \forall P_1 \ldots P_n\ [\bigwedge_{i=1}^n \forall s_1, s_2, \vec{x}_i\ (Do(p_i, s_1, s_2) \rightarrow Do(P_i(\vec{x}_i), s_1, s_2))] \rightarrow Do(p, s, s')$, where $Do(P_i(\vec{x}_i),\ s_1, s_2) \overset{def}{=} P_i(\vec{x}_i[s_1], s_1, s_2)$ and $P_i(\vec{x}_i[s_1], s_1, s_2)$ is a predicate representing the P_i procedure call (Reiter, 2001). This is the situation calculus definition (of the semantics of programs involving recursive procedure calls) corresponding to the more usual Scott-Strachey least fixpoint definition in standard programming language semantics (see (Reiter, 2001)).

2.3 Normal Form Games

Normal form games from classical game theory (von Neumann & Morgenstern, 1947) describe the possible actions of $n \geqslant 2$ 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. More formally, a *normal form game* $G = (I, (A_i)_{i \in I}, (R_i)_{i \in I})$ consists of a set of *agents* $I = \{1, \ldots, n\}$ with $n \geqslant 2$, 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 agent i. If $n = 2$, then G is called a *two-player normal form game* (or simply *matrix game*). If additionally $R_1 = -R_2$, then G is a *zero-sum matrix game*; we then often omit R_2 and abbreviate R_1 by R.

The behavior of the agents in a normal form game is expressed through the notions of pure and mixed strategies, which specify which of its actions an agent should execute and which of its actions an agent should execute with which probability, respectively. For example, in two-finger Morra, a pure strategy for player E (or O) is to show two fingers, and a mixed strategy for player E (or O) is to show one finger with the probability $7/12$ and two fingers with the probability $5/12$. 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 given by $R_i(a)$. A *mixed strategy* for agent $i \in I$ is any probability distribution π_i over its set of actions 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 $\sum_{a = (a_i)_{i \in I} \in A} R_i(a) \cdot \Pi_{i \in I} \pi_i(a_i)$.

Towards optimal behavior of the agents in a normal form game, 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, given a normal form game $G = (I, (A_i)_{i \in I}, (R_i)_{i \in I})$, a mixed strategy profile $\pi = (\pi_i)_{i \in I}$ is a *Nash equilibrium* (or also *Nash pair* when $|I| = 2$) of G iff for every agent $i \in I$, it holds that $R_i(\pi \leftarrow \pi'_i) \leqslant R_i(\pi)$ for every mixed strategy π'_i, where $\pi \leftarrow \pi'_i$ is obtained from π by replacing π_i by π'_i. For example, in two-finger Morra, the mixed strategy profile where each player shows one finger with the probability $7/12$ and two fingers with the probability $5/12$ is a Nash equilibrium. Every normal form game G has at least one Nash equilibrium among its mixed (but not necessarily pure) strategy profiles, and many normal form games have multiple Nash equilibria. In the two-player case, they can be computed by linear complementary programming and linear programming in the general and the zero-sum case, respectively. 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 $v^i_f(G)$. In the zero-sum two-player case, also Nash selection functions can be computed by linear programming.

In the zero-sum two-player case, if (π_1, π_2) and (π'_1, π'_2) are two Nash equilibria of G, then $R_1(\pi_1, \pi_2) = R_1(\pi'_1, \pi'_2)$, and also (π_1, π'_2) and (π'_1, π_2) are Nash equilibria of G. That is, the expected reward to the agents is the same under any Nash equilibrium, and Nash equilibria can be freely "mixed" to form new Nash equilibria. The strategies of agent 1 in Nash equilibria are the optimal solutions of the following linear program: $\max v$ subject to (i) $v \leqslant \sum_{a_1 \in A_1} \pi(a_1) \cdot R_1(a_1, a_2)$ for all $a_2 \in A_2$, (ii) $\sum_{a_1 \in A_1} \pi(a_1) = 1$, and (iii) $\pi(a_1) \geqslant 0$ for all $a_1 \in A_1$. Moreover, the expected reward to agent 1 under a Nash equilibrium is the optimal value of the above linear program.

2.4 Stochastic Games

Stochastic games (Owen, 1982), or also called Markov games (van der Wal, 1981; Littman, 1994), generalize both normal form games and Markov decision processes (MDPs) (Puterman, 1994).

A stochastic game consists of a set of states S, a normal form game for every state $s \in S$ (with common sets of agents and sets of actions for each agent), and a transition function that associates with every state $s \in S$ and joint action of the agents a probability distribution on future states $s' \in S$. Formally, a *stochastic game* $G = (I, S, (A_i)_{i \in I}, P, (R_i)_{i \in I})$ consists of a set of agents $I = \{1, \ldots, n\}$, $n \geqslant 2$, a nonempty finite set of states S, a nonempty finite set of actions A_i for each agent $i \in I$, a transition function P that associates with every state $s \in S$ and joint action $a \in A = \times_{i \in I} A_i$ a probability distribution $P(\cdot \mid s, a)$ over the set of states S, 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* $R(s, a)$ to agent i. If $n = 2$, then G is a *two-player stochastic game*. If also $R_1 = -R_2$, then G is a *zero-sum two-player stochastic game*; we then often omit R_2 and abbreviate R_1 by R.

Assuming a finite horizon $H \geqslant 0$, a pure (resp., mixed) time-dependent policy associates with every state $s \in S$ and number of steps to go $h \in \{0, \ldots, H\}$ a pure (resp., mixed) strategy of a normal form game. Formally, a *pure policy* α_i for agent $i \in I$ assigns to each state $s \in S$ and number of steps to go $h \in \{0, \ldots, H\}$ an action from A_i. A *pure policy profile* $\alpha = (\alpha_i)_{i \in I}$ consists of a pure policy α_i for each agent $i \in I$. The *H-step reward* to agent $i \in I$ under a start state $s \in S$ and the pure policy profile $\alpha = (\alpha_i)_{i \in I}$, denoted $G_i(H, s, \alpha)$, is defined as $R_i(s, \alpha(s, 0))$, if $H = 0$, and $R_i(s, \alpha(s, H)) + \sum_{s' \in S} P(s' | s, \alpha(s, H)) \cdot G_i(H{-}1, s', \alpha)$, otherwise. A *mixed policy* π_i for agent $i \in I$ assigns to every state $s \in S$ and number of steps to go $h \in \{0, \ldots, H\}$ a probability distribution over the set of actions A_i. A *mixed policy profile* $\pi = (\pi_i)_{i \in I}$ consists of a mixed policy π_i for each agent $i \in I$. The *expected H-step reward* to agent i under a start state s and the mixed policy profile $\pi = (\pi_i)_{i \in I}$, denoted $G_i(H, s, \pi)$, is defined as $\mathbf{E}[R_i(s, a) \mid \pi(s, 0)]$, if $H = 0$, and $\mathbf{E}[R_i(s, a) + \sum_{s' \in S} P(s' | s, a) \cdot G_i(H{-}1, s', \pi) \mid \pi(s, H)]$, otherwise.

The notion of a finite-horizon Nash equilibrium for stochastic games is then defined as follows. A mixed policy profile $\pi = (\pi_i)_{i \in I}$ is a *(H-step) Nash equilibrium* (or also *(H-step) Nash pair* when $|I| = 2$) of G iff for every agent $i \in I$ and every start state $s \in S$, it holds that $G_i(H, s, \pi \leftarrow \pi'_i) \leqslant G_i(H, s, \pi)$ for every mixed policy π'_i, where $\pi \leftarrow \pi'_i$ is obtained from π by replacing π_i by π'_i. Every stochastic game G has at least one Nash equilibrium among its mixed (but not necessarily pure) policy profiles, and it may have exponentially many Nash equilibria.

Nash equilibria for G can be computed by finite-horizon value iteration from local Nash equilibria of normal form games as follows (Kearns et al., 2000). We assume an arbitrary Nash selection function f for normal form games (with the set of agents $I = \{1, \ldots, n\}$ and the sets of actions $(A_i)_{i \in I}$). For every state $s \in S$ and every number of steps to go $h \in \{0, \ldots, H\}$, the normal form game $G[s, h] = (I, (A_i)_{i \in I}, (Q_i[s, h])_{i \in I})$

is defined by $Q_i[s,0](a) = R_i(s,a)$ and $Q_i[s,h](a) = R_i(s,a) + \sum_{s' \in S} P(s'|s,a) \cdot v^i_f(G[s', h{-}1])$ for every joint action $a \in A = \times_{i \in I} A_i$ and every agent $i \in I$. For every agent $i \in I$, let the mixed policy π_i be defined by $\pi_i(s,h) = f_i(G[s,h])$ for every $s \in S$ and $h \in \{0, \ldots, H\}$. Then, $\pi = (\pi_i)_{i \in I}$ is a H-step Nash equilibrium of G, and it holds $G_i(H, s, \pi) = v^i_f(G[s,H])$ for every agent $i \in I$ and every state $s \in S$.

In the case of zero-sum two-player stochastic games G, by induction on $h \in \{0, \ldots, H\}$, it is easy to see that, for every $s \in S$ and $h \in \{0, \ldots, H\}$, the normal form game $G[s,h]$ is also zero-sum. Moreover, all Nash equilibria that are computed by the above finite-horizon value iteration produce the same expected H-step reward, and they can be freely "mixed" to form new Nash equilibria.

3 Game-Theoretic Golog (GTGolog)

In this section, we present the agent programming language GTGolog for the case of two competing agents (note that its generalization to two competing teams of agents is given in Section 6). We first introduce the domain theory and then the syntax and semantics of GTGolog programs.

3.1 Domain Theory

GTGolog programs are interpreted relative to a domain theory, which is an extension of a basic action theory by stochastic actions, reward functions, and utility functions. Formally, in addition to a basic action theory AT, a *domain theory* $DT = (AT, ST, OT)$ consists of a *stochastic theory* ST and an *optimization theory* OT, which are both defined below. We assume two (zero-sum) competing agents $\boldsymbol{a}$ and $\boldsymbol{o}$, also called the *agent* and the *opponent*, respectively. In the agent programming use of GTGolog, $\boldsymbol{a}$ is under our control, while $\boldsymbol{o}$ is not, whereas in the game specifying use of GTGolog, we adopt an objective view on both agents. The set of primitive actions is partitioned into the sets of primitive actions A and O of agents $\boldsymbol{a}$ and $\boldsymbol{o}$, respectively. A *single-agent action* of agent $\boldsymbol{a}$ (resp., $\boldsymbol{o}$) is any concurrent action over A (resp., O). A *two-agent action* is any concurrent action over $A \cup O$. For example, the concurrent actions $\{\mathit{moveTo}(\boldsymbol{a},1,2)\} \subseteq A$ and $\{\mathit{moveTo}(\boldsymbol{o},2,3)\} \subseteq O$ are single-agent actions of $\boldsymbol{a}$ and $\boldsymbol{o}$, respectively, and thus also two-agent actions, while the concurrent action $\{\mathit{moveTo}(\boldsymbol{a},1,2), \mathit{moveTo}(\boldsymbol{o},2,3)\}$ is only a two-agent action.

A *stochastic theory* ST is a set of axioms that define stochastic actions. As usual (Boutilier et al., 2000; Finzi & Pirri, 2001), we represent stochastic actions through a finite set of deterministic actions. When a stochastic action is executed, then "nature" chooses and executes with a certain probability exactly one of its deterministic actions. We use the predicate $\mathit{stochastic}(c,s,n,p)$ to encode that when executing the stochastic action c in the situation s, nature chooses the deterministic action n with the probability p. We then call n a *deterministic component* of c in s. Here, for every stochastic action c and situation s, the set of all (n,p) such that $\mathit{stochastic}(c,s,n,p)$ is a probability function on the set of all deterministic components n of c in s, denoted $\mathit{prob}(c,s,n)$. We assume that c and all its nature choices n have the same preconditions. A stochastic action c is then indirectly represented by providing a *successor state axiom* for each associated nature choice n. Thus, basic action theories AT are extended to a probabilistic setting in a minimal way. For example, consider the stochastic action $\mathit{moveS}(k,x,y)$ of the agent $k \in \{\boldsymbol{a}, \boldsymbol{o}\}$ moving to the position (x,y), which has the effect that k moves to either (x,y) or $(x,y{+}1)$. The following formula associates with $\mathit{moveS}(k,x,y)$ its deterministic components and their probabilities 0.9 and 0.1, respectively:

$$\begin{array}{l}\mathit{stochastic}(\{\mathit{moveS}(k,x,y)\}, s, \{\mathit{moveTo}(k,x,t)\}, p) \overset{def}{=} \\ \qquad k \in \{\boldsymbol{a}, \boldsymbol{o}\} \wedge ((t = y \wedge p = 0.9) \vee (t = y{+}1 \wedge p = 0.1))\,.\end{array}$$

The stochastic action $\mathit{moveS}(k,x,y)$ is then fully specified by the precondition and successor state axioms of $\mathit{moveTo}(k,x,y)$ in Section 2.1. The possible deterministic effects of the concurrent execution of $\mathit{moveS}(\boldsymbol{a},x,y)$ and $\mathit{moveS}(\boldsymbol{o},x,y)$ along with their probabilities may be encoded by:

$$\begin{aligned}&\mathit{stochastic}(\{\mathit{moveS}(\boldsymbol{a},x,y),\mathit{moveS}(\boldsymbol{o},x,y)\},s,\{\mathit{moveTo}(\boldsymbol{a},x,t),\mathit{moveTo}(\boldsymbol{o},x,t')\},p)\overset{def}{=}\\&\qquad(t=y\wedge t'=y{+}1\wedge p=0.5)\vee(t=y{+}1\wedge t'=y\wedge p=0.5)\,.\end{aligned}$$

We assume that the domain is *fully observable*. To this end, we introduce *observability axioms*, which disambiguate the state of the world after executing a stochastic action. For example, after executing $\mathit{moveS}(a,x,y)$, we test the predicates $at(a,x,y,s)$ and $at(a,x,y+1,s)$ to check which of the two possible deterministic components (that is, either $\mathit{moveTo}(a,x,y)$ or $\mathit{moveTo}(a,x,y{+}1)$) was actually executed. This condition is represented by the predicate $\mathit{condStAct}(c,s,n)$, where c is a stochastic action, s is a situation, n is a deterministic component of c, and $\mathit{condStAct}(c,s,n)$ is true iff executing c in s has resulted in actually executing n. For example, the predicate $\mathit{condStAct}(c,\,s,\,n)$ for the stochastic action $\mathit{moveS}(k,x,y)$ is defined as follows:

$$\begin{aligned}&\mathit{condStAct}(\{\mathit{moveS}(k,x,y)\},s,\{\mathit{moveTo}(k,x,y)\})\overset{def}{=}at(k,x,y,s)\,,\\&\mathit{condStAct}(\{\mathit{moveS}(k,x,y)\},s,\{\mathit{moveTo}(k,x,y{+}1)\})\overset{def}{=}at(k,x,y{+}1,s)\,.\end{aligned}$$

An *optimization theory* OT specifies a reward function, a utility function, and Nash selection functions. The reward function associates with every two-agent action α and situation s, a reward to agent $\boldsymbol{a}$, denoted $\mathit{reward}(\alpha,s)$. Since we assume two zero-sum competing agents $\boldsymbol{a}$ and $\boldsymbol{o}$, the reward to agent $\boldsymbol{o}$ is at the same time given by $-\mathit{reward}(\alpha,s)$. For example, $\mathit{reward}(\{\mathit{moveTo}(\boldsymbol{a},x,\,y)\},s)=y$ may encode that the reward to agent $\boldsymbol{a}$ when moving to the position (x,y) in the situation s is given by y. Note that the reward function for stochastic actions is defined through a reward function for their deterministic components. The utility function $\mathit{utility}$ maps every pair consisting of a reward v and a probability value pr (that is, a real from the unit interval $[0,1]$) to a real-valued utility $\mathit{utility}(v,pr)$. We assume that $\mathit{utility}(v,1)=v$ and $\mathit{utility}(v,0)=0$ for all rewards v. An example of a utility function is $\mathit{utility}(v,pr)=v\cdot pr$. Informally, differently from actions in decision-theoretic planning, actions in Golog may fail due to unsatisfied preconditions. Hence, the usefulness of an action/program does not only depend on its reward, but also on the probability that it is executable. The utility function then combines the reward of an action/program with the probability that it is executable. In particular, $\mathit{utility}(v,pr)=v\cdot pr$ weights the reward of an action/program with the probability that it is executable. Finally, we assume Nash selection functions $\mathit{selectNash}$ for zero-sum matrix games of the form $(I,(A_i)_{i\in I},R)$, where $I=\{\boldsymbol{a},\boldsymbol{o}\}$ and the sets of actions $A_{\boldsymbol{a}}$ and $A_{\boldsymbol{o}}$ are nonempty sets of single-agent actions of agents $\boldsymbol{a}$ and $\boldsymbol{o}$, respectively. Similarly to all arithmetic operations, utility functions and Nash selection functions are assumed to be pre-interpreted (rigid), and thus they are not explicitly axiomatized in the domain theory.

Example 3.1 *(Rugby Domain cont'd)* Consider the following rugby domain, which is a slightly modified version of the soccer domain by Littman (1994). The rugby field (see Fig. 1) is a 4 × 7 grid of 28 squares, and it includes two designated areas representing two goals. There are two players, denoted $\boldsymbol{a}$ and $\boldsymbol{o}$, each occupying a square, and each able to do one of the following moves on each turn: N, S, E, W and stand (move up, move down, move left, move right, and no move, respectively). The ball is represented by a circle and also occupies a square. A player 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. When the ball owner goes into the square occupied by the other player, if the other player stands, then

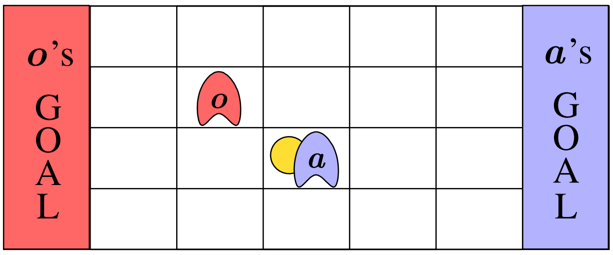


Figure 1: Rugby Domain.

the possession of the ball changes. Therefore, a good defensive maneuver is to stand where the other agent wants to go.

