Strategy Representation and Reasoning for Incomplete Information Concurrent Games in the Situation Calculus

Liping Xiong and Yongmei Liu
Dept. of Computer Science
Sun Yat-sen University, Guangzhou 510006, China
xionglp3@mail2.sysu.edu.cn, ymliu@mail.sysu.edu.cn

Abstract
Strategy representation and reasoning for incomplete information concurrent games has recently received much attention in multi-agent system and AI communities. However, most of the logical frameworks are based on concrete game models, lack the abilities to reason about strategies explicitly or specify strategies procedurally, and ignore the issue of coordination within a coalition. In this paper, by a simple extension of a variant of multi-agent epistemic situation calculus with a strategy sort, we develop a general framework for strategy representation and reasoning for incomplete information concurrent games. Based on Golog, we propose a strategy programming language which can be conveniently used to specify collective strategies of coalitions at different granularities. We present a formalization of joint abilities of coalitions under commitments to strategy programs. Different kinds of individual strategic abilities can be distinguished in our framework. Both strategic abilities in ATL and joint abilities of Ghaderi et al. can be considered as joint abilities under special programs in our framework. We illustrate our work with a variant of Levesque’s Squirrels World.

1 Introduction
Strategy representation and reasoning has recently received much attention in multi-agent system and AI communities. Many strategic logics have been established, and most of them are built upon Alternating-time Temporal Logic (ATL) [Alur et al., 2002] where formula $\langle G \rangle \phi$ expresses that coalition $G$ has a group strategy to ensure temporal goal $\phi$ holds no matter what the other agents do.

Many extensions of ATL have been proposed, mainly along two lines. On one hand, in reality, players often have incomplete information about the game states, e.g., in poker games. Although they may have strategies to ensure a goal holds, they may not know what these strategies are or how to execute strategies where the prescribed actions for indistinguishable states are not the same. Extensions of ATL are proposed to deal with such games [van der Hoek and Wooldridge, 2003; Jamroga and van der Hoek, 2004; Jamroga and Ågotnes, 2007]. On the other hand, strategies are treated implicitly in ATL, and extensions of ATL are proposed to reason about strategies explicitly, that is, treat strategies as first-order objects [Walther et al., 2007; Mogavero et al., 2010].

However, the above extensions suffer from other limitations. Firstly, the extensions are mainly propositional, hence lacking expressiveness to compactly specify game structures or indistinguishable states. Secondly, as pointed out by [Ramanujam and Simon, 2008; van Eijck, 2013], these extensions treat strategies as abstract objects rather than considering the internal structure of strategies, that is, the combination of basic strategies to form complex ones. Thirdly, most extensions ignore the coordination problem. As discussed in [Ghaderi et al., 2007], a coalition may have many group strategies to ensure a goal, yet a player may not know other players’ choices, hence the coalition may end up with a group strategy which may not ensure the goal.

Other than modal logics, another main family of logics in AI is action formalisms. A prominent example of action formalisms is the situation calculus [Reiter, 2001], which is a first-order language with some second-order ingredients suitable for reasoning about actions and change. Based on the situation calculus, a logic programming language Golog [Levesque et al., 1997] has been designed for high-level agent control. There have been a few works studying strategic reasoning in the situation calculus [Schulte and Delgrande, 2004; Farinelli et al., 2007; De Giacomo et al., 2010; Ghaderi et al., 2007]. The first one deals with von Neumann Morgenstern games, while the second one focuses on Markov games, using Golog to specify agent behavior. The third work studies complete information turn-based games, where ConGolog [De Giacomo et al., 2000], a concurrent version of Golog, is also used to specify game structures. However, these works either focus on solution concepts like Nash equilibria rather than ATL-like properties, or cannot reason about strategies explicitly. Lastly, Ghaderi et al. study the coordination problem and present a formalization of joint ability of coalitions based on the idea of iterated elimination of dominated strategies [Osborne and Rubinstein, 1999]. Nonetheless, it is desirable to have a more general account of joint ability of a coalition under constraints, which maybe a rough collective strategy which the coalition commits to, or a protocol that the players must comply with, such as traffic rules.

In this paper, we propose a framework based on the sit-
u1323nation calculus for strategy representation and reasoning for incomplete information concurrent games. We first propose a simple extension with a strategy sort of a concurrent variant of multi-agent epistemic situation calculus, which can be used to compactly represent possibly infinite concurrent games, and reason about strategies explicitly. Then we propose a strategy programming language based on Golog, which can be used to specify collective strategies of coalitions. We emphasize that in this paper, by the word “collective strategy”, we mean group strategy which is common knowledge of the coalition. Next, we adapt the approach of Ghaderi et al. to formalize joint ability of coalitions under commitments to strategy programs. We illustrate our logical framework with a variant of Levesque’s Squirrels World.