We define the domain theory $DT = (AT, ST, OT)$ as follows. As for the basic action theory AT, we introduce the deterministic action $move(\alpha, m)$ (encoding that agent α performs m among N, S, E, W and $stand$), and the fluents $at(\alpha, x, y, s)$ (encoding that agent α is at position (x, y) in situation s) and $haveBall(\alpha, s)$ (encoding that agent α is the ball owner in situation s), which are defined by the following successor state axioms:

$$\begin{aligned} &at(\alpha, x, y, do(c, s)) \equiv at(\alpha, x, y, s) \wedge \neg\exists m\,(move(\alpha, m) \in c) \vee \\ &\quad \exists x', y', m\,(at(\alpha, x', y', s) \wedge move(\alpha, m) \in c \wedge \phi(x, y, x', y', m))\,; \end{aligned}$$

$$haveBall(\alpha, do(c, s)) \equiv haveBall(\alpha, s) \wedge \neg\exists\beta\,(cngBall(\beta, c, s)) \vee cngBall(\alpha, c, s)\,.$$

Here, $\phi(x, y, x', y', m)$ is true iff the coordinates change from (x', y') to (x, y) due to m, that is,

$$\begin{aligned} &\phi(x, y, x', y', m) \overset{def}{=} (m \notin \{N, S, E, W\} \wedge x = x' \wedge y = y') \vee \\ &\quad (m = N \wedge x = x' \wedge y = y'{+}1) \vee (m = S \wedge x = x' \wedge y = y'{-}1) \vee \\ &\quad (m = E \wedge x = x'{+}1 \wedge y = y') \vee (m = W \wedge x = x'{-}1 \wedge y = y')\,, \end{aligned}$$

and $cngBall(\alpha, c, s)$ is true iff the ball possession changes to agent α after the action c in s, that is,

$$\begin{aligned} &cngBall(\alpha, c, s) \overset{def}{=} \exists x, y, \beta, x', y', m\,(at(\alpha, x, y, s) \wedge move(\alpha, stand) \in c \wedge \beta \neq \alpha \wedge \\ &\quad haveBall(\beta, s) \wedge at(\beta, x', y', s)\ \wedge move(\beta, m) \in c \wedge \phi(x, y, x', y', m))\,. \end{aligned}$$

The precondition axioms encode that the agents cannot go out of the rugby field:

$$\begin{aligned} &Poss(move(\alpha, m), s) \equiv \neg\exists x, y\,(at(\alpha, x, y, s) \wedge ((x = 0 \wedge m = W) \vee \\ &\quad (x = 6 \wedge m = E) \vee (y = 1 \wedge m = S) \vee (y = 4 \wedge m = N))). \end{aligned}$$

Moreover, every possible two-agent action consists of at most one standard action per agent, that is,

$$\begin{aligned} &Poss(\{move(\alpha, m_1), move(\beta, m_2)\}, s) \ \equiv \\ &\quad Poss(move(\alpha, m_1), s) \wedge Poss(move(\beta, m_2), s) \wedge \alpha \neq \beta\,. \end{aligned}$$

To keep this example technically as simple as possible, we use no stochastic actions here, and thus the stochastic theory ST is empty. As for the optimization theory OT, we use the product as the utility function $utility$ and any suitable Nash selection function $selectNash$ for matrix games. Furthermore, we define the reward function $reward$ for agent $\boldsymbol{a}$ as follows:

$$\begin{aligned} &reward(c, s) = r \overset{def}{=} \exists\alpha\,(goal(\alpha, do(c, s)) \wedge (\alpha = \boldsymbol{a} \wedge r = 1000 \vee \alpha = \boldsymbol{o} \wedge r = -\,1000)) \vee \\ &\quad \neg\exists\alpha\,(goal(\alpha, do(c, s))) \wedge evalPos(c, r, s)\,. \end{aligned}$$

Intuitively, the reward to agent $\boldsymbol{a}$ is 1000 (resp., -1000), if $\boldsymbol{a}$ (resp., $\boldsymbol{o}$) scores a goal, and the reward to agent $\boldsymbol{a}$ depends on the position of the ball-owner after executing c in s, otherwise. Here, the predicates $goal(\alpha, s)$ and $evalPos(c, r, s)$ are defined as follows:

$$goal(\alpha, s) \overset{def}{=} \exists x, y\, (haveBall(\alpha, s) \wedge at(\alpha, x, y, s) \wedge goalPos(\alpha, x, y))$$

$$evalPos(c, r, s) \overset{def}{=} \exists \alpha, x, y\, (haveBall(\alpha, do(c, s)) \wedge at(\alpha, x, y, do(c, s)) \wedge \\ \quad (\alpha = \boldsymbol{a} \wedge r = 6 - x \vee \alpha = \boldsymbol{o} \wedge r = -x))\,,$$

where $goalPos(\alpha, x, y)$ is true iff (x, y) are the goal coordinates of the adversary of α, and the predicate $evalPos(c, r, s)$ describes the reward r to agent $\boldsymbol{a}$ depending on the ball-owner and the position of the ball-owner after executing c in s. Informally, the reward to agent $\boldsymbol{a}$ is high (resp., low) if $\boldsymbol{a}$ is the ball-owner and close to (resp., far from) the adversary goal, and the reward to agent $\boldsymbol{a}$ is high (resp., low) if $\boldsymbol{o}$ is the ball-owner and far from (resp., close to) the adversary goal.

3.2 Syntax of GTGolog

In the sequel, let DT be a domain theory. We define GTGolog by induction as follows. A *program* p in GTGolog has one of the following forms (where α is a two-agent action or the empty action nop (which is always executable and does not change the state of the world), ϕ is a condition, $p, p_1, p_2, \ldots, p_n$ are programs without procedure declarations, $P_1, \ldots, P_n$ are procedure names, $x, \vec{x}_1, \ldots, \vec{x}_n$ are arguments, and $a_1, \ldots, a_n$ and $o_1, \ldots, o_m$ are single-agent actions of agents $\boldsymbol{a}$ and $\boldsymbol{o}$, respectively, and $\tau = \{\tau_1, \tau_2, \ldots, \tau_n\}$ is a finite nonempty set of ground terms):

(1) *Deterministic or stochastic two-agent action*: α.

(2) *Nondeterministic action choice of agent* $\boldsymbol{a}$: **choice**$(\boldsymbol{a}\colon a_1 | \cdots | a_n)$.

(3) *Nondeterministic action choice of agent* $\boldsymbol{o}$: **choice**$(\boldsymbol{o}\colon o_1 | \cdots | o_m)$.

(4) *Nondeterministic joint action choice*: **choice**$(\boldsymbol{a}\colon a_1 | \cdots \mid a_n) \parallel$ **choice**$(\boldsymbol{o}\colon o_1 | \cdots | o_m)$.

(5) *Test action*: $\phi?$.

(6) *Sequence*: $[p_1; p_2]$.

(7) *Nondeterministic choice of two programs*: $(p_1 \,|\, p_2)$.

(8) *Nondeterministic choice of program argument*: $\pi[x : \tau](p(x))$.

(9) *Nondeterministic iteration*: $p^{\star}$.

(10) *Conditional*: **if** ϕ **then** p_1 **else** p_2.

(11) *While-loop*: **while** ϕ **do** p.

(12) *Procedures*: **proc** $P_1(\vec{x}_1)\; p_1$ **end** ; . . .; **proc** $P_n(\vec{x}_n)\; p_n$ **end** ; p.

Hence, compared to Golog, we now also have two-agent actions (instead of only primitive or concurrent actions) and stochastic actions (instead of only deterministic actions). Furthermore, we now additionally have three different kinds of nondeterministic action choices for the two agents in (2)–(4), where one ore both of the two agents can choose among a finite set of single-agent actions. Informally, (2) (resp., (3)) stands for “do an optimal action for agent $\boldsymbol{a}$ (resp., $\boldsymbol{o}$) among $a_1, \ldots, a_n$ (resp., $o_1, \ldots, o_m$)”, while (4) stands for “do any action $a_i \cup o_j$, where $i \in \{1, \ldots, n\}$ and $j \in \{1, \ldots, m\}$, with an optimal probability $\pi_{i,j}$”. The formal semantics of (2)–(4) is defined in such a way that an optimal action is chosen for each of the two agents (see Section 4.1). As usual, the sequence operator ";" is associative (for example, $[[p_1; p_2]; p_3]$ and $[p_1; [p_2; p_3]]$ have the same semantics), and we often use "$p_1; p_2$", "**if** ϕ **then** p_1", and "$\pi x\,(p(x))$" to abbreviate "$[p_1; p_2]$", "**if** ϕ **then** p_1 **else** nop", and $\pi[x : \tau](p(x))$, respectively, when there is no danger of confusion.

Example 3.2 *(Rugby Domain cont'd)* A complete rugby session can be encoded through the following GTGolog procedure relative to the domain theory DT of Example 3.1:

```
proc $game()$
while $\neg goal(\boldsymbol{a}) \wedge \neg goal(\boldsymbol{o})$ do
    choice($\boldsymbol{a}$: $move(\boldsymbol{a}, N) \mid move(\boldsymbol{a}, S) \mid move(\boldsymbol{a}, E) \mid move(\boldsymbol{a}, W) \mid move(\boldsymbol{a}, stand)$) $\|$
        choice($\boldsymbol{o}$: $move(\boldsymbol{o}, N) \mid move(\boldsymbol{o}, S) \mid move(\boldsymbol{o}, E) \mid move(\boldsymbol{o}, W) \mid move(\boldsymbol{o}, stand)$)
end.
```

Informally, while no goal is reached, the agents $\boldsymbol{a}$ and $\boldsymbol{o}$ simultaneously perform one move each.

The above GTGolog procedure $game$ represents a generic rugby session. In addition to this, some more specialized rugby playing behavior can also be formulated in GTGolog. For example, agent $\boldsymbol{a}$ could discriminate different situations Φ_i, $i \in \{1, \ldots, l\}$, where the rugby session can be simplified (that is, the possible moves of the two agents $\boldsymbol{a}$ and $\boldsymbol{o}$ can be restricted):

```
proc $game'()$
while $\neg goal(\boldsymbol{a}) \wedge \neg goal(\boldsymbol{o})$ do
    if $\Phi_1$ then $schema_1$
        else if $\Phi_2$ then $schema_2$
            else $game$
end.
```

For example, consider an attacking ball owner $\boldsymbol{a}$, which is closer to the adversary's goal than the adversary (that is, $\Phi_1(s) = \exists x, y, x', y'\ (at(\boldsymbol{a}, x, y, s) \wedge at(\boldsymbol{o}, x', y', s) \wedge x' > x)$). In such a situation, since the adversary $\boldsymbol{o}$ is behind, a good way of acting of agent $\boldsymbol{a}$ is to move quickly towards $\boldsymbol{o}$'s goal. This can be encoded as a GTGolog procedure $schema_1$:

```
proc $schema_1()$
if $\neg goal(\boldsymbol{a})$ then $move(\boldsymbol{a}, W)$
end.
```

As another example, consider a situation s in which $\Phi_2(s) = haveBall(\boldsymbol{a}, s) \wedge \exists x, x', y\ (at(\boldsymbol{a}, x, y, s) \wedge at(\boldsymbol{o}, x', y, s) \wedge x' = x - 1)$ is true, that is, agent $\boldsymbol{a}$ has the ball and is facing the opponent $\boldsymbol{o}$ who is closer to its goal. In this case, a good way of acting of agent $\boldsymbol{a}$ is to try a dribbling maneuver in k steps. This can be encoded by the GTGolog procedure **proc** $schema_2\ \pi k\ (dribbling(k))$ **end**, where $dribbling(k)$ is given as follows:

```
proc $dribbling(k)$
if $k > 0$ then [
    choice($\boldsymbol{a}$: $move(\boldsymbol{a}, S) \mid move(\boldsymbol{a}, W)$) $\|$
        choice($\boldsymbol{o}$: $move(\boldsymbol{o}, S) \mid move(\boldsymbol{o}, stand)$);
    $dribbling(k-1)$ ]
end.
```

Hence, $game'$ specializes $game$ during the run of $schema_2$ by restricting the meaningful possible moves for both the agent $\boldsymbol{a}$ and its adversary $\boldsymbol{o}$ during the dribbling phase.

3.3 Policies and Nash Equilibria of GTGolog

We now define the formal semantics of GTGolog programs p relative to a domain theory DT in terms of a set of Nash equilibria of p, which are optimal finite-horizon policies of p. We first associate with every GTGolog program p, situation s, and horizon $H \geqslant 0$, a set of H-step policies π along with their expected H-step utilities $U_{\boldsymbol{a}}$ and $U_{\boldsymbol{o}}$ to agents $\boldsymbol{a}$ and $\boldsymbol{o}$, respectively. We then define the notion of an H-step Nash equilibrium to characterize a subset of optimal such policies, which is the natural semantics of a GTGolog program relative to a domain theory.

Intuitively, given a horizon $H \geqslant 0$, an H-step policy π of a GTGolog program p in a situation s relative to a domain theory DT is obtained from the H-horizon part of p by replacing every single-agent choice by a single action, and every multi-agent choice by a collection of probability distributions, one over the actions of each agent. Every such H-step policy π is associated with an expected H-step reward to $\boldsymbol{a}$ (resp., $\boldsymbol{o}$), an H-step success probability (which is the probability that π is executable in s), and an expected H-step utility to $\boldsymbol{a}$ (resp., $\boldsymbol{o}$) (which is computed from the expected H-step reward and the H-step success probability using the utility function).

Formally, the *nil-terminated variant* of a GTGolog program p, denoted $\widehat{p}$, is inductively defined by $\widehat{p} = [p_1; \widehat{p}_2]$, if $p = [p_1; p_2]$, and $\widehat{p} = [p; \mathit{nil}]$, otherwise. Given a GTGolog program p relative to a domain theory DT, a horizon $H \geqslant 0$, and a start situation s, we say that π is an *H-step policy* of p in s relative to DT with *expected H-step reward* v (resp., $-v$), *H-step success probability* pr, and *expected H-step utility* $U_{\boldsymbol{a}}(H, s, \pi) = \mathit{utility}(v, pr)$ (resp., $U_{\boldsymbol{o}}(H, s, \pi) = -\mathit{utility}(v, pr)$) to agent $\boldsymbol{a}$ (resp., $\boldsymbol{o}$) iff $DT \models G(\widehat{p}, s, H, \pi, v, pr)$, where the macro $G(\widehat{p}, s, h, \pi, v, pr)$, for every number of steps to go $h \in \{0, \ldots, H\}$, is defined by induction on the structure of $\widehat{p}$ as follows (intuitively, $\widehat{p}$, s, and h are the input values of G, while π, v, and pr are the output values of G):

- *Null program* or *zero horizon*:
  If $\widehat{p} = \mathit{nil}$ or $h = 0$, then:

  $$G(\widehat{p}, s, h, \pi, v, pr) \stackrel{def}{=} \pi = \mathit{nil} \wedge v = 0 \wedge pr = 1 \,.$$

  Informally, p has only the policy $\pi = \mathit{nil}$ along with the expected reward $v = 0$ and the success probability $pr = 1$.

- *Deterministic first program action*:
  If $\widehat{p} = [c\,; p']$, where c is a deterministic action, and $h > 0$, then:

  $$\begin{array}{l} G([c\,; p'], s, h, \pi, v, pr) \stackrel{def}{=} \\ \quad (\neg \mathit{Poss}(c, s) \wedge \pi = \mathit{stop} \wedge v = 0 \wedge pr = 0) \vee \\ \quad (\mathit{Poss}(c, s) \wedge \exists \pi', v', pr'\, (G(p', do(c, s), h{-}1, \pi', v', pr') \wedge \\ \qquad \pi = c\,; \pi' \wedge v = v' {+} \mathit{reward}(c, s) \wedge pr = pr')) \,. \end{array}$$

  Informally, if c is not executable in s, then p has only the policy $\pi = \mathit{stop}$ along with the expected reward $v = 0$ and the success probability $pr = 0$. Here, stop is a zero-cost action, which takes the agents to an absorbing state, where they stop the execution of the policy and wait for further instructions. Otherwise, every policy of p is of the form $\pi = c\,; \pi'$ with the expected reward $v = v' {+} \mathit{reward}(c, s)$ and the success probability $pr = pr'$, where π' is a policy for the execution of p' in $do(c, s)$ with the expected reward v' and the success probability pr'.

- *Stochastic first program action (choice of nature)*:
  If $\widehat{p} = [c\,; p']$, where c is a stochastic action, and $h > 0$, then:

$$
\begin{aligned}
&G([c\,; p'], s, h, \pi, v, pr) \stackrel{def}{=} \\
&\quad \exists l, n_1, \ldots, n_l, \pi_1, \ldots, \pi_l, v_1, \ldots, v_l, pr_1, \ldots, pr_l \; (\textstyle\bigwedge_{i=1}^{l} G([n_i\,; p'], \\
&\qquad s, h, n_i\,; \pi_i, v_i, pr_i) \wedge \{n_1, \ldots, n_l\} = \{n \mid \exists p\, (\mathit{stochastic}(c, s, n, p))\} \wedge \\
&\qquad \pi = c\,; \textbf{if } \mathit{condStAct}(c, s, n_1) \textbf{ then } \pi_1 \textbf{ else if } \mathit{condStAct}(c, s, n_1) \textbf{ then } \pi_2 \\
&\qquad\quad \ldots \textbf{ else if } \mathit{condStAct}(c, s, n_l) \textbf{ then } \pi_l \wedge \\
&\qquad v = \textstyle\sum_{i=1}^{l} v_i \cdot \mathit{prob}(c, s, n_i) \wedge pr = \textstyle\sum_{i=1}^{l} pr_i \cdot \mathit{prob}(c, s, n_i))\,.
\end{aligned}
$$

  Informally, every policy of p consists of c and a conditional plan expressed as a cascade of if-then-else statements, considering each possible choice of nature, associated with the expected reward and the expected success probability. The n_i's are the choices of nature of c in s, and the $\mathit{condStAct}(c, s, n_1)$'s are their conditions from the observability axioms. Note that the agents perform an implicit sensing operation when evaluating these conditions.

- *Nondeterministic first program action (choice of agent $\boldsymbol{a}$)*:
  If $\widehat{p} = [\textbf{choice}(\boldsymbol{a}\colon\, a_1 | \cdots | a_m)\,; p']$ and $h > 0$, then:

$$
\begin{aligned}
&G([\textbf{choice}(\boldsymbol{a}\colon\, a_1 | \cdots | a_m)\,; p'], s, h, \pi, v, pr) \stackrel{def}{=} \\
&\quad \exists \pi_1, \ldots, \pi_m, v_1, \ldots, v_m, pr_1, \ldots, pr_m, k \; (\textstyle\bigwedge_{i=1}^{m} G([a_i\,; p'], s, h, a_i\,; \pi_i, v_i, pr_i) \wedge \\
&\qquad k \in \{1, \ldots, m\} \wedge \pi = a_k\,; \textbf{if } \mathit{condNonAct}(a_1 | \cdots | a_m, a_1) \textbf{ then } \pi_1 \\
&\qquad\quad \textbf{else if } \mathit{condNonAct}(a_1 | \cdots | a_m, a_2) \textbf{ then } \pi_2 \\
&\qquad\quad \ldots \textbf{ else if } \mathit{condNonAct}(a_1 | \cdots | a_m, a_m) \textbf{ then } \pi_m \wedge \\
&\qquad v = v_k \wedge pr = pr_k)\,.
\end{aligned}
$$

  Informally, every policy π of p consists of any action a_k and one policy π_i of p' for every possible action a_i. The expected reward and the success probability of π are given by the expected reward v_k and the success probability pr_k of π_k. For agent $\boldsymbol{o}$ to observe which action among $a_1, \ldots, a_m$ was actually executed by agent $\boldsymbol{a}$, we use a cascade of if-then-else statements with conditions of the form $\mathit{condNonAct}(a_1 | \cdots | a_m, a_i)$ (being true when a_i was actually executed), which are tacitly assumed to be defined in the domain theory DT. Note that the conditions $\mathit{condNonAct}(a_1 | \cdots | a_m, a_i)$ here are to observe which action a_i was actually executed by agent $\boldsymbol{a}$, while the conditions $\mathit{condStAct}(c, s, n_i)$ above are to observe which action n_i was actually executed by nature after a stochastic action c in s. In the sequel, we also use $\mathit{condNonAct}(a_i)$ to abbreviate $\mathit{condNonAct}(a_1 | \cdots | a_m, a_i)$.

- *Nondeterministic first program action (choice of agent $\boldsymbol{o}$)*:
  If $\widehat{p} = [\textbf{choice}(\boldsymbol{o}\colon\, o_1 | \cdots | o_n)\,; p']$ and $h > 0$, then:

$$
\begin{aligned}
&G([\textbf{choice}(\boldsymbol{o}\colon\, o_1 | \cdots | o_n)\,; p'], s, h, \pi, v, pr) \stackrel{def}{=} \\
&\quad \exists \pi_1, \ldots, \pi_n, v_1, \ldots, v_n, pr_1, \ldots, pr_n, k \; (\textstyle\bigwedge_{j=1}^{n} G([o_j\,; p'], s, h, o_j\,; \pi_j, v_j, pr_j) \wedge \\
&\qquad k \in \{1, \ldots, n\} \wedge \pi = o_k\,; \textbf{if } \mathit{condNonAct}(o_1 | \cdots | o_n, o_1) \textbf{ then } \pi_1 \\
&\qquad\quad \textbf{else if } \mathit{condNonAct}(o_1 | \cdots | o_n, o_2) \textbf{ then } \pi_2 \\
&\qquad\quad \ldots \textbf{ else if } \mathit{condNonAct}(o_1 | \cdots | o_n, o_n) \textbf{ then } \pi_n \wedge \\
&\qquad v = v_k \wedge pr = pr_k)\,.
\end{aligned}
$$

  This is similar to the case of nondeterministic first program action with choice of agent $\boldsymbol{a}$.

- *Nondeterministic first program action (joint choice of both $\boldsymbol{a}$ and $\boldsymbol{o}$)*:
  If $\widehat{p} = [\textbf{choice}(\boldsymbol{a}\colon a_1|\cdots|a_m) \parallel \textbf{choice}(\boldsymbol{o}\colon o_1|\cdots|o_n)\,;p']$ and $h > 0$, then:

$$
\begin{aligned}
&G([\textbf{choice}(\boldsymbol{a}\colon a_1|\cdots|a_m) \parallel \textbf{choice}(\boldsymbol{o}\colon o_1|\cdots|o_n)\,;p'], s, h, \pi, v, pr) \stackrel{def}{=}\\
&\quad \exists \pi_{1,1}, \ldots, \pi_{m,n}, v_{1,1}, \ldots, v_{m,n}, pr_{1,1}, \ldots, pr_{m,n}, \pi_{\boldsymbol{a}}, \pi_{\boldsymbol{o}}\ (\textstyle\bigwedge_{i=1}^{m} \bigwedge_{j=1}^{n} G([a_i \cup o_j\,;p'],\\
&\qquad s, h, a_i \cup o_j\,; \pi_{i,j}, v_{i,j}, pr_{i,j}) \wedge \pi_{\boldsymbol{a}} \in PD(\{a_1, \ldots, a_m\}) \wedge \pi_{\boldsymbol{o}} \in PD(\{o_1, \ldots, o_n\}) \wedge\\
&\qquad \pi = \pi_{\boldsymbol{a}} \cdot \pi_{\boldsymbol{o}}\,; \textbf{if}\ condNonAct(a_1|\cdots|a_m, a_1) \wedge condNonAct(o_1|\cdots|o_n, o_1)\ \textbf{then}\ \pi_{1,1}\\
&\qquad\quad \textbf{else if}\ condNonAct(a_1|\cdots|a_m, a_2) \wedge condNonAct(o_1|\cdots|o_n, o_1)\ \textbf{then}\ \pi_{2,1}\\
&\qquad\quad \ldots\ \textbf{else if}\ condNonAct(a_1|\cdots|a_m, a_m) \wedge condNonAct(o_1|\cdots|o_n, o_n)\ \textbf{then}\ \pi_{m,n} \wedge\\
&\qquad v = \textstyle\sum_{i=1}^{m} \sum_{j=1}^{n} v_{i,j} \cdot \pi_{\boldsymbol{a}}(a_i) \cdot \pi_{\boldsymbol{o}}(o_j) \wedge pr = \sum_{i=1}^{m} \sum_{j=1}^{n} pr_{i,j} \cdot \pi_{\boldsymbol{a}}(a_i) \cdot \pi_{\boldsymbol{o}}(o_j))\,,
\end{aligned}
$$

  where $PD(\{a_1, \ldots, a_m\})$ (resp., $PD(\{o_1, \ldots, o_n\})$) denotes the set of all probability distributions over $\{a_1, \ldots, a_m\}$ (resp., $\{o_1, \ldots, o_n\}$), and $\pi_{\boldsymbol{a}} \cdot \pi_{\boldsymbol{o}}$ denotes the probability distribution over $\{a_i \cup o_j \mid i \in \{1, \ldots, m\},\ j \in \{1, \ldots, n\}\}$ that is defined by $(\pi_{\boldsymbol{a}} \cdot \pi_{\boldsymbol{o}})(a_i \cup o_j) = \pi_{\boldsymbol{a}}(a_i) \cdot \pi_{\boldsymbol{o}}(o_j)$ for all $i \in \{1, \ldots, m\}$ and $j \in \{1, \ldots, n\}$ (recall that the a_i's (resp., o_j's) are single-agent actions of agent $\boldsymbol{a}$ (resp., $\boldsymbol{o}$), and thus concurrent actions over A (resp., O)).

  Informally, every policy π of p consists of a probability distribution $\pi_{\boldsymbol{a}}$ over $a_1, \ldots, a_m$, a probability distribution $\pi_{\boldsymbol{o}}$ over $o_1, \ldots, o_n$, and one policy $\pi_{i,j}$ of p' for every possible joint action $a_i \cup o_j$. The expected reward and the success probability of π are given by the expected reward and the expected success probability of the policies $\pi_{i,j}$. Here, $\pi_{\boldsymbol{a}}$ specifies the probabilities with which agent $\boldsymbol{a}$ should execute the actions $a_1, \ldots, a_m$, while $\pi_{\boldsymbol{o}}$ specifies the probabilities with which agent $\boldsymbol{o}$ should execute the actions $o_1, \ldots, o_n$. Hence, assuming the usual probabilistic independence between the distributions $\pi_{\boldsymbol{a}}$ and $\pi_{\boldsymbol{o}}$ in stochastic games, every possible joint action $a_i \cup o_j$ is executed with the probability $(\pi_{\boldsymbol{a}} \cdot \pi_{\boldsymbol{o}})(a_i \cup o_j)$.

  For agents $\boldsymbol{a}$ and $\boldsymbol{o}$ to observe which actions among $o_1, \ldots, o_n$ and $a_1, \ldots, a_m$ were actually executed by the opponent, we use a cascade of if-then-else statements involving the conditions $condNonAct(a_1|\cdots|a_m, a_i)$ and $condNonAct(o_1|\cdots|o_n, o_j)$, respectively.

- *Test action*:
  If $\widehat{p} = [\phi?\,;p']$ and $h > 0$, then:

$$
\begin{aligned}
&G([\phi?\,;p'], s, h, \pi, v, pr) \stackrel{def}{=}\\
&\quad (\neg\phi[s] \wedge \pi = stop \wedge v = 0 \wedge pr = 0) \vee (\phi[s] \wedge G(p', s, h, \pi, v, pr))\,.
\end{aligned}
$$

  Informally, if ϕ does not hold in s, then p has only the policy $\pi = stop$ along with the expected reward $v = 0$ and the success probability $pr = 0$. Otherwise, π is a policy of p iff it is a policy of p' with the same expected reward and success probability.

- *Nondeterministic choice of two programs*:
  If $\widehat{p} = [(p_1 \mid p_2); p']$ and $h > 0$, then:

$$
\begin{aligned}
&G([(p_1 \mid p_2); p'], s, h, \pi, v, pr) \stackrel{def}{=}\\
&\quad \exists \pi_1, \pi_2, v_1, v_2, pr_1, pr_2, k\ (\textstyle\bigwedge_{j \in \{1,2\}} G([p_j; p'], s, h, \pi_j, v_j, pr_j) \wedge\\
&\qquad k \in \{1, 2\} \wedge \pi = \pi_k \wedge v = v_k \wedge pr = pr_k)\,.
\end{aligned}
$$

  Informally, π is a policy of p iff π is a policy of either $[p_1\,;p']$ or $[p_2\,;p']$ with the same expected reward and success probability.

- *Conditional*:
  If $\widehat{p} = [\textbf{if } \phi \textbf{ then } p_1 \textbf{ else } p_2; p']$ and $h > 0$, then:

  $$G([\textbf{if } \phi \textbf{ then } p_1 \textbf{ else } p_2; p'], s, h, \pi, v, pr) \overset{def}{=} G([([\phi?; p_1] \mid [\neg\phi?; p_2]); p'], s, h, \pi, v, pr)\,.$$

  This case is reduced to the cases of test action and nondeterministic choice of two programs.

- *While-loop*:
  If $\widehat{p} = [\textbf{while } \phi \textbf{ do } p; p']$ and $h > 0$, then:

  $$G([\textbf{while } \phi \textbf{ do } p; p'], s, h, \pi, v, pr) \overset{def}{=} G([[\phi?; p]^{\star}; \neg\phi?], s, h, \pi, v, pr)\,.$$

  This case is reduced to the cases of test action and nondeterministic iteration.

- *Nondeterministic choice of program argument*:
  If $\widehat{p} = [\pi[x : \tau](p(x)); p']$, where $\tau = \{\tau_1, \tau_2, \ldots, \tau_n\}$, and $h > 0$, then:

  $$G([\pi[x : \tau](p(x)); p'], s, h, \pi, v, pr) \overset{def}{=} G([(\cdots(p(\tau_1)|p(\tau_2))|\cdots|p(\tau_n)); p'], s, h, \pi, v, pr)\,.$$

  This case is reduced to the case of nondeterministic choice of two programs.

- *Nondeterministic iteration*:
  If $\widehat{p} = [p^{\star}; p']$ and $h > 0$, then:

  $$G([p^{\star}; p'], s, h, \pi, v, pr) \overset{def}{=} G([[\textbf{proc } \mathit{nit}\,(\mathit{nop} \mid [p\,;\mathit{nit}])\textbf{ end}; \mathit{nit}]; p'], s, h, \pi, v, pr)\,.$$

  This case is reduced to the cases of procedures and nondeterministic choice of two programs.

- *Procedures:* We consider the cases of (1) handling procedure declarations and (2) handling procedure calls. To this end, we slightly extend the first argument of G by a store for procedure declarations, which can be safely ignored in all the above constructs of GTGolog.

  (1) If $\widehat{p} = [\textbf{proc } P_1(\vec{x}_1)\; p_1 \textbf{ end}\,; \ldots; \textbf{proc } P_n(\vec{x}_n)\; p_n \textbf{ end}\,; p]\langle\rangle$ and $h > 0$, then:

  $$\begin{aligned}&G([\textbf{proc } P_1(\vec{x}_1)\; p_1 \textbf{ end}\,; \ldots; \textbf{proc } P_n(\vec{x}_n)\; p_n \textbf{ end}\,; p]\langle\rangle, s, h, \pi, v, pr) \overset{def}{=}\\ &\quad G([p]\langle\textbf{proc } P_1(\vec{x}_1)\; p_1 \textbf{ end}\,; \ldots; \textbf{proc } P_n(\vec{x}_n)\; p_n \textbf{ end}\rangle, s, h, \pi, v, pr)\,.\end{aligned}$$

  Informally, we store the procedure declarations at the end of the first argument of G.

  (2) If $\widehat{p} = [P_i(\vec{x}_i); p']\langle d\rangle$ and $h > 0$, then:

  $$G([P_i(\vec{x}_i); p']\langle d\rangle, s, h, \pi, v, pr) \overset{def}{=} G([p_d(P_i(\vec{x}_i)); p']\langle d\rangle, s, h, \pi, v, pr)\,.$$

  Informally, we replace a procedure call $P_i(\vec{x}_i)$ by its code $p_d(P_i(\vec{x}_i))$ from d.

We are now ready to define the notion of an H-step Nash equilibrium as follows. An H-step policy π of a GTGolog program p in a situation s relative to a domain theory DT is an *H-step Nash equilibrium* of p in s relative to DT iff (i) $U_{\boldsymbol{a}}(H, s, \pi') \leqslant U_{\boldsymbol{a}}(H, s, \pi)$ for all H-step policies π' of p in s relative to DT obtained from π by modifying only actions of agent $\boldsymbol{a}$, and (ii) $U_{\boldsymbol{o}}(H, s, \pi') \leqslant U_{\boldsymbol{o}}(H, s, \pi)$ for all H-step policies π' of p in s relative to DT obtained from π by modifying only actions of agent $\boldsymbol{o}$.

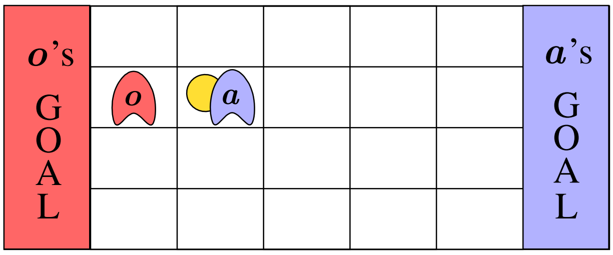


Figure 2: Rugby Domain.

Example 3.3 *(Rugby Domain cont'd)* Consider again the GTGolog procedure $game$ of Example 3.2 relative to the domain theory of Example 3.1. Let the initial situation of AT be as in Fig. 1, where agent $\boldsymbol{a}$ is at $(3,2)$, agent $\boldsymbol{o}$ is at $(2,3)$, and agent $\boldsymbol{a}$ is the ball owner in situation S_0, which is expressed by the formula $at(\boldsymbol{a},3,2,S_0) \wedge at(\boldsymbol{o},2,3,S_0) \wedge haveBall(\boldsymbol{a},S_0)$. Assuming the horizon $H=3$, a 3-step policy of $game$ along with its expected 3-step utility to agent $\boldsymbol{a}$ in situation S_0 is then given by π and $utility(v,pr)$ such that $DT \models G([game; nil], S_0, 3, \pi, v, pr)$, respectively. It is not difficult to verify that there exists a pure 3-step Nash equilibrium π of $game$ in S_0 that leads agent $\boldsymbol{a}$ to score a goal after executing three times $move(\boldsymbol{a},W)$. Suppose next that $(2,3)$ and $(1,3)$ are the initial positions of $\boldsymbol{a}$ and $\boldsymbol{o}$, respectively (see Fig. 2). Then, there exist only mixed 3-step Nash equilibria of $game$ in S_0, since any pure way of acting of $\boldsymbol{a}$ can be blocked by $\boldsymbol{o}$. Furthermore, assuming the same initial situation and a program composed of a 2-step $dribbling(2)$ (see Example 3.2) followed by the action $move(\boldsymbol{a},W)$, an associated 3-step policy along with its expected 3-step utility to agent $\boldsymbol{a}$ in situation S_0 is given by π and $utility(v,pr)$ such that $DT \models G([dribbling(2); move(\boldsymbol{a},W); nil], S_0, 3, \pi, v, pr)$, respectively. One resulting π is the fully instantiated policy for both agents $\boldsymbol{a}$ and $\boldsymbol{o}$ of utilities 507.2652 and -507.2652 that can be divided into the following two single-agent policies for agents $\boldsymbol{a}$ and $\boldsymbol{o}$, respectively (which is in fact the optimal policy computed by the GTGolog interpreter in Section 4.1; see Appendix C):

```
$\pi_{\boldsymbol{a}} = [(move(\boldsymbol{a},S),0.5042),(move(\boldsymbol{a},W),0.4958)];$
**if** $condNonAct(move(\boldsymbol{a},W))$ **then** $move(\boldsymbol{a},S)$
    **else if** $condNonAct(move(\boldsymbol{o},S))$ **then** $[(move(\boldsymbol{a},S),0.9941),(move(\boldsymbol{a},W),0.0059)]$
        **else** $move(\boldsymbol{a},W);$
$move(\boldsymbol{a},W);$

$\pi_{\boldsymbol{o}} = [(move(\boldsymbol{o},S),0.5037),(move(\boldsymbol{o},stand),0.4963)];$
**if** $condNonAct(move(\boldsymbol{a},S)) \wedge condNonAct(move(\boldsymbol{o},S))$
    **then** $[(move(\boldsymbol{o},S),0.0109),(move(\boldsymbol{o},stand),0.9891)]$
        **else** $move(\boldsymbol{o},S);$
$nop.$
```

4 A GTGolog Interpreter

In this section, we first describe a GTGolog interpreter. We then provide optimality and representation results, and we finally describe an implementation in constraint logic programming.

4.1 Formal Specification

We now define an interpreter for GTGolog programs p relative to a domain theory DT. We do this by defining the macro $DoG(\widehat{p}, s, H, \pi, v, pr)$, which takes as input the nil-terminated variant $\widehat{p}$ of a GTGolog program p, a situation s, and a finite horizon $H \geqslant 0$, and which computes as output an H-step policy π for both agents $\boldsymbol{a}$ and $\boldsymbol{o}$ in s (one among all H-step Nash equilibria of p in s; see Theorem 4.1), the expected H-step reward v (resp., $-v$) of π to agent $\boldsymbol{a}$ (resp., $\boldsymbol{o}$) in s, and the success probability pr of π in s. Thus, $\mathit{utility}(v, pr)$ (resp., $-\mathit{utility}(v, pr)$) is the expected H-step utility of π to agent $\boldsymbol{a}$ (resp., $\boldsymbol{o}$) in s. Note that if the program p fails to terminate before the horizon end is reached, then it is stopped, and the best partial policy is returned. Intuitively, in the agent programming use of GTGolog, our aim is to control agent $\boldsymbol{a}$, which is given the H-step policy π that is specified by the macro DoG for p in s, and which then executes its part of π, whereas in the game specifying use of GTGolog, we have an objective view on both agents, and thus we are interested in the H-step policy π that is specified by the macro DoG for p in s as a whole.

We define the macro $DoG(\widehat{p}, s, h, \pi, v, pr)$, for every nil-terminated variant $\widehat{p}$ of a GTGolog program p, situation s, and number of steps to go $h \in \{0, \ldots, H\}$, by induction as follows:

- The macro $DoG(\widehat{p}, s, h, \pi, v, pr)$ is defined in the same way as the macro $G(\widehat{p}, s, h, \pi, v, pr)$ for the cases null program and zero horizon, deterministic first program action, stochastic first program action (nature choice), test action, nondeterministic choice of action arguments, nondeterministic iteration, conditional, while-loop, and procedures.

- Nondeterministic first program action (choice of agent $\boldsymbol{a}$): The definition of DoG is obtained from the one of G by replacing "$k \in \{1, \ldots, m\}$" by "$k = \mathrm{argmax}_{i \in \{1,\ldots,m\}}\, \mathit{utility}(v_i, pr_i)$." Informally, given several possible actions $a_1, \ldots, a_m$ for agent $\boldsymbol{a}$, the interpreter selects an optimal one for agent $\boldsymbol{a}$, that is, an action a_i with greatest expected utility $\mathit{utility}(v_i, pr_i)$.

- Nondeterministic first program action (choice of agent $\boldsymbol{o}$): The definition of DoG is obtained from the one of G by replacing "$k \in \{1, \ldots, n\}$" by "$k = \mathrm{argmin}_{j \in \{1,\ldots,n\}}\, \mathit{utility}(v_j, pr_j)$." Informally, agent $\boldsymbol{a}$ assumes a rational behavior of agent $\boldsymbol{o}$, which is connected to minimizing the expected utility of agent $\boldsymbol{a}$ (since we consider a zero-sum setting). Hence, the interpreter selects an action o_j among $o_1, \ldots, o_n$ with smallest expected utility $\mathit{utility}(v_j, pr_j)$.

- Nondeterministic first program action (joint choice of both $\boldsymbol{a}$ and $\boldsymbol{o}$): The definition of DoG is obtained from the one of G by replacing "$\pi_{\boldsymbol{a}} \in PD(\{a_1, \ldots, a_m\}) \wedge \pi_{\boldsymbol{o}} \in PD(\{o_1, \ldots, o_n\})$" by "$(\pi_{\boldsymbol{a}}, \pi_{\boldsymbol{o}}) = \mathit{selectNash}(\{r_{i,j} = \mathit{utility}(v_{i,j}, pr_{i,j}) \mid i \in \{1, \ldots, m\}, j \in \{1, \ldots, n\}\})$." Informally, for every possible joint action choice $a_i \cup o_j$, we compute an optimal policy $\pi_{i,j}$ along with its expected reward $v_{i,j}$ and success probability $pr_{i,j}$. We then select a Nash pair $(\pi_{\boldsymbol{a}}, \pi_{\boldsymbol{o}})$ from all mixed strategies of the matrix game consisting of all $r_{i,j} = \mathit{utility}(v_{i,j}, pr_{i,j})$ with $i \in \{1, \ldots, m\}$ and $j \in \{1, \ldots, n\}$ by using the Nash selection function $\mathit{selectNash}$.

- Nondeterministic choice of two programs: The definition of DoG is obtained from the one of G by replacing "$k \in \{1, 2\}$" by "$k = \mathrm{argmax}_{i \in \{1,2\}}\, \mathit{utility}(v_i, pr_i)$." Informally, given two possible program choices p_1 and p_2, the interpreter selects an optimal one for agent $\boldsymbol{a}$, that is, a program p_i with greatest expected utility $\mathit{utility}(v_i, pr_i)$.

4.2 Optimality, Faithfulness, and Complexity Results

The following theorem shows that the macro DoG is optimal in the sense that, for every horizon $H \geqslant 0$, among the set of all H-step policies π of a GTGolog program p relative to a domain theory DT in a situation s, it computes an H-step Nash equilibrium and its expected H-step utility. The main idea behind its proof is that DoG generalizes the computation of an H-step Nash equilibrium by finite-horizon value iteration for stochastic games (Kearns et al., 2000).

Theorem 4.1 *Let $DT = (AT, ST, OT)$ be a domain theory, and let p be a GTGolog program relative to DT. Let s be a situation, let $H \geqslant 0$ be a horizon, and let $DT \models DoG(\widehat{p}, s, H, \pi, v, pr)$. Then, π is an H-step Nash equilibrium of p in s, and $utility(v, pr)$ is its expected H-step utility.*

In general, for every horizon $H \geqslant 0$, there may be exponentially many Nash equilibria among the H-step policies of a GTGolog program p. When controlling the agent $\boldsymbol{a}$ by providing it with a Nash equilibrium of p, we assume that the agent $\boldsymbol{o}$ follows a Nash equilibrium. However, we do not know which one the agent $\boldsymbol{o}$ actually uses. The next theorem shows that this is not necessary, as long as the agent $\boldsymbol{o}$ computes its Nash equilibrium in the same way as we do for the agent $\boldsymbol{a}$. That is, different Nash equilibria computed by DoG can be freely "mixed". This result follows from a similar result for matrix games (von Neumann & Morgenstern, 1947) and Theorem 4.1.

Theorem 4.2 *Let DT be a domain theory, and let p be a GTGolog program relative to DT. Let s be a situation, and let $H \geqslant 0$ be a horizon. Let π and π' be H-step policies of p in s computed by DoG using different Nash selection functions. Then, π and π' have the same expected H-step utility, and the H-step policy of p in s obtained by mixing π and π' is also an H-step Nash equilibrium.*

The following theorem shows that GTGolog programs faithfully extend stochastic games. That is, GTGolog programs can represent stochastic games, and in the special case where they syntactically model stochastic games, they are also semantically interpreted as stochastic games. Thus, GTGolog programs have a nice semantic behavior in such special cases. More concretely, the theorem says that, given any horizon $H \geqslant 0$, every zero-sum two-player stochastic game can be encoded as a program p in GTGolog, such that DoG computes one of its H-step Nash equilibria and its expected H-step reward. Here, we slightly extend basic action theories in the situation calculus by introducing one situation constant S_z for every state z of the stochastic game (see the proof of Theorem 4.3 for technical details). The theorem is proved by induction on the horizon $H \geqslant 0$, using finite-horizon value iteration for stochastic games (Kearns et al., 2000).

Theorem 4.3 *Let $G = (I, Z, (A_i)_{i \in I}, P, R)$ with $I = \{\boldsymbol{a}, \boldsymbol{o}\}$ be a zero-sum two-player stochastic game, and let $H \geqslant 0$ be a horizon. Then, there exists a domain theory $DT = (AT, ST, OT)$, a set of situation constants $\{S_z \mid z \in Z\}$, and a set of GTGolog programs $\{p^h \mid h \in \{0, \ldots, H\}\}$ relative to DT such that $\delta = (\delta_{\boldsymbol{a}}, \delta_{\boldsymbol{o}})$ is an H-step Nash equilibrium of G, where every $(\delta_{\boldsymbol{a}}(z, h), \delta_{\boldsymbol{o}}(z, h)) = (\pi_{\boldsymbol{a}}, \pi_{\boldsymbol{o}})$ is given by $DT \models DoG(\widehat{p}^h, S_z, h{+}1, \pi_{\boldsymbol{a}} \| \pi_{\boldsymbol{o}}; \pi', v, pr)$ for every state $z \in Z$ and every $h \in \{0, \ldots, H\}$. Furthermore, the expected H-step reward $G(H, z, \delta)$ is given by $utility(v, pr)$, where $DT \models DoG(\widehat{p}^H, S_z, H{+}1, \pi, v, pr)$, for every state $z \in Z$.*

The following theorem shows that using DoG for computing the H-step Nash equilibrium of a GTGolog program p relative to a domain theory DT in a situation s along with its expected H-step utility generates $O(n^{4H})$ leaves in the evaluation tree, where $H \geqslant 0$ is the horizon, and n is the maximum among (a) 2, (b) the maximum number of actions of an agent in nondeterministic (single or joint) action choices in p, (c) the maximum number of choices of nature after stochastic actions in p, and (d) the maximum number of arguments in nondeterministic choices of an argument in p. Hence, in the special case where the horizon H is bounded by a constant (which is a quite reasonable assumption in many applications in practice), this number of generated leaves is polynomial. Since in zero-sum matrix games, one Nash equilibrium along with its reward to the agents can be computed in polynomial time by linear programming (see Section 2.3), it thus follows that in the special case where (i) the horizon H is bounded by a constant, and (ii) evaluating the predicates $Poss(c, s)$, $reward(c, s)$, etc. relative to DT can be done in polynomial time, the H-step Nash equilibrium of p in s and its expected H-step utility can also be computed in polynomial time.

Theorem 4.4 *Let DT be a domain theory, and let p be a GTGolog program relative to DT. Let s be a situation, and let $H \geqslant 0$ be a horizon. Then, computing the H-step policy π of p in s and its expected H-step utility $utility(v, pr)$ via DoG generates $O(n^{4H})$ leaves in the evaluation tree.*

4.3 Implementation

We have implemented a simple GTGolog interpreter for two competing agents, where we use linear programming to calculate the Nash equilibrium at each two-agent choice step. The interpreter is realized as a constraint logic program in Eclipse 5.7 and uses the eplex library to define and solve the linear programs for the Nash equilibria. Some excerpts of the interpreter code are given in Appendix B, and we illustrate how the Rugby Domain is implemented in Prolog in Appendix C.

5 Example

In this section, we give another illustrative example for GTGolog programs. It is inspired by the stratagus domain due to Marthi et al. (2005).

Example 5.1 *(Stratagus Domain)* The stratagus field consists of 9×9 positions (see Fig. 3). There are two agents, denoted $\boldsymbol{a}$ and $\boldsymbol{o}$, which occupy one position each. The stratagus field has designated areas representing two *gold-mines*, one *forest*, and one *base* for each agent (see Fig. 3). The two agents can move one step in one of the directions *N*, *S*, *E*, and *W*, or remain stationary. Each of the two agents can also pick up one unit of wood (resp., gold) at the forest (resp., gold-mines), and drop these resources at its base. Each action of the two agents can fail, resulting in a stationary move. Any carried object drops when the two agents collide. After each step, the agents $\boldsymbol{a}$ and $\boldsymbol{o}$ receive the (zero-sum) rewards $r_{\boldsymbol{a}} - r_{\boldsymbol{o}}$ and $r_{\boldsymbol{o}} - r_{\boldsymbol{a}}$, respectively, where r_k for $k \in \{\boldsymbol{a}, \boldsymbol{o}\}$ is 0, 1, and 2 when k brings nothing, one unit of wood, and one unit of gold to its base, respectively.

The domain theory $DT = (AT, ST, OT)$ for the above stratagus domain is defined as follows. As for the basic action theory AT, we assume the deterministic actions $move(\alpha, m)$ (agent α performs m among N, S, E, W, and $stand$), $pickUp(\alpha, o)$ (agent α picks up the object o), and $drop(\alpha, o)$ (agent α drops the object o), as well as the relational fluents $at(\alpha, x, y, s)$ (agent α is at the position (x, y) in the situation s), $onFloor(o, x, y, s)$ (object o is at the position (x, y) in the situation s), and $holds(\alpha, o, s)$ (agent α holds the

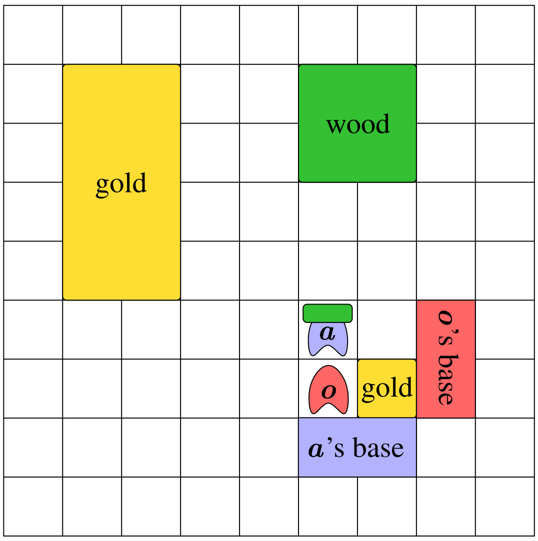


Figure 3: Stratagus Domain.

object o in the situation s), which are defined through the following successor state axioms:

$$\begin{aligned}&\mathit{at}(\alpha, x, y, \mathit{do}(c, s)) \equiv \mathit{at}(\alpha, x, y, s) \wedge \neg\exists m\,(\mathit{move}(\alpha, m) \in c) \vee\\&\quad \exists x', y'(\mathit{at}(\alpha, x', y', s) \wedge \exists m\,(\mathit{move}(\alpha, m) \in c \wedge \phi(x, y, x', y', m)))\,;\end{aligned}$$

$$\begin{aligned}&\mathit{onFloor}(o, x, y, \mathit{do}(c, s)) \equiv \mathit{onFloor}(o, x, y, s) \wedge \neg\exists\alpha\,(\mathit{pickUp}(\alpha, o) \in c) \vee\\&\quad \exists\alpha\,(\mathit{holds}(\alpha, o, s) \wedge \mathit{at}(\alpha, x, y, s) \wedge (\mathit{drop}(\alpha, o) \in c \vee \mathit{collision}(c, s)))\,;\end{aligned}$$

$$\mathit{holds}(\alpha, o, \mathit{do}(c, s)) \equiv \mathit{holds}(\alpha, o, s) \wedge \mathit{drop}(\alpha, o) \notin c \wedge \neg\mathit{collision}(c, s) \vee \mathit{pickUp}(\alpha, o) \in c\,.$$

Here, $\phi(x, y, x', y', m)$ represents the coordinate change due to m, and $\mathit{collision}(c, s)$ encodes that action c causes a collision between the agents $\boldsymbol{a}$ and $\boldsymbol{o}$ in the situation s, that is,

$$\begin{aligned}&\mathit{collision}(c, s) \overset{\mathit{def}}{=} \exists\alpha, \beta, x, y\,(\alpha \neq \beta \wedge \exists x', y'(\mathit{at}(\alpha, x', y', s) \wedge\\&\quad \exists m\,(\mathit{move}(\alpha, m) \in c \wedge \phi(x, y, x', y', m))) \wedge \exists x'', y''(\mathit{at}(\beta, x'', y'', s) \wedge\\&\quad \exists m\,(\mathit{move}(\beta, m) \in c \wedge \phi(x, y, x'', y'', m))) \wedge (x' \neq x \vee y' \neq y) \wedge (x'' \neq x \vee y'' \neq y))\,.\end{aligned}$$

The deterministic actions $\mathit{move}(\alpha, m)$, $\mathit{drop}(\alpha, o)$, and $\mathit{pickUp}(\alpha, o)$ are associated with precondition axioms as follows:

$$\begin{aligned}&\mathit{Poss}(\mathit{move}(\alpha, m), s) \equiv \neg\exists x, y\,(\mathit{at}(\alpha, x, y, s) \wedge ((y = 9 \wedge m = N) \vee\\&\quad (y = 1 \wedge m = S) \vee (x = 9 \wedge m = E) \vee (x = 1 \wedge m = W)))\,;\\&\mathit{Poss}(\mathit{drop}(\alpha, o), s) \equiv \mathit{holds}(\alpha, o, s)\,;\\&\mathit{Poss}(\mathit{pickUp}(\alpha, o), s) \equiv \neg\exists o'\, \mathit{holds}(\alpha, o', s) \wedge \exists x, y\,(\mathit{at}(\alpha, x, y, s) \wedge \mathit{onFloor}(o, x, y, s))\,.\end{aligned}$$

Here, the first axiom forbids α to go out of the 9×9 game-field. Every two-agent action consists of at most one standard action per agent, and we assume the following extra precondition axiom, which encodes that two agents cannot pick up the same object at the same time:

$$\begin{aligned}&\mathit{Poss}(\{\mathit{pickUp}(\alpha, o_1), \mathit{pickUp}(\beta, o_2)\}, s) \equiv \neg\exists o'\, \mathit{holds}(\alpha, o', s) \wedge \neg\exists o''\, \mathit{holds}(\beta, o'', s) \wedge\\&\quad \exists x, y, x', y'(\mathit{at}(\alpha, x, y, s) \wedge \mathit{onFloor}(o_1, x, y, s) \wedge \mathit{at}(\beta, x', y', s) \wedge\\&\qquad \mathit{onFloor}(o_2, x', y', s) \wedge (x \neq x' \vee y \neq y' \vee (\alpha = \beta \wedge o_1 = o_2)))\,.\end{aligned}$$

As for the stochastic theory ST, we assume the stochastic actions $moveS(\alpha, m)$ (agent α executes m among N, S, E, W, and $stand$), $pickUpS(\alpha, o)$ (agent α picks up the object o), $dropS(\alpha, o)$ (agent α drops the object o), which may succeed or fail with certain probabilities, and which are associated with their deterministic components as follows:

$$\begin{aligned}
&stochastic(\{moveS(\alpha, m)\}, s, \{a\}, p) \stackrel{def}{=} \\
&\quad a = move(\alpha, m) \wedge p = 1.0\,; \\
&stochastic(\{pickUpS(\alpha, o)\}, s, \{a\}, p) \stackrel{def}{=} \\
&\quad a = pickUp(\alpha, o) \wedge p = 0.9 \vee a = move(\alpha, stand) \wedge p = 0.1\,; \\
&stochastic(\{dropS(\alpha, o)\}, s, \{a\}, p) \stackrel{def}{=} \\
&\quad a = drop(\alpha, o) \wedge p = 0.9 \vee a = move(\alpha, stand) \wedge p = 0.1\,.
\end{aligned}$$

Here, $move(\alpha, stand)$ encodes the action failure.

As for the optimization theory OT, we use again the product as the utility function $utility$ and any suitable Nash selection function $selectNash$ for matrix games. Furthermore, we define the reward function $reward$ for agent $\boldsymbol{a}$ as follows:

$$\begin{aligned}
&reward(c, s) = r \stackrel{def}{=} \\
&\quad \exists r_{\boldsymbol{a}}, r_{\boldsymbol{o}}(rewardAct(\boldsymbol{a}, c, s) = r_{\boldsymbol{a}} \wedge rewardAct(\boldsymbol{o}, c, s) = r_{\boldsymbol{o}} \wedge r = r_{\boldsymbol{a}} - r_{\boldsymbol{o}})\,.
\end{aligned}$$

Here, $rewardAct(\alpha, c, s)$ for $\alpha \in \{\boldsymbol{a}, \boldsymbol{o}\}$ is defined as follows:

$$\begin{aligned}
&rewardAct(\alpha, c, s) = r \stackrel{def}{=} \exists o\,(drop(\alpha, o) \in c \wedge \exists x, y\,(at(\alpha, x, y, s) \wedge \\
&\qquad base(\alpha, x, y) \wedge (\neg holds(\alpha, o, s) \wedge r = 0 \vee holds(\alpha, o, s) \wedge (gold(o) \wedge r = 20 \vee \\
&\qquad wood(o) \wedge r = 10)) \vee \neg base(\alpha, x, y) \wedge (\neg holds(\alpha, o, s) \wedge r = -1 \vee \\
&\qquad holds(\alpha, o, s) \wedge (gold(o) \wedge r = -4 \vee wood(o) \wedge r = -2)))) \vee \\
&\quad \exists o\,(pickUp(\alpha, o) \in c \wedge (holds(\alpha, o, s) \wedge r = -1 \vee \neg holds(\alpha, o, s) \wedge r = 3)) \vee \\
&\quad move(\alpha, m) \in c \wedge (\neg collision(c, s) \wedge r = -1 \vee collision(c, s) \wedge (\neg \exists o\, holds(\alpha, o, s) \wedge \\
&\qquad r = -1 \vee \exists o\,(holds(\alpha, o, s) \wedge (gold(o) \wedge r = -4 \vee wood(o) \wedge r = -2))))\,.
\end{aligned}$$

Consider now the situation shown in Fig. 3. Agent $\boldsymbol{a}$ holds one unit of wood and is going towards its base, while agent $\boldsymbol{o}$ is close to the gold-mine at the corner of the two bases. How should the agents now act in such a situation? There are several aspects that the agents have to consider. On the one hand, agent $\boldsymbol{a}$ should try to move towards the base as soon as possible. On the other hand, however, agent $\boldsymbol{a}$ should also avoid to collide with agent $\boldsymbol{o}$ and lose the possession of the carried object. Hence, agent $\boldsymbol{o}$ may try to reach a collision, but agent $\boldsymbol{o}$ is also interested in picking up one unit of gold as soon as possible and then move towards its base. This decision problem for the two agents $\boldsymbol{a}$ and $\boldsymbol{o}$ is quite complex. But, assuming the finite horizon $H = 5$, a partially specified way of acting for both agents is defined by the following GTGolog program:

```
proc $schema(n)$
if $n > 0$ then [
    if $facing(\boldsymbol{a}, \boldsymbol{o})$ then $surpass(1)$
        else $dropToBase(\boldsymbol{a}, \boldsymbol{o})$;
    $schema(n-1)$]
end,
```

where $\mathit{surpass}$ and $\mathit{dropToBase}$ are defined as follows:

```
proc $\mathit{surpass}(k)$
if $k > 0$ then [
    choice($\boldsymbol{a}\colon \mathit{move}(\boldsymbol{a}, E) \mid \mathit{move}(\boldsymbol{a}, S) \mid \mathit{move}(\boldsymbol{a}, W)) \parallel$
        choice($\boldsymbol{o}\colon \mathit{move}(\boldsymbol{o}, E) \mid \mathit{move}(\boldsymbol{o}, W) \mid \mathit{move}(\boldsymbol{o}, \mathit{stand}))$;
    $\mathit{surpass}(k{-}1)$]
end;

proc $\mathit{dropToBase}(\boldsymbol{a}, \boldsymbol{o})$
if $\mathit{atBase}(\boldsymbol{a}) \wedge \mathit{atBase}(\boldsymbol{o})$ then $\pi o_1\,(\pi o_2\,(\{\mathit{drop}(\boldsymbol{a}, o_1), \mathit{drop}(\boldsymbol{o}, o_2)\}))$
    else if $\mathit{atBase}(\boldsymbol{a}) \wedge \neg \mathit{atBase}(\boldsymbol{o})$ then $\pi o_1\,(\{\mathit{drop}(\boldsymbol{a}, o_1)\})$
        else if $\neg \mathit{atBase}(\boldsymbol{a}) \wedge \mathit{atBase}(\boldsymbol{o})$ then $\pi o_2\,(\{\mathit{drop}(\boldsymbol{o}, o_2)\})$
            else $\mathit{getObject}(\boldsymbol{a}, \boldsymbol{o})$
end.
```

Here, $\mathit{getObject}$ makes the agents decide whether to move or pick up, depending on the context:

```
proc $\mathit{getObject}(\boldsymbol{a}, \boldsymbol{o})$
if $\mathit{condPickUp}(\boldsymbol{a}) \wedge \mathit{condPickUp}(\boldsymbol{o})$ then
    $\pi o_1\,(\pi o_2\,(\{\mathit{pickUp}(\boldsymbol{a}, o_1), \mathit{pickUp}(\boldsymbol{o}, o_2)\}))$
else if $\mathit{condPickUp}(\boldsymbol{a}) \wedge \neg \mathit{condPickUp}(\boldsymbol{o})$ then
    $\pi o_1\,(\{\mathit{pickUp}(\boldsymbol{a}, o_1), \mathit{move}(\boldsymbol{o}, E)\})$
else if $\neg \mathit{condPickUp}(\boldsymbol{a}) \wedge \mathit{condPickUp}(\boldsymbol{o})$ then
    $\pi o_2\,(\{\mathit{move}(\boldsymbol{a}, S), \mathit{pickUp}(\boldsymbol{o}, o_2)\})$
else if $\neg \mathit{condPickUp}(\boldsymbol{a}) \wedge \neg \mathit{condPickUp}(\boldsymbol{o})$ then
    $\{\mathit{move}(\boldsymbol{a}, S), \mathit{move}(\boldsymbol{o}, E)\}$
end,
```

where $\mathit{condPickUp}(x) \overset{\mathit{def}}{=} \neg \exists o\,(\mathit{holds}(x, o)) \wedge \mathit{atObject}(x) \wedge \neg \mathit{atBase}(x)$. Hence, we select a set of possible action choices for the two agents, and leave the charge of determining a fully instantiated policy to the GTGolog interpreter. For example, an optimal 5-step policy π that the GTGolog interpreter associates with $\mathit{schema}(5)$, along with its expected utility $\mathit{utility}(v, \mathit{pr})$ to agent $\boldsymbol{a}$ in situation S_0, is given by $\mathit{DT} \models \mathit{DoG}([\mathit{schema}(5); \mathit{nil}], S_0, 5, \pi, v, \mathit{pr})$. One such optimal policy π (of utilities 6.256 and -6.256 to agents $\boldsymbol{a}$ and $\boldsymbol{o}$, respectively) can be divided into the following two single-agent policies for agents $\boldsymbol{a}$ and $\boldsymbol{o}$, respectively:

```
$\pi_{\boldsymbol{a}} = [(\mathit{move}(\boldsymbol{a}, E), 0.5128), (\mathit{move}(\boldsymbol{a}, S), 0.4872)]$;
if $\mathit{condNonAct}(\mathit{move}(\boldsymbol{a}, E)) \wedge \mathit{condNonAct}(\mathit{move}(\boldsymbol{o}, E))$ then
    $[\mathit{move}(\boldsymbol{a}, S); \mathit{drop}(\boldsymbol{a}, p_1); \mathit{move}(\boldsymbol{a}, S); \mathit{move}(\boldsymbol{a}, S)]$
else if $\mathit{condNonAct}(\mathit{move}(\boldsymbol{a}, S)) \wedge \mathit{condNonAct}(\mathit{move}(\boldsymbol{o}, E))$ then
    $[\mathit{move}(\boldsymbol{a}, S); \mathit{drop}(\boldsymbol{a}, p_1); \mathit{move}(\boldsymbol{a}, S); \mathit{nop}]$
else if $\mathit{condNonAct}(\mathit{move}(\boldsymbol{a}, E)) \wedge \mathit{condNonAct}(\mathit{move}(\boldsymbol{o}, \mathit{stand}))$ then
    $[\mathit{move}(\boldsymbol{a}, S); \mathit{pickUp}(\boldsymbol{a}, p_1); \mathit{move}(\boldsymbol{a}, S); \mathit{drop}(\boldsymbol{a}, p_1)]$
else if $\mathit{condNonAct}(\mathit{move}(\boldsymbol{a}, S)) \wedge \mathit{condNonAct}(\mathit{move}(\boldsymbol{o}, \mathit{stand}))$ then
    $[\mathit{move}(\boldsymbol{a}, S); \mathit{drop}(\boldsymbol{a}, p_1); \mathit{move}(\boldsymbol{a}, S); \mathit{move}(\boldsymbol{a}, S)]$;
```

```
$\pi_{\boldsymbol{o}} = [(move(\boldsymbol{o}, E), 0.4872), (move(\boldsymbol{o}, stand), 0.5128)]$;
**if** $condNonAct(move(\boldsymbol{a}, E)) \wedge condNonAct(move(\boldsymbol{o}, E))$ **then**
    $[move(\boldsymbol{o}, E); drop(\boldsymbol{a}, p_1); move(\boldsymbol{a}, S); move(\boldsymbol{a}, S)]$
**else if** $condNonAct(move(\boldsymbol{a}, S)) \wedge condNonAct(move(\boldsymbol{o}, E))$ **then**
    $[pickUp(\boldsymbol{o}, p_2); nop; move(\boldsymbol{o}, E); drop(\boldsymbol{o}, p_2)]$
**else if** $condNonAct(move(\boldsymbol{a}, E)) \wedge condNonAct(move(\boldsymbol{o}, stand))$ **then**
    $[move(\boldsymbol{o}, E); pickUp(\boldsymbol{o}, p_2); move(\boldsymbol{o}, E); drop(\boldsymbol{o}, p_2)]$
**else if** $condNonAct(move(\boldsymbol{a}, S)) \wedge condNonAct(move(\boldsymbol{o}, stand))$ **then**
    $[move(\boldsymbol{o}, E); nop; pickUp(\boldsymbol{o}, p_2); move(\boldsymbol{o}, E)]$.
```

6 GTGolog with Teams

In this section, we extend the presented GTGolog for two competing agents to the case of two competing teams of agents, where every team consists of a set of cooperative agents. Here, all members of the same team have the same reward, while any two members of different teams have zero-sum rewards. Formally, we assume two competing teams $\boldsymbol{a} = \{\boldsymbol{a}_1, \ldots, \boldsymbol{a}_n\}$ and $\boldsymbol{o} = \{\boldsymbol{o}_1, \ldots, \boldsymbol{o}_m\}$ consisting of $n \geqslant 1$ agents $\boldsymbol{a}_1, \ldots, \boldsymbol{a}_n$ and $m \geqslant 1$ agents $\boldsymbol{o}_1, \ldots, \boldsymbol{o}_m$, respectively. The set of primitive actions is now partitioned into the sets of primitive actions $A_1, \ldots, A_n$, $O_1, \ldots, O_m$ of agents $\boldsymbol{a}_1, \ldots, \boldsymbol{a}_n$, $\boldsymbol{o}_1, \ldots, \boldsymbol{o}_m$, respectively. A *single-agent action* of agent $\boldsymbol{a}_i$ (resp., $\boldsymbol{o}_j$) is any concurrent action over A_i (resp., O_j), for $i \in \{1, \ldots, n\}$ and $j \in \{1, \ldots, m\}$. A *single-team action* of team $\boldsymbol{a}$ (resp., $\boldsymbol{o}$) is any concurrent action over $A_1 \cup \cdots \cup A_n$ (resp., $O_1 \cup \cdots \cup O_m$). A *two-team action* is any concurrent action over $A_1 \cup \cdots \cup A_n \cup O_1 \cup \cdots \cup O_m$.

As for the syntax of GTGolog for two competing teams of agents, the nondeterministic action choices (2)–(4) for two agents in Section 3.2 are now replaced by the following nondeterministic action choices (2′)–(4′) for two teams of agents (where $a_{i,1}, \ldots, a_{i,k_i}$ and $o_{j,1}, \ldots, o_{j,l_j}$ are single-agent actions of agents $\boldsymbol{a}_i$ and $\boldsymbol{o}_j$, for $i \in \{1, \ldots, n\}$ and $j \in \{1, \ldots, m\}$, respectively):

(2′) *Nondeterministic action choice of team $\boldsymbol{a}$*:

$$\textbf{choice}(\boldsymbol{a}_1 \colon a_{1,1} | \cdots | a_{1,k_1}) \parallel \cdots \parallel \textbf{choice}(\boldsymbol{a}_n \colon a_{n,1} | \cdots | a_{n,k_n})\,.$$

(3′) *Nondeterministic action choice of team $\boldsymbol{o}$*:

$$\textbf{choice}(\boldsymbol{o}_1 \colon o_{1,1} | \cdots | o_{1,l_1}) \parallel \cdots \parallel \textbf{choice}(\boldsymbol{o}_m \colon o_{m,1} | \cdots | o_{m,l_m})\,.$$

(4′) *Nondeterministic joint action choice*:

$$\begin{aligned}&\textbf{choice}(\boldsymbol{a}_1 \colon a_{1,1} | \cdots | a_{1,k_1}) \parallel \cdots \parallel \textbf{choice}(\boldsymbol{a}_n \colon a_{n,1} | \cdots | a_{n,k_n}) \parallel \\ &\quad \textbf{choice}(\boldsymbol{o}_1 \colon o_{1,1} | \cdots | o_{1,l_1}) \parallel \cdots \parallel \textbf{choice}(\boldsymbol{o}_m \colon o_{m,1} | \cdots | o_{m,l_m})\,.\end{aligned}$$

Informally, (2′) (resp., (3′)) now stands for "do an optimal action among $a_{i,1}, \ldots, a_{i,k_i}$ (resp., $o_{j,1}, \ldots,$ o_{j,l_j}) for every member $\boldsymbol{a}_i$ (resp., $\boldsymbol{o}_j$) of the team $\boldsymbol{a}$ (resp., $\boldsymbol{o}$)", while (4′) stands for "do any action $a_{1,p_1} \cup$ $\cdots \cup a_{n,p_n} \cup o_{1,q_1} \cup \cdots \cup o_{m,q_m}$ with an optimal probability". Observe that the selection of *exactly one action* per team member in (2′)–(4′) can be easily extended to the selection of *at most one action* per team member by simply adding the empty action *nop* to the set of actions of each agent. Similarly, nondeterministic action

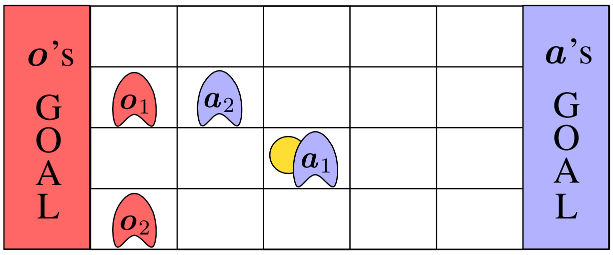


Figure 4: Rugby Domain: Two competing teams $\boldsymbol{a} = \{\boldsymbol{a}_1, \boldsymbol{a}_2\}$ and $\boldsymbol{o} = \{\boldsymbol{o}_1, \boldsymbol{o}_2\}$.

choices of a subteam of $\boldsymbol{a}$ (resp., $\boldsymbol{o}$) and nondeterministic joint action choices of subteams of $\boldsymbol{a}$ and $\boldsymbol{o}$ can also be realized by using nop. The formal semantics of (2′)–(4′) can then be defined in such a way that an optimal two-team action is chosen for each of the two teams. In particular, an H-step policy is obtained from the H-step part of an extended GTGolog program by replacing (i) every nondeterministic action choice of a team by one of its single-team actions and (ii) every nondeterministic joint action choice by a collection of probability distributions over its single-agent actions, namely one probability distribution over the single-agent actions of each agent. An optimal H-step policy of an extended GTGolog program is then chosen by (i) maximizing (resp., minimizing) the expected H-step utility and (ii) selecting a Nash equilibrium. Here, the members of every team are coordinated by assuming that (i) they select a common unique maximum (resp., minimum), which is achieved by assuming a total order on the set of all single-team actions, and (ii) they select a common unique Nash equilibrium, which is achieved by assuming that the members of every team have the same Nash selection functions.

Example 6.1 *(Rugby Domain cont'd)* We assume a team of two agents $\boldsymbol{a} = \{\boldsymbol{a}_1, \boldsymbol{a}_2\}$ against a team of two agents $\boldsymbol{o} = \{\boldsymbol{o}_1, \boldsymbol{o}_2\}$, where $\boldsymbol{a}_1$ and $\boldsymbol{o}_1$ are the *captains* of $\boldsymbol{a}$ and $\boldsymbol{o}$, respectively (see Fig. 4). An agent can 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. Each agent can do one of the following actions on each turn: N, S, E, W, $stand$, $passTo(\beta)$, and $receive$ (move up, move down, move right, move left, no move, pass, and receive the ball, respectively).

We define the domain theory $DT = (AT, ST, OT)$ as follows. Concerning the basic action theory AT, we assume the deterministic action $move(\alpha, m)$ (encoding that agent α executes m), where $\alpha \in \boldsymbol{a} \cup \boldsymbol{o}$, $m \in \{N, S, E, W, stand, passTo(\alpha'), receive\}$, and α' is a team mate of α, and the fluents $at(\alpha, x, y, s)$ (encoding that agent α is at position (x, y) in situation s) and $haveBall(\alpha, s)$ (encoding that agent α has the ball in situation s). They are defined by the following successor state axioms, which are a slightly modified version of the successor state axioms in Example 3.1:

$$\begin{aligned}&at(\alpha, x, y, do(c, s)) \equiv at(\alpha, x, y, s) \wedge \neg\exists m\,(move(\alpha, m) \in c) \vee \\ &\quad \exists x', y', m\,(at(\alpha, x', y', s) \wedge move(\alpha, m) \in c \wedge \phi(x, y, x', y', m))\,;\end{aligned}$$

$$\begin{aligned}&haveBall(\alpha, do(c, s)) \equiv haveBall(\alpha, s) \wedge \neg\exists\beta\,(cngBall(\beta, c, s) \vee rcvBall(\beta, c, s)) \vee \\ &\quad cngBall(\alpha, c, s) \vee rcvBall(\alpha, c, s)\,.\end{aligned}$$

Here, $\phi(x, y, x', y', m)$ is as in Example 3.1, $cngBall(\alpha, c, s)$ is true iff the ball possession changes to α after an action c in s (in the cases of either an adversary block or an offside ball passage), and $rcvBall(\alpha, c, s)$ is true iff agent α (not in offside) receives the ball from the ball owner, that is,

$$\begin{aligned}&\mathit{cngBall}(\alpha, c, s) \overset{\mathit{def}}{=} \exists x, y, \beta, x', y', m\, (\mathit{at}(\alpha, x, y, s) \wedge \mathit{move}(\alpha, \mathit{stand}) \in c \wedge \beta \neq \alpha \wedge \\ &\quad \mathit{haveBall}(\beta, s) \wedge \mathit{at}(\beta, x', y', s) \wedge \mathit{move}(\beta, m) \in c \wedge \phi(x, y, x', y', m)) \vee \\ &\quad \exists \beta, \gamma, x, y, x', y'\, (\beta \neq \gamma \wedge \mathit{haveBall}(\gamma, s) \wedge \mathit{move}(\gamma, \mathit{passTo}(\beta)) \in c \wedge \mathit{at}(\beta, x, y, s) \wedge \\ &\quad \mathit{at}(\gamma, x', y', s) \wedge (\alpha = \boldsymbol{o}_1 \wedge \beta \in \boldsymbol{a} \wedge \gamma \in \boldsymbol{a} \wedge x < x' \vee \alpha = \boldsymbol{a}_1 \wedge \beta \in \boldsymbol{o} \wedge \gamma \in \boldsymbol{o} \wedge x > x'))\,;\end{aligned}$$

$$\begin{aligned}&\mathit{rcvBall}(\alpha, c, s) \overset{\mathit{def}}{=} \exists \beta, x, y, x', y'\, (\alpha \neq \beta \wedge \mathit{haveBall}(\alpha', s) \wedge \mathit{move}(\beta, \mathit{passTo}(\alpha)) \in c \wedge \\ &\quad \mathit{at}(\alpha, x, y, s) \wedge \mathit{at}(\beta, x', y', s) \wedge (\alpha \in \boldsymbol{a} \wedge \beta \in \boldsymbol{a} \wedge x \geq x' \vee \alpha \in \boldsymbol{o} \wedge \beta \in \boldsymbol{o} \wedge x \leq x'))\,.\end{aligned}$$

Furthermore, we assume similar precondition axioms as in Example 3.1.

As for the stochastic theory ST, we assume the stochastic action $\mathit{moveS}(\alpha, m)$, which represents agent α's attempt in doing $m \in \{N, S, E, W, \mathit{stand}, \mathit{passTo}(\beta), \mathit{receive}\}$. It can either succeed, and then the deterministic action $\mathit{move}(\alpha, m)$ is executed, or it can fail, and then the deterministic action $\mathit{move}(\alpha, \mathit{stand})$ (that is, no change) is executed:

$$\begin{aligned}&\mathit{stochastic}(\{\mathit{moveS}(\alpha, m)\}, s, \{a\}, p) \overset{\mathit{def}}{=} m = \mathit{stand} \wedge a = \mathit{move}(\alpha, \mathit{stand}) \wedge p = 1 \vee \\ &\quad m \neq \mathit{stand} \wedge (a = \mathit{move}(\alpha, m) \wedge p = 0.9 \vee a = \mathit{move}(\alpha, \mathit{stand}) \wedge p = 0.1)\,;\end{aligned}$$

$$\begin{aligned}&\mathit{stochastic}(\{\mathit{moveS}(\alpha, m), \mathit{moveS}(\alpha', m')\}, s, \{a_\alpha, a_{\alpha'}\}, p) \overset{\mathit{def}}{=} \\ &\quad \exists p_1, p_2\, (\mathit{stochastic}(\{\mathit{moveS}(\alpha, m)\}, s, \{a_\alpha\}, p_1) \wedge \\ &\qquad \mathit{stochastic}(\{\mathit{moveS}(\alpha', m')\}, s, \{a_{\alpha'}\}, p_2) \wedge p = p_1 \cdot p_2)\,.\end{aligned}$$

As for the optimization theory OT, two agents in the same team have common rewards, and two agents in different teams have zero-sum rewards. The reward function for team $\boldsymbol{a}$ is defined by:

$$\begin{aligned}&\mathit{reward}(c, s) = r \overset{\mathit{def}}{=} \exists \alpha\, (\mathit{goal}(\alpha, \mathit{do}(c, s)) \wedge (\alpha \in \boldsymbol{a} \wedge r = 1000 \vee \alpha \in \boldsymbol{o} \wedge r = -\,1000)) \vee \\ &\quad \neg \exists \alpha\, (\mathit{goal}(\alpha, \mathit{do}(c, s))) \wedge \mathit{evalTeamPos}(c, r, s)\,,\end{aligned}$$

where $\mathit{evalTeamPos}(c, r, s)$ estimates the reward r associated with the team $\boldsymbol{a}$ in the situation s, considering the ball possession and the positions of the agents in both teams.

The GTGolog procedure game (for two agents $\boldsymbol{a}$ and $\boldsymbol{o}$) of Example 3.1 may now be generalized to the following GTGolog procedure game'' (for two teams $\boldsymbol{a} = \{\boldsymbol{a}_1, \boldsymbol{a}_2\}$ and $\boldsymbol{o} = \{\boldsymbol{o}_1, \boldsymbol{o}_2\}$):

```
proc $game''()$
while $\neg goal(\boldsymbol{a}_1) \wedge \neg goal(\boldsymbol{a}_2) \wedge \neg goal(\boldsymbol{o}_1) \wedge \neg goal(\boldsymbol{o}_2)$ do
    choice($\boldsymbol{a}_1$: $move(\boldsymbol{a}_1, N) \mid move(\boldsymbol{a}_1, S) \mid move(\boldsymbol{a}_1, E) \mid move(\boldsymbol{a}_1, W) \mid move(\boldsymbol{a}_1, stand)$) ||
    choice($\boldsymbol{a}_2$: $move(\boldsymbol{a}_2, N) \mid move(\boldsymbol{a}_2, S) \mid move(\boldsymbol{a}_2, E) \mid move(\boldsymbol{a}_2, W) \mid move(\boldsymbol{a}_2, stand)$) ||
    choice($\boldsymbol{o}_1$: $move(\boldsymbol{o}_1, N) \mid move(\boldsymbol{o}_1, S) \mid move(\boldsymbol{o}_1, E) \mid move(\boldsymbol{o}_1, W) \mid move(\boldsymbol{o}_1, stand)$) ||
    choice($\boldsymbol{o}_2$: $move(\boldsymbol{o}_2, N) \mid move(\boldsymbol{o}_2, S) \mid move(\boldsymbol{o}_2, E) \mid move(\boldsymbol{o}_2, W) \mid move(\boldsymbol{o}_2, stand)$)
end.
```

7 Related Work

In this section, we discuss closely related work on (i) high-level agent programming, (ii) first-order decision- and game-theoretic models, and (iii) other decision- and game-theoretic models.

7.1 High-Level Agent Programming

Among the most closely related works are perhaps other recent extensions of DTGolog (Dylla, Ferrein, & Lakemeyer, 2003; Ferrein, Fritz, & Lakemeyer, 2005; Fritz & McIlraith, 2005). More precisely, Dylla et al. (2003) present IPCGolog, which is a multi-agent Golog framework for team playing. IPCGolog integrates different features like concurrency, exogenous actions, continuous change, and the possibility to project into the future. This framework is demonstrated in the robotic soccer domain (Ferrein et al., 2005). In this context, multi-agent coordination is achieved without communication by assuming that the world models of the agents do not differ too much. Differently from GTGolog, however, no game-theoretic mechanism is deployed. Fritz and McIlraith (2005) propose a framework for agent programming extending DTGolog with qualitative preferences, which are compiled into a DTGolog program, integrating competing preferences through multi-program synchronization. Here, multi-program synchronization is used to allow the execution of a DTGolog program along with a concurrent program that encodes the qualitative preferences. Qualitative preferences are ranked over the quantitative ones. Differently from our work, high-level programming is used only for a single agent and no game-theoretic technique is employed to make decisions.

A further approach that is closely related to DTGolog is *ALisp* (Andre & Russell, 2002), which is a partial programming language, which augments Lisp with a nondeterministic construct. Given a partial program, a hierarchical reinforcement learning algorithm finds a policy that is consistent with the program. Marthi et al. (2005) introduce the concurrent version of *ALisp*, a language for hierarchical reinforcement learning in multi-effector problems. The language extends *ALisp* to allow multi-threaded partial programs. In this framework, the high-level programming approach is deployed to support hierarchical reinforcement learning, however, differently from GTGolog, no background (logic-based) theory is provided and reasoning is not deployed.

7.2 First-Order Decision- and Game-Theoretic Models

Other related research deals with relational and first-order extensions of MDPs (Boutilier et al., 2001; Yoon et al., 2002; Martin & Geffner, 2004; Gardiol & Kaelbling, 2003; Sanner & Boutilier, 2005), multi-agent MDPs (Guestrin et al., 2003, 2001), and stochastic games (Finzi & Lukasiewicz, 2004b). In (Gardiol & Kaelbling, 2003), the envelope method is used over structured dynamics. An initial trajectory (an envelope of states) to the goal is provided, and then the policy is gradually refined by extending the envelope. The approach aims at balancing between fully ground and purely logical representations, and between sequential plans and full MDP policies. In (Yoon et al., 2002) and (Martin & Geffner, 2004), policies are learned through generalization from small problems represented in first-order MDPs. Boutilier et al. (2001) find policies for first-order MDPs by computing the value-function of a first-order domain. The approach provides a symbolic version of the value iteration algorithm producing logical expressions that stand for sets of underlying states. A similar approach is used in our work on relational stochastic games (Finzi & Lukasiewicz, 2004b), where a multi-agent policy is associated with the generated state formulas. In the GTGolog approach, instead, the generated policy is produced as an instance of an incomplete program.

Another first-order decision- and game-theoretic formalism is Poole's independent choice logic (ICL) (1997, 2000), which is based on acyclic logic programs under different "choices". Each choice along with the acyclic logic program produces a first-order model. By placing a probability distribution over the different choices, one then obtains a distribution over the set of first-order models. Poole's ICL can be used for logically encoding games in extensive and normal form (Poole, 1997). Differently from our work, this framework aims more at representing generalized strategies, while the problem of policy synthesis is not addressed. Furthermore, our view in this paper is more directed towards using game theory for optimal agent control in multi-agent systems.

7.3 Other Decision- and Game-Theoretic Models

Less closely related are works on factored and structured representations of decision- and game-theoretic problems. An excellent overview of factored and structured representations of decision-theoretic problems is given in (Boutilier, Dean, & Hanks, 1999), focusing especially on abstraction, aggregation, and decomposition techniques based on AI-style representations. Structured representations of games (Kearns, Littman, & Singh, 2001; Koller & Milch, 2001; Vickrey & Koller, 2002; Blum, Shelton, & Koller, 2003) exploit a notion of locality of interaction for compactly specifying games in normal and extensive form. They include graphical games (Kearns et al., 2001; Vickrey & Koller, 2002) and multi-agent influence diagrams (Koller & Milch, 2001). Graphical games compactly specify normal form games: Each player's reward function depends on a subset of players described in a graph structure. Here, an n-player normal form game is explicitly described by an undirected graph on n vertices, representing the n players, and a set of n matrices, each representing a local subgame (involving only some of the players). Multi-agent influence diagrams compactly specify extensive form games. They are an extension of influence diagrams to the multi-agent case and are represented as directed acyclic graphs over chance, decision, and utility nodes. Hence, the main focus of the above works is on compactly representing normal and extensive form games and on using these compact representations for efficiently computing Nash equilibria. Our main focus in this paper, in contrast, is on agent programming in environments with adversaries. Furthermore, from the perspective of specifying games, differently from the above works, our framework here allows for specifying the game structure using logic-based action descriptions and for encoding game runs using agent programs (which are procedurally much richer than extensive form games).

Finally, another less closely related work deals with interactive POMDPs (I-POMDPs) (Gmytrasiewicz & Doshi, 2005), which are essentially a multi-agent generalization of POMDPs, where agents maintain beliefs over physical states of the environment and over models of other agents. Hence, I-POMDPs are very different from the formalism of this paper, since they concern the partially observable case, they are not based on logic-based action descriptions along with agent programs, and they also do not use the concept of a Nash equilibrium to define optimality.

8 Conclusion

We have presented the agent programming language GTGolog, which is a combination of explicit agent programming in Golog with game-theoretic multi-agent planning in stochastic games. It is a generalization of DTGolog to multi-agent systems with two competing single agents or two competing teams of cooperative agents, where any two agents in the same team have the same reward, and any two agents in different teams have zero-sum rewards. In addition to being a language for programming agents in multi-agent systems, GTGolog can also be considered as a new language for specifying games in game theory. GTGolog allows for specifying a partial control program in a high-level logical language, which is then completed by an interpreter in an optimal way. We have defined a formal semantics of GTGolog programs in terms of a set of Nash equilibria, and we have then specified a GTGolog interpreter that computes one of these Nash equilibria. We have shown that the interpreter has other nice features. In particular, we have proved that the computed Nash equilibria can be freely mixed to form new Nash equilibria, and that GTGolog programs faithfully extend (finite-horizon) stochastic games. Furthermore, we have also shown that under suitable assumptions, computing the specified Nash equilibrium can be done in polynomial time. Finally, we have also described a first prototype implementation of a simple GTGolog interpreter.

In a companion work (Finzi & Lukasiewicz, 2005b, 2007a), we extend GTGolog to the cooperative partially observable case. We present the agent programming language POGTGolog, which combines explicit agent programming in Golog with game-theoretic multi-agent planning in partially observable stochastic games (POSGs) (Hansen et al., 2004), and which allows for modeling one team of cooperative agents under partial observability, where the agents may have different initial belief states and not necessarily the same rewards. In a closely related paper (Farinelli, Finzi, & Lukasiewicz, 2007), we present the agent programming language TEAMGOLOG for programming a team of cooperative agents under partial observability. It is based on the key concepts of a synchronization state and a communication state, which allow the agents to passively resp. actively coordinate their behavior, while keeping their belief states, observations, and activities invisible to the other agents. In another companion work (Finzi & Lukasiewicz, 2006, 2007b) to the current paper, we present an approach to adaptive multi-agent programming, which integrates GTGolog with adaptive dynamic programming techniques. It extends GTGolog in such a way that the transition probabilities and reward values of the domain need not be known in advance, and thus that the agents themselves explore and adapt these data. Intuitively, it allows the agents to on-line instantiate a partially specified behavior playing against an adversary. Differently from the classical Golog approach, here the interpreter generates not only complex sequences of actions (the policy), but also the state abstraction induced by the program at the different executive stages (machine states).

An interesting topic for future research is to explore whether GTGolog (and thus also POGTGolog) can be extended to the general partially observable case, where we have two competing agents under partial observability or two competing teams of cooperative agents under partial observability. Another interesting topic is to investigate whether POGTGolog and an eventual extension to the general partially observable case can be combined with adaptive dynamic programming along the lines of the adaptive version of GTGolog in (Finzi & Lukasiewicz, 2006, 2007b).

Appendix A: Proofs for Sections 4.2

Proof of Theorem 4.1. Let $DT = (AT, ST, OT)$ be a domain theory, let p be a GTGolog program relative to DT, let s be a situation, and let $H \geqslant 0$ be a horizon. Observe first that $DT \models DoG(\widehat{p}, s, H, \pi, v, pr)$ implies $DT \models G(\widehat{p}, s, H, \pi, v, pr)$. Hence, if $DT \models DoG(\widehat{p}, s, H, \pi, v, pr)$, then π is a H-step policy of p in s, and $utility(v, pr)$ is its expected H-step utility. Therefore, it only remains to prove the following statement: $(\star)$ if $DT \models DoG(\widehat{p}, s, H, \pi, v, pr)$, then π is an H-step Nash equilibrium of p in s. We give a proof by induction on the structure of DoG.

Basis: The statement $(\star)$ trivially holds for the null program ($\widehat{p} = nil$) and zero horizon ($H = 0$) cases. Indeed, in these cases, DoG generates only the policy $\pi = nil$.

Induction: For every program construct that involves no action choice of one of the two agents, the statement $(\star)$ holds by the induction hypothesis. We now prove $(\star)$ for the remaining constructs:

(1) *Nondeterministic action choice of agent* $\boldsymbol{a}$ *(resp.,* $\boldsymbol{o}$*):* Let $\widehat{p} = [\textbf{choice}(\boldsymbol{a}\colon a_1 | \cdots | a_m)\,; p']$, and let π be the H-step policy associated with $\widehat{p}$ via DoG. By the induction hypothesis, for every $i \in \{1, \ldots, m\}$, it holds that $DT \models DoG([a_i; p'], s, H, a_i; \pi_i, v_i, pr_i)$ implies that the policy $a_i; \pi_i$ is an H-step Nash equilibrium of the program $[a_i; p']$ in s. By construction, π is the policy with the maximal expected H-step utility among the $a_i; \pi_i$'s. Hence, any different action selection a_j would not be better for $\boldsymbol{a}$, that is,

$U_{\boldsymbol{a}}(H, s, a_j; \pi_j) \leqslant U_{\boldsymbol{a}}(H, s, \pi)$ for all $j \in \{1, \ldots, m\}$. That is, any first action deviation from π would not better for $\boldsymbol{a}$. Moreover, since each $a_i; \pi_i$ is an H-step Nash equilibrium of $[a_i; p']$ in s, also any following deviation from π would not be better for $\boldsymbol{a}$. In summary, this shows that $U_{\boldsymbol{a}}(H, s, \pi') \leqslant U_{\boldsymbol{a}}(H, s, \pi)$ for every H-step policy π' of $\widehat{p}$ in s that coincides with π on the actions of $\boldsymbol{o}$. Also for agent $\boldsymbol{o}$, any unilateral deviation π'' from π cannot be better. In fact, since $\boldsymbol{o}$ is not involved in the first action choice, $\boldsymbol{o}$ can deviate from π only after $\boldsymbol{a}$'s selection of $a_i; \pi_i$, but this would not be better for $\boldsymbol{o}$ by the induction hypothesis. Hence, $U_{\boldsymbol{o}}(H, s, \pi'') \leqslant U_{\boldsymbol{o}}(H, s, \pi)$ for every H-step policy π'' of $\widehat{p}$ in s that coincides with π on the actions of $\boldsymbol{a}$. For the case of nondeterministic action choice of agent $\boldsymbol{o}$, the line of argumentation is similar, using the minimal expected H-step utility instead of the maximal one.

(2) *Nondeterministic joint action choice:* Let $\widehat{p} = [\mathbf{choice}(\boldsymbol{a}\colon a_1 | \cdots | a_m) \parallel \mathbf{choice}(\boldsymbol{o}\colon o_1 | \cdots | o_n);\, p']$, and let π be the H-step policy that is associated with $\widehat{p}$ via DoG. By the induction hypothesis, $DT \models DoG([a_i \cup o_j; p'], s, H, a_i \cup o_j; \pi_{i,j}, v_{i,j}, pr_{i,j})$ implies that each $a_i \cup o_j; \pi_{i,j}$ is an H-step Nash equilibrium of $[a_i \cup o_j; p']$ in s. We now prove that π is an H-step Nash equilibrium of $\widehat{p}$ in s. Observe first that, by construction, π is of the form $\pi_{\boldsymbol{a}} \cdot \pi_{\boldsymbol{o}}; \pi'$, where $(\pi_{\boldsymbol{a}}, \pi_{\boldsymbol{o}})$ is a Nash equilibrium (computed via the Nash selection function $selectNash$) of the matrix game consisting of all $r_{i,j} = utility(v_{i,j}, pr_{i,j})$ with $i \in \{1, \ldots, m\}$ and $j \in \{1, \ldots, n\}$. Thus, if agent $\boldsymbol{a}$ deviates from $\pi_{\boldsymbol{a}}$ with $\pi'_{\boldsymbol{a}}$, it would not do better, that is, $U_{\boldsymbol{a}}(H, s, \pi'_{\boldsymbol{a}} \cdot \pi_{\boldsymbol{o}}; \pi') \leqslant U_{\boldsymbol{a}}(H, s, \pi_{\boldsymbol{a}} \cdot \pi_{\boldsymbol{o}}; \pi')$. The same holds for $\boldsymbol{o}$, that is, for any deviation $\pi'_{\boldsymbol{o}}$ from $\pi_{\boldsymbol{o}}$, we get $U_{\boldsymbol{o}}(H, s, \pi_{\boldsymbol{a}} \cdot \pi'_{\boldsymbol{o}}; \pi') \leqslant U_{\boldsymbol{o}}(H, s, \pi_{\boldsymbol{a}} \cdot \pi_{\boldsymbol{o}}; \pi')$. That is, any first action deviation from π would not be better for $\boldsymbol{a}$ and $\boldsymbol{o}$. Moreover, by the induction hypothesis, also any following deviation from π' would not be better for $\boldsymbol{a}$ and $\boldsymbol{o}$. In summary, this shows that $U_{\boldsymbol{a}}(H, s, \pi') \leqslant U_{\boldsymbol{a}}(H, s, \pi)$ and $U_{\boldsymbol{o}}(H, s, \pi') \leqslant U_{\boldsymbol{o}}(H, s, \pi)$ for every H-step policy π' of $\widehat{p}$ in s that coincides with π on the actions of $\boldsymbol{o}$ and $\boldsymbol{a}$, respectively.

(3) *Nondeterministic choice of two programs:* The line of argumentation is similar to the one in the case of nondeterministic action choice of agent $\boldsymbol{a}$ above. □

Proof of Theorem 4.2. Immediate by Theorem 4.1 and the result that in zero-sum matrix games, the expected reward is the same under any Nash equilibrium, and Nash equilibria can be freely "mixed" to form new Nash equilibria (von Neumann & Morgenstern, 1947). □

Proof of Theorem 4.3. Suppose that $G = (I, Z, (A_i)_{i \in I}, P, R)$ with $I = \{\boldsymbol{a}, \boldsymbol{o}\}$ is a zero-sum two-player stochastic game. Without loss of generality, let $A_{\boldsymbol{a}}$ and $A_{\boldsymbol{o}}$ be disjoint. We now construct a domain theory $DT = (AT, ST, OT)$, a set of situation constants $\{S_z \mid z \in Z\}$, and a set of GTGolog programs $\{p^h \mid h \in \{0, \ldots, H\}\}$ relative to DT such that $\delta = (\delta_{\boldsymbol{a}}, \delta_{\boldsymbol{o}})$ is an H-step Nash equilibrium of G, where every $(\delta_{\boldsymbol{a}}(z, h), \delta_{\boldsymbol{o}}(z, h)) = (\pi_{\boldsymbol{a}}, \pi_{\boldsymbol{o}})$ is given by $DT \models DoG(\widehat{p}^h, S_z, h{+}1, \pi_{\boldsymbol{a}} \cdot \pi_{\boldsymbol{o}}; \pi', v, pr)$ for every $z \in Z$ and $h \in \{0, \ldots, H\}$, and the expected H-step reward $G(H, z, \delta)$ is given by $utility(v, pr)$, where $DT \models DoG(\widehat{p}^H, S_z, H{+}1, \pi, v, pr)$, for every $z \in Z$.

The basic action theory AT comprises a situation constant S_z for every state $z \in Z$ and a fluent $state(z, s)$ that associates with every situation s a state $z \in Z$ such that $state(z, S_z)$ for all $z \in Z$. Here, every state $z \in Z$ serves as a constant, and different states are interpreted in a different way. Informally, the set of all situations is given by the set of all situations that are reachable from the situations S_z with $z \in Z$ (and thus we do not use the situation S_0), and Z partitions the set of all situations into equivalence classes (one for each $z \in Z$) via the fluent $state(z, s)$. It also comprises a deterministic action $n_{a,o,z}$ for every $(a, o) \in A_{\boldsymbol{a}} \times A_{\boldsymbol{o}}$ and $z \in Z$, which performs a transition into the situation S_z, that is, $state(z, do(n_{a,o,z}, s))$ for all states $z \in Z$ and situations s. The actions $n_{a,o,z}$ are executable in every situation s, that is, $Poss(n_{a,o,z}, s) \equiv \top$ for all states $z \in Z$ and situations s. We assume two agents $\boldsymbol{a}$ and $\boldsymbol{o}$, whose sets of actions are given by $A_{\boldsymbol{a}}$ and $A_{\boldsymbol{o}}$, respectively.

The stochastic theory ST comprises a stochastic two-agent action $\{a, o\}$ for every joint action $(a, o) \in A_{\boldsymbol{a}} \times A_{\boldsymbol{o}}$ along with the set of all axioms $\mathit{stochastic}(\{a, o\}, s, n_{a,o,z'}, P(z' \mid z, a, o))$ such that $z, z' \in Z$ and s is a situation that satisfies $\mathit{state}(z, s)$ and that contains at most $H + 1$ actions, which represent the transition probabilities for the joint action (a, o) of G.

The optimization theory OT comprises the set of all axioms $\mathit{reward}(\{n_{a,o,z'}\}, s) = R(z, a, o)$ such that $(a, o) \in A_{\boldsymbol{a}} \times A_{\boldsymbol{o}}$, $z, z' \in Z$, and s is a situation that satisfies $\mathit{state}(z, s)$ and that contains at most $H + 1$ actions, which encode the reward function of G. Let $f = \mathit{selectNash}$ be a Nash selection function for zero-sum matrix games of the form $M = (I, (A_i)_{i \in I}, S)$, and let the expected reward to agent $\boldsymbol{a}$ under the Nash equilibrium $f(M)$ be denoted by $v_f(M)$.

Finally, every program p^h is a sequence of $h{+}1$ nondeterministic joint action choices of the form **choice**$(\boldsymbol{a}\colon a_1 | \cdots | a_n) \parallel$ **choice**$(\boldsymbol{o}\colon o_1 | \cdots | o_m)$, where $a_1, \ldots, a_n$ and $o_1, \ldots, o_m$ are all the singleton subsets of $A_{\boldsymbol{a}}$ and $A_{\boldsymbol{o}}$ (representing all the actions in $A_{\boldsymbol{a}}$ and $A_{\boldsymbol{o}}$), respectively.

Observe first that $pr = 1$ for every success probability pr computed in DoG for such programs p^h. By the assumed properties of utility functions, it thus follows that $\mathit{utility}(v, pr) = v$ for every expected reward v and success probability pr computed in DoG for the programs p^h.

We now prove the statement of the theorem by induction on the horizon $H \geqslant 0$. For every state $z \in Z$ and $h \in \{0, \ldots, H\}$, let the zero-sum matrix game $G[z, h] = (I, (A_i)_{i \in I}, Q[z, h])$ be defined by $Q[z, h](a_i, o_j) = v_{i,j}$, where $v_{i,j}$ is given by $DT \models DoG([\{a_i, o_j\}; \widehat{p}^{h-1}], S_z, h{+}1, \pi_{i,j}, v_{i,j}, pr_{i,j})$. By induction on the horizon $H \geqslant 0$, we now prove that

($\star$) (i) $Q[z, 0](a_i, o_j) = R(z, a_i, o_j)$ for every state $z \in Z$, and (ii) $Q[z, h](a_i, o_j) = R(z, a_i, o_j) + \sum_{z' \in Z} P(z' | z, a_i, o_j) \cdot v_f(G[z', h{-}1])$ for every state $z \in Z$ and $h \in \{1, \ldots, H\}$.

This then implies hat $v_f(G[z, h]) = v$ and $f(G[z, h]) = (\pi_{\boldsymbol{a}}, \pi_{\boldsymbol{o}})$ are given by $DT \models DoG(\widehat{p}^h, S_z, h + 1, \pi_{\boldsymbol{a}} \cdot \pi_{\boldsymbol{o}}; \pi', v, pr)$ for every $z \in Z$ and $h \in \{0, \ldots, H\}$. Furthermore, by finite-horizon value iteration (Kearns et al., 2000), the mixed policy $\delta = (\delta_{\boldsymbol{a}}, \delta_{\boldsymbol{o}})$ that is defined by $(\delta_{\boldsymbol{a}}(z, h), \delta_{\boldsymbol{o}}(z, h)) = f(G[z, h])$, for every $z \in Z$ and $h \in \{0, \ldots, H\}$, is a H-step Nash equilibrium of G, and it holds that $G(H, z, \delta) = v_f(G[z, H])$ for every $z \in Z$. This then proves the theorem. Hence, it only remains to show by induction on the horizon $H \geqslant 0$ that ($\star$) holds, which is done as follows:

Basis: Let $H = 0$, and thus we only have to consider the case $h = 0$. Let $DT \models DoG([\{a_i, o_j\}; \widehat{p}^{-1}], S_z, 1, \pi_{i,j}, v_{i,j}, pr_{i,j})$. Using the definition of DoG for the case of stochastic first program action, we then obtain $v_{i,j} = \sum_{z' \in Z} v_{z'} \cdot \mathit{prob}(\{a_i, o_j\}, S_z, n_{a_i,o_j,z'})$, where $v_{z'}$ is given by $DT \models DoG([\{n_{a_i,o_j,z'}\}; \widehat{p}^{-1}], S_z, 1, \pi_{z'}, v_{z'}, pr_{z'})$. Using the definition of DoG for the case of deterministic first program action, we obtain $v_{z'} = v'_{z'} + \mathit{reward}(\{n_{a_i,o_j,z'}\}, S_z) = 0 + R(z, a_i, o_j)$. In summary, this shows that $v_{i,j} = \sum_{z' \in Z} R(z, a_i, o_j) \cdot \mathit{prob}(\{a_i, o_j\}, S_z, n_{a_i,o_j,z'}) = R(z, a_i, o_j)$.

Induction: Let $H > 0$. By the induction hypothesis, (i) $Q[z, 0](a_i, o_j) = R(z, a_i, o_j)$ for every state $z \in Z$ and (ii) $Q[z, h](a_i, o_j) = R(z, a_i, o_j) + \sum_{z' \in Z} P(z' | z, a_i, o_j) \cdot v_f(G[z', h{-}1])$ for every state $z \in Z$ and number of steps to go $h \in \{1, \ldots, H{-}1\}$. Furthermore, as argued above, $v_f(G[z, h]) = v$ and $f(G[z, h]) = (\pi_{\boldsymbol{a}}, \pi_{\boldsymbol{o}})$ are given by $DT \models DoG(\widehat{p}^h, S_z, h + 1, \pi_{\boldsymbol{a}} \cdot \pi_{\boldsymbol{o}}; \pi', v, pr)$ for every state $z \in Z$ and number of steps to go $h \in \{0, \ldots, H{-}1\}$. Assume that $DT \models DoG([\{a_i, o_j\}; \widehat{p}^{h-1}], S_z, h + 1, \pi_{i,j}, v_{i,j}, pr_{i,j})$. Using the definition of DoG for the case of stochastic first program action, we then obtain $v_{i,j} = \sum_{z' \in Z} \mathit{prob}(\{a_i, o_j\}, S_z, n_{a_i,o_j,z'}) \cdot v_{z'} = P(z' | z, a_i, o_j) \cdot v_{z'}$, where the value $v_{z'}$ is given by $DT \models DoG([\{n_{a_i,o_j,z'}\}; \widehat{p}^{h-1}], S_z, h + 1, \pi_{z'}, v_{z'}, pr_{z'})$. Using the definition of DoG for the case of deterministic first program action, we obtain $v_{z'} = \mathit{reward}(\{n_{a_i,o_j,z'}\}, S_z) + v'_{z'} = R(z, a_i, o_j) + v'_{z'}$. By the induction hypothesis, it follows that $v'_{z'} = v_f(G[z', h{-}1])$. In summary, this proves that $v_{i,j} = R(z, a_i, o_j) + \sum_{z' \in Z} P(z' | z, a_i, o_j) \cdot v_f(G[z', h{-}1])$. □

Proof of Theorem 4.4. The maximal number of branches that DoG can generate in one step of the horizon is achieved by combining (b) nondeterministic joint action choices with a maximum number of actions for each agent, (c) stochastic actions with a maximum number of choices of nature, and (d) nondeterministic choices of an argument with a maximum number of arguments. Since an upper bound for this maximal number is given by n^4, computing the H-step policy π of p in s and its expected H-step utility $utility(v, pr)$ via DoG generates $O(n^{4H})$ leaves in the evaluation tree. □

Appendix B: Implementation of the GTGolog Interpreter

We have realized a simple GTGolog interpreter for two competing agents, which is implemented as a constraint logic program in Eclipse 5.7 and uses the eplex library for solving linear programs. Similarly as for standard Golog, the interpreter is obtained by translating the rules of Section 3.4 into Prolog clauses, which is illustrated by the following excerpts from the interpreter code:

- Null program or zero horizon:

```
doG(P,S,0,Pi,V,Pr) :- Pi=nil, V=0, Pr=1.
doG(nil,S,H,Pi,V,Pr) :- Pi=nil, V=0, Pr=1.
```

- Deterministic first program action:

```
doG(A:C,S,H,Pi,V,Pr) :- concurrentAction(A), (not poss(A,S), Pi=stop, V=0,
     Pr=0; poss(A,S), H1 is H-1, doG(C,do(A,S),H1,Pi1,V1,Pr1), agent(Ag),
     reward(Ag,R,A,S), seq(A,Pi1,Pi), V is V1+R, Pr=Pr1).
```

  Here, $concurrentAction(C)$ means that C is a concurrent action:

```
concurrentAction([A|C]) :- not A=choice(_,_), primitive_action(A),
     concurrentAction(C).
```

- Stochastic first program action (choice of nature):

```
doG(A:B,S,H,Pi,V,Pr) :- genDetComponents(A,C,S),
     bigAndDoG(A,C,B,S,H,Pi1,V,Pr), seq(A,Pi1,Pi).

bigAndDoG(A,[],B,S,H,nil,0,0).
bigAndDoG(A,[C1|LC],B,S,H,Pi,V,Pr) :-
     doG([C1]:B,S,H,Pi1,V1,Pr1), bigAndDoG(A,LC,B,S,H,Pi2,V2,Pr2),
     prob(C1,A,S,Pr3), Pi=if(condStAct(A,C1),Pi1,Pi2), Pr is Pr1*Pr3+Pr2,
     V is V1*Pr1*Pr3+V2*Pr2.
```

  Here, $genDetComponents(A, N, S)$ defines the deterministic components N of the stochastic action A, and $prob(C, A, S, P)$ defines its associated probabilities:

```
genDetComponents([],[],S).
genDetComponents([A|LA],List,S) :- setof(X,stochastic(A,S,X,_),C),
     genDetComponents(LA,List1), append(C,List1,List).

prob(C,A,S,P) :-  stochastic(A,S,C,P), poss(C,S), !; P=0.0.
```

- Nondeterministic first program action (choice of one agent):

```
doG([choice(Ag,C1)]:E,S,H,Pi,R,Pr) :- agent(Ag), doMax(C1,E,S,H,Pi,R,Pr);
     opponent(Ag), doMin(C1,E,S,H,Pi,R,Pr).
```

Here, the predicate doMax (resp., doMin) selects an optimal policy associated with a possible choice in $C1$ (resp., $C2$):

```
doMax([A],E,S,H,Pi,R,Pr) :- doG([A]:E,S,H,Pi,R,Pr).
doMax([A|L],E,S,H,Pi,R,Pr) :- doG([A]:E,S,H,Pi1,R1,Pr1),
     doMax(L,E,S,H,Pi2,R2,Pr2), utility(Ut1,R1,Pr1), utility(Ut2,R2,Pr2),
     (Ut1>=Ut2, Pi=Pi1, R=R1, Pr=Pr1; Ut1<Ut2, Pi=Pi2, R=R2, Pr=Pr2).

doMin([A],E,S,H,Pi,R,Pr) :- doG([A]:E,S,H,Pi,R,Pr).
doMin([A|L],E,S,H,Pi,R,Pr) :- not L=[], doG([A]:E,S,H,Pi1,R1,Pr1),
     doMax([A|L],E,S,H,Pi2,R2,Pr2),utility(Ut1,R1,Pr1),utility(Ut2,R2,Pr2),
     (Ut2>=Ut1, Pi=Pi1, R=R1, Pr=Pr1; Ut2<Ut1, Pi=Pi2, R=R2, Pr=Pr2).
```

- Nondeterministic first program action (joint choice of both agents):

```
doG([choice(Ag1,C1),choice(Ag2,C2)]:E,S,H,Pi,R,Pr) :-
     agent(Ag1), opponent(Ag2), doMinMax(C1,C2,E,S,H,Pi,R,Pr);
     agent(Ag2), opponent(Ag1), doMinMax(C2,C1,E,S,H,Pi,R,Pr).
```

Here, $\mathit{doMinMax}$ provides the policy Pi and the probability Pr using a minmax algorithm over the choices $C1$ and $C2$ in S, given the horizon H and the program E:

```
doMinMax(C1,C2,E,S,H,Pi,R,Pr) :-
      doMatrix(C2,C1,E,S,H,PiMatrix,RMatrix,UtMatrix,PrMatrix),
      selectNash(StrA,StrO,UtMatrix,R), probNash(StrA,StrO,PrMatrix,Pr),
      strNash(C1,C2,StrA,StrO,PiMatrix,Pi).
```

The predicate $\mathit{doMatrix}$ defines the matrix game associated with the possible choices $C2$ and $C1$. This is encoded by the matrix of utilities $\mathit{UtMatrix}$, the matrix of rewards $\mathit{RMatrix}$, and the matrix of probabilities $\mathit{PrMatrix}$:

```
doMatrix([],B,E,S,H,[],[],[],[]).
doMatrix([A|L],B,E,S,H,[PiLine|PiSubMatrix],[RLine|RSubMatrix],
          [UtLine|UtSubMatrix],[PrLine|PrSubMatrix]) :-
     doVector(A,B,E,S,H,PiLine,RLine,UtLine,PrLine),
     doMatrix(L,B,E,S,H,PiSubMatrix,RSubMatrix,UtSubMatrix,PrSubMatrix).

doVector(A,[],E,S,H,[],[],[],[]).
doVector(A,[B|L],E,S,H,[Pi|PiM],[R|RM],[Ut|UtM],[Pr|PrM]) :-
     doG([B,A]:E,S,H,Pi1,R,Pr), seq([B,A],Pi,Pi1),
     doVector(A,L,E,S,H,PiM,RM,UtM,PrM), utility(Ut,R,Pr).
```

The predicate $\mathit{selectNash}(\mathit{StrA}, \mathit{StrO}, \mathit{UtMatrix}, R)$ solves the matrix game $\mathit{UtMatrix}$, providing the probability distributions StrA and StrO over the possible choices $C1$ and $C2$, respectively. The Nash equilibrium is computed by a constraint solver implemented in C++ using the *glpk* library for linear programming. Here, the *yield* command (see eclipse-C++ embedding library) sends the utility matrix $\mathit{UtMatrix}$ to the C++ solver receiving back the result $\mathit{result}(\mathit{StrA}, \mathit{StrO}, R)$, that is, the probability distributions StrA and StrO for the agent and the opponent, along with the utility R:

```
selectNash(StrA,StrO,UtMatrix,R) :- yield(UtMatrix,result(StrA,StrO,R)).
```

The predicate $probNash(StrA, StrO, PrMatrix, Pr)$ calculates the success probability $Pr = StrA \cdot PrMatrix \cdot StrO$, while the predicate $strNash(C1, C2, StrA, StrO, PiMatrix, Pi)$ inductively defines the Nash strategy Pi:

```
strNash(C1,C2,StrA,StrO,PiMatrix,Pi) :- genNashStrategy(C1,C2,PiMatrix,Pi1),
     Pi=alea([[C1,StrA],[C2,StrO]],Pi1).

genNashStrategy(LA,[O],[PiL],Pi) :- genNashStrategy1(O,LA,PiL,Pi), !.
genNashStrategy(LA,[O|CO],[PiL|PiMatrix],Pi) :-
     genNashStrategy1(O,LA,PiL,Pi2), genNashStrategy(LA,CO,PiMatrix,Pi3),
     Pi=if(condNonAct(O),Pi2,Pi3).

genNashStrategy1(O,[A],[Pi],Pi) :- !.
genNashStrategy1(O,[A|LA],[Pi1|PiL],Pi) :-
     genNashStrategy1(O,LA,PiL,Pi2), Pi=if(condNonAct(A),Pi1,Pi2).
```

- Test action, conditional, and while-loop (as in standard Golog):

```
doG(?(T):A,S,H,Pi,R,Pr) :- holds(T,S), doG(A,S,H,Pi,R,Pr);
     not holds(T,S), Pi=stop, V=0, Pr=0.

doG(if(T,A,B):C,S,H,Pi,R,Pr) :- holds(T,S), doG(A:C,S,H,Pi,R,Pr);
     not holds(T,S), doG(B:C,S,H,Pi,R,Pr).

doG(while(T,A):B,S,H,Pi,R,Pr):- holds(T,S),
     doG(A:while(T,A):B,S,H,Pi,R,Pr); not holds(T,S), doG(B,S,H,Pi,R,Pr).
```

- Procedures (as in standard Golog):

```
doG(A:B,S,H,Pi,R,Pr) :- proc(A,C), doG(C:B,S,H,Pi,R,Pr).
```

In the above code, the predicate $utility$ defines the utility function, and the predicates sub and $holds$ are from the standard Golog implementation:

```
utility(Ut,R,Pr) :- Ut is R*Pr.

sub(X1,X2,T1,T2) :- var(T1), T2=T1.
sub(X1,X2,T1,T2) :- not var(T1), T1=X1, T2=X2.
sub(X1,X2,T1,T2) :- not T1=X1, T1=..[F|L1], sub_list(X1,X2,L1,L2), T2=..[F|L2].

sub_list(X1,X2,[],[]).
sub_list(X1,X2,[T1|L1],[T2|L2]) :- sub(X1,X2,T1,T2), sub_list(X1,X2,L1,L2).

holds(P & Q,S) :- holds(P,S), holds(Q,S).
holds(P v Q,S) :- holds(P,S); holds(Q,S).
holds(P => Q,S) :- holds(-P v Q,S).
holds(P <=> Q,S) :- holds((P => Q) & (Q => P),S).
holds(-(-P),S) :- holds(P,S).
holds(-(P & Q),S) :- holds(-P v -Q,S).
holds(-(P v Q),S) :- holds(-P & -Q,S).
holds(-(P => Q),S) :- holds(-(-P v Q),S).
holds(-(P <=> Q),S) :- holds(-((P => Q) & (Q => P)),S).
holds(-all(V,P),S) :- holds(some(V,-P),S).
holds(-some(V,P),S) :- not holds(some(V,P),S).  % Negation
holds(-P,S) :- isAtom(P), not holds(P,S).       % by failure.
holds(all(V,P),S) :- holds(-some(V,-P),S).
holds(some(V,P),S) :- sub(V,_,P,P1), holds(P1,S).

holds(A,S) :- restoreSitArg(A,S,F), F;
     not restoreSitArg(A,S,F), isAtom(A), A.

seq(A,Pi1,A:Pi1).

isAtom(A) :- not (A=-W; A=(W1 & W2); A=(W1 => W2); A=(W1 <=> W2);
     A=(W1 v W2); A=some(X,W); A=all(X,W)).
```

Appendix C: Implementation of the Rugby Domain

The domain theory of the Rugby Domain in Examples 3.1 to 3.3 is implemented by the following Prolog program, which encodes its basic action theory and its optimization theory.

We first declare two players, that is, the agent $\boldsymbol{a}$ and its opponent $\boldsymbol{o}$, and we encode a game configuration that represents the initial state of the world: We consider an initial situation S_0, where agent $\boldsymbol{a}$ is in position $(2, 3)$ and has the ball, and agent $\boldsymbol{o}$ is in position $(1, 3)$:

```
agent(a). opponent(o).

at(a,2,3,s0). haveBall(a,s0). at(o,1,3,s0).
```

The action $move(\alpha, m)$ described in Example 3.1 is encoded by the action $move(\alpha, x, y)$, where the arguments x and y represent horizontal and vertical shifts, respectively, that is, N, S, E, W, and $stand$ are encoded by $(0, 1)$, $(0, -1)$, $(1, 0)$, $(-1, 0)$, and $(0, 0)$, respectively. The fluents $at(\alpha, x, y, s)$, $haveBall(\alpha, s)$, and $cngBall(\alpha, c, s)$ require the following successor state axioms:

```
at(Ag,X,Y,do(C,S)) :- at(Ag,X,Y,S), not member(move(Ag,X1,Y1),C);
     at(Ag,X2,Y2,S), member(move(Ag,DX,DY),C), X is X2+DX, Y is Y2+DY.

haveBall(Ag1,do(C,S)) :- (agent(Ag1), opponent(Ag2); agent(Ag2), opponent(Ag1)),
     (haveBall(Ag1,S), not cngBall(Ag2,C,S); cngBall(Ag1,C,S)).

cngBall(Ag1,C,S) :- (agent(Ag1), opponent(Ag2); agent(Ag2), opponent(Ag1)),
     at(Ag1,X,Y,S), member(move(Ag1,0,0),C), at(Ag2,X1,Y1,S), haveBall(Ag2,S),
     member(move(Ag2,DX,DY),C), X2 is X1+DX, Y2 is Y1+DY, X2=X, Y2=Y.
```

We next define the preconditions $Poss(a, s)$ for each primitive action a in situation s, and (as in concurrent Golog) the preconditions $Poss(c, s)$ for each concurrent action c in s, where the latter here require that all the primitive actions mentioned in c are executable in s:

```
poss(move(Ag,X,Y),S) :- (X=0; Y=0), (X=1; X=-1; X=0), (Y=1; Y=-1; Y=0),
     at(Ag,X2,Y2,S), (X2=0, not X=-1; X2=6, not X=1; Y2=1, not Y=-1;
     Y2=4, not Y=1).
```

```
poss([move(Ag1,X1,Y1), move(Ag2,X2,Y2)],S) :-
     poss(move(Ag1,X1,Y1),S), poss(move(Ag2,X2,Y2), not Ag1=Ag2.

poss(C,S) :- allPoss(C,S).

allPoss([],S).
allPoss([A|R],S) :- poss(A,S), allPoss(R,S).
```

We finally represent the function $\mathit{reward}(c, s)$ through the predicate $\mathit{reward}(\alpha, r, c, s)$, which gives a high (resp., low) reward r in the case of a goal by α (resp., the adversary of α), and the reward r depends on the positions of the agents $\boldsymbol{a}$ and $\boldsymbol{o}$, as defined by $\mathit{evalPos}(\alpha, c, r, s)$, otherwise:

```
reward(Ag,R,C,S) :- goal(Ag1,do(C,S)), (Ag1=Ag, R is 1000;
     not Ag1=Ag, R is -1000), !; evalPos(Ag,C,R,S).

evalPos(Ag,C,R,S) :- haveBall(Ag1,do(C,S)), at(Ag1,X,Y,do(C,S)),
     (Ag=o, Ag1=o, R is X; Ag=o, Ag1=a, R is X-6;
     Ag=a, Ag1=a, R is 6-X; Ag=a, Ag1=o, R is -X).

goal(Ag,S) :- haveBall(Ag,S), at(Ag,X,Y,S), goalPos(Ag,X,Y).

goalPos(a,0,Y) :- Y=1; Y=2; Y=3; Y=4.
goalPos(o,6,Y) :- Y=1; Y=2; Y=3; Y=4.
```

Given the domain theory, we can formulate a GTGolog program. For example, consider the following program, which coincides with $\mathit{dribbling}(2); \mathit{move}(\boldsymbol{a}, W)$ (see Example 3.2), where twice agent $\boldsymbol{a}$ (resp., $\boldsymbol{o}$) can move either S or W (resp., stand), and then agent $\boldsymbol{a}$ moves W:

```
proc(schema,
  [choice(a,[move(a,0,-1),move(a,-1,0)]),choice(o,[move(o,0,-1),move(o,0,0)])]:
  [choice(a,[move(a,0,-1),move(a,-1,0)]),choice(o,[move(o,0,-1),move(o,0,0)])]:
  [move(a,-1,0)]).
```

Informally, the two agents $\boldsymbol{a}$ and $\boldsymbol{o}$ are facing each other. The former has to perform a dribbling in order to score a goal, while the latter can try to guess $\boldsymbol{a}$'s move to change the ball possession. This action requires a mixed policy, which can be generated by the following query:

```
:- doG(schema:nil,s0,3,Pi,R,Pr).
```

The result of the previous query is a fully instantiated policy π for both agents $\boldsymbol{a}$ and $\boldsymbol{o}$, which can be divided into the following two single-agent policies $\pi_{\boldsymbol{a}}$ and $\pi_{\boldsymbol{o}}$ for agents $\boldsymbol{a}$ and $\boldsymbol{o}$, respectively:

```
[move(a,0,-1),move(a,-1,0)]:[0.5042,0.4958];
if condNonAct(move(a,-1,0))
   then move(a,0,-1)
   else if condNonAct(move(o,0,-1))
      then [move(a,0,-1),move(a,-1,0)]:[0.9941,0.0059]
      else move(a,-1,0);
move(a,-1,0);

[move(o,0,-1),move(o,0,0)]:[0.5037,0.4963];
if condNonAct(move(a,0,-1)) and condNonAct(move(o,0,-1))
   then [move(o,0,-1),move(o,0,0)]:[0.0109,0.9891]
   else move(o,0,-1);
nop.
```

The other computed results (in 0.27s cpu time), namely, the expected 3-step reward r and the success probability pr of the computed 3-step policy are given as follows:

```
R = 507.2652
Pr = 1.0
```

References

Andre, D., & Russell, S. J. (2002). State abstraction for programmable reinforcement learning agents. In *Proceedings AAAI-2002*, pp. 119–125. AAAI Press.

Bacchus, F., Halpern, J. Y., & Levesque, H. J. (1999). Reasoning about noisy sensors and effectors in the situation calculus. *Artif. Intell.*, *111*(1–2), 171–208.

Baral, C., Tran, N., & Tuan, L.-C. (2002). Reasoning about actions in a probabilistic setting. In *Proceedings AAAI-2002*, pp. 507–512. AAAI Press.

Blum, B., Shelton, C. R., & Koller, D. (2003). A continuation method for Nash equilibria in structured games. In *Proceedings IJCAI-2003*, pp. 757–764. Morgan Kaufmann.

Boutilier, C., Dean, T., & Hanks, S. (1999). Decision-theoretic planning: Structural assumptions and computational leverage. *J. Artif. Intell. Res.*, *11*, 1–94.

Boutilier, C., Reiter, R., & Price, B. (2001). Symbolic dynamic programming for first-order MDPs. In *Proceedings IJCAI-2001*, pp. 690–700. Morgan Kaufmann.

Boutilier, C., Reiter, R., Soutchanski, M., & Thrun, S. (2000). Decision-theoretic, high-level agent programming in the situation calculus. In *Proceedings AAAI-2000*, pp. 355–362. AAAI Press/MIT Press.

Dylla, F., Ferrein, A., & Lakemeyer, G. (2003). Specifying multirobot coordination in ICPGolog – from simulation towards real robots. In *Proceedings AOS-2003*.

Eiter, T., & Lukasiewicz, T. (2003). Probabilistic reasoning about actions in nonmonotonic causal theories. In *Proceedings UAI-2003*, pp. 192–199. Morgan Kaufmann.

Farinelli, A., Finzi, A., & Lukasiewicz, T. (2007). Team programming in Golog under partial observability. In *Proceedings IJCAI-2007*, pp. 2097–2102. AAAI Press/IJCAI.

Ferrein, A., Fritz, C., & Lakemeyer, G. (2005). Using Golog for deliberation and team coordination in robotic soccer. *Künstliche Intelligenz*, *1*, 24–43.

Finzi, A., & Pirri, F. (2001). Combining probabilities, failures and safety in robot control. In *Proceedings IJCAI-2001*, pp. 1331–1336. Morgan Kaufmann.

Finzi, A., & Lukasiewicz, T. (2003). Structure-based causes and explanations in the independent choice logic. In *Proceedings UAI-2003*, pp. 225–232. Morgan Kaufmann.

Finzi, A., & Lukasiewicz, T. (2004a). Game-theoretic agent programming in Golog. In *Proceedings ECAI-2004*, pp. 23–27. IOS Press.

Finzi, A., & Lukasiewicz, T. (2004b). Relational Markov games. In *Proceedings JELIA-2004*, Vol. 3229 of *LNCS/LNAI*, pp. 320–333. Springer.

Finzi, A., & Lukasiewicz, T. (2005a). Game-theoretic reasoning about actions in nonmonotonic causal theories. In *Proc. LPNMR-2005*, Vol. 3662 of *LNCS/LNAI*, pp. 185–197. Springer.

Finzi, A., & Lukasiewicz, T. (2005b). Game-theoretic Golog under partial observability (poster). In *Proceedings AAMAS-2005*, pp. 1301–1302. ACM Press.

Finzi, A., & Lukasiewicz, T. (2007a). Game-theoretic agent programming in Golog under partial observability. In *Proceedings KI-2006*, Vol. 4314 of *LNCS/LNAI*, pp. 389–403. Springer. Extended Report 1843-05-02, Institut für Informationssysteme, TU Wien, December 2006.

Finzi, A., & Lukasiewicz, T. (2006). Adaptive multi-agent programming in GTGolog (poster). In *Proceedings ECAI-2006*, pp. 753–754. IOS Press.

Finzi, A., & Lukasiewicz, T. (2007b). Adaptive multi-agent programming in GTGolog. In *Proceedings KI-2006*, Vol. 4314 of *LNCS/LNAI*, pp. 113–127. Springer.

Fritz, C., & McIlraith, S. (2005). Compiling qualitative preferences into decision-theoretic Golog programs. In *Proceedings NRAC-2005*.

Gardiol, N. H., & Kaelbling, L. P. (2003). Envelope-based planning in relational MDPs. In *Proceedings NIPS-2003*. MIT Press.

Goldman, C. V., & Zilberstein, S. (2004). Decentralized control of cooperative systems: Categorization and complexity analysis. *J. Artif. Intell. Res.*, *22*, 143–174.

Gmytrasiewicz, P. J., & Doshi, P. (2005). A framework for sequential planning in multi-agent settings. *J. Artif. Intell. Res.*, *24*, 49–79.

Grosskreutz, H., & Lakemeyer, G. (2001). Belief update in the pGOLOG framework. In *Proceedings KI/ÖGAI-2001*, Vol. 2174 of *LNCS/LNAI*, pp. 213–228. Springer.

Guestrin, C., Koller, D., Gearhart, C., & Kanodia, N. (2003). Generalizing plans to new environments in relational MDPs. In *Proceedings IJCAI-2003*. Morgan Kaufmann.

Guestrin, C., Koller, D., & Parr, R. (2001). Multiagent planning with factored MDPs. In *Proceedings NIPS-2001*, pp. 1523–1530. MIT Press.

Hansen, E. A., Bernstein, D. S., & Zilberstein, S. (2004). Dynamic programming for partially observable stochastic games. In *Proceedings AAAI-2004*, pp. 709–715. AAAI Press/MIT Press.

Iocchi, L., Lukasiewicz, T., Nardi, D., & Rosati, R. (2004). Reasoning about actions with sensing under qualitative and probabilistic uncertainty. In *Proc. ECAI-2004*, pp. 818–822. IOS Press.

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

Kearns, M. J., Mansour, Y., & Singh, S. P. (2000). Fast planning in stochastic games. In *Proceedings UAI-2000*, pp. 309–316. Morgan Kaufmann.

Kearns, M. J., Littman, M. L., & Singh, S. P. (2001). Graphical models for game theory. In *Proceedings UAI-2001*, pp. 253–260. Morgan Kaufmann.

Koller, D., & Milch, B. (2001). Multi-agent influence diagrams for representing and solving games. In *Proceedings IJCAI-2001*, pp. 1027–1036. Morgan Kaufmann.

Littman, M. L. (1994). Markov games as a framework for multi-agent reinforcement learning. In *Proceedings ICML-1994*, pp. 157–163. Morgan Kaufmann.

Marthi, B., Russell, S. J., Latham, D., & Guestrin, C. (2005). Concurrent hierarchical reinforcement learning. In *Proceedings IJCAI-2005*, pp. 779–785. Professional Book Center.

Martin, M., & Geffner, H. (2004). Learning generalized policies from planning examples using concept languages. *Appl. Intell.*, *20*(1), 9–19.

Mateus, P., Pacheco, A., Pinto, J., Sernadas, A., & Sernadas, C. (2001). Probabilistic situation calculus. *Ann. Math. Artif. Intell.*, *32*(1–4), 393–431.

McCarthy, J., & Hayes, P. J. (1969). Some philosophical problems from the standpoint of artificial intelligence. In *Machine Intelligence*, Vol. 4, pp. 463–502. Edinburgh University Press.

Nair, R., Tambe, M., Yokoo, M., Pynadath, D. V., & Marsella, S. (2003). Taming decentralized POMDPs: Towards efficient policy computation for multiagent settings. In *Proceedings IJCAI-2003*, pp. 705–711. Morgan Kaufmann.

Owen, G. (1982). *Game Theory: Second Edition*. Academic Press.

Peshkin, L., Kim, K.-E., Meuleau, N., & Kaelbling, L. P. (2000). Learning to cooperate via policy search. In *Proceedings UAI-2000*, pp. 489–496. Morgan Kaufmann.

Pinto, J. (1998). Integrating discrete and continuous change in a logical framework. *Computational Intelligence*, *14*(1), 39–88.

Poole, D. (1997). The independent choice logic for modelling multiple agents under uncertainty. *Artif. Intell.*, *94*(1–2), 7–56.

Poole, D. (2000). Logic, knowledge representation, and Bayesian decision theory. In *Proceedings CL-2000*, Vol. 1861 of *LNCS*, pp. 70–86. Springer.

Puterman, M. L. (1994). *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley.

Reiter, R. (1991). The frame problem in the situation calculus: A simple solution (sometimes) and a completeness result for goal regression. In *Artificial Intelligence and Mathematical Theory of Computation: Papers in Honor of John McCarthy*, pp. 359–380. Academic Press.

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

Sanner, S., & Boutilier, C. (2005). Approximate linear programming for first-order MDPs. In *Proceedings UAI-2005*, pp. 509–517.

van der Wal, J. (1981). *Stochastic Dynamic Programming*, Vol. 139 of *Mathematical Centre Tracts*. Morgan Kaufmann.

Vickrey, D., & Koller, D. (2002). Multi-agent algorithms for solving graphical games. In *Proceedings AAAI-2002*, pp. 345–351. AAAI Press.

von Neumann, J., & Morgenstern, O. (1947). *The Theory of Games and Economic Behavior*. Princeton University Press.

Yoon, S. W., Fern, A., & Givan, R. (2002). Inductive policy selection for first-order MDPs. In *Proceedings UAI-2002*, pp. 568–576. Morgan Kaufmann.