2 Preliminaries

In this section, we introduce the situation calculus and Golog.

The situation calculus [Reiter, 2001] is a many-sorted first-order logic language (with some second-order elements) specifically designed for representing dynamically changing worlds. There are three disjoint sorts: situation for situations, action for actions, and object for everything else. In this language, the constant $S_0$ is used to denote the initial situation; the binary function $do(a, s)$ is used to denote the successor situation of $s$ resulting from performing action $a$, and $do([a_1, a_2, ... , a_k], s)$ is used as a shorthand for $do(a_k, do(a_{k-1}, ... do(a_2, do(a_1, s))));$ the binary predicate $Poss(a, s)$ means that action $a$ is possible in situation $s$. Actions can be parameterized, e.g., $pick(r, x)$ represents robot $r$ picking up object $x$. There are relational and functional fluents whose values vary from situation to situation. These fluents are denoted by symbols that take a situation term as their last argument. There are also situation-independent predicates and functions. Finally, there is a binary predicate $\sqsubseteq$ on situations: $s \sqsubseteq s'$ means that $s$ is a proper subhistory of $s'$. We use $s \sqsubseteq s'$ as a shorthand for $s \subseteq s' \land s \neq s'$, and we let $s \subseteq s'$ abbreviate for $s \sqsubseteq s' \land \forall a \forall s''(s \sqsubseteq do(a, s'') \subseteq s'' \sqsubseteq poss(a, s''))$, meaning that $s$ is a subhistory of $s'$ and every action on the way from $s$ to $s'$ is possible. We say that a situation $s$ is executable if it is possible to perform the actions in $s$ one by one: $Exec(s) \equiv \forall a, s'.do(a, s') \subseteq s \sqsubseteq poss(a, s')$.

In this language, an application domain is specified by a basic action theory (BAT) which describes how the world changes as the result of the available actions. Each BAT $D$ consists of the following five disjoint parts:

1. $\Sigma$, the foundational axioms of the situation calculus;
2. $P_{op}$, a precondition axiom for each action specifying when the action can be legally performed;
3. $P_{ss}$, a successor state axiom (SSA) for each fluent which describes how fluent values change between situations;
4. $P_{una}$, unique name axioms for actions;
5. $P_{So}$, axioms describing the initial situation $S_0$.

Liu and Levesque [2014] propose a multi-agent extension of the situation calculus. They use a special fluent $B(i, s', s)$, which means that agent $i$ considers situation $s'$ accessible from situation $s$, and introduce a special predicate $A(i, a', a, s)$, meaning that in situation $s$, agent $i$ considers action $a'$ as a possible alternative of action $a$. The following is their successor state axiom for the $B$ fluent:

$$B(i, s', do(a, s)) \equiv \exists a^*.B(i, s^*, s) \land A(i, a^*, a, s) \land (Poss(a, s) \supset Poss(a^*, s^*)) \land s' = do(a^*, s^*)$$

Intuitively, for agent $i$, situation $s'$ is accessible after action $a$ is performed in situation $s$ if it is the result of doing some alternative $a^*$ of $a$ in some $s^*$ accessible from $s$, and executability of $a$ in $s$ implies that of $a^*$ in $s^*$. Then beliefs are defined as follows. Let $\phi(s)$ be a formula with a single situation variable $s$.

- Agent $i$ believes $\phi$:
  $$Bel(i, \phi(now), s) \equiv \forall s'.B(i, s', s) \supset \phi(s')$$

- Agent $i$ truly believes $\phi$:
  $$TBel(i, \phi(now), s) \equiv \phi(s) \land Bel(i, \phi(now), s)$$

Based on the situation calculus, Golog [Levesque et al., 1997] has been proposed to represent complex actions obtained by the combinations of primitive actions. Golog programs are defined by the following constructs:

1. $a$, primitive action;
2. $\delta_1 \delta_2$, action sequence;
3. $\varphi?$, test;
4. $\delta_1 | \delta_2$, nondeterministic choice of actions;
5. $\pi.x, \delta$, nondeterministic choice of arguments;
6. $\delta^*$, nondeterministic iteration.

Conditionals and loops are defined as abbreviations. The formal semantics of Golog is usually specified by an abbreviation $Do(\delta, s, s')$, which intuitively means that executing $\delta$ brings us from situation $s$ to $s'$. Another form of semantics for Golog is the transition semantics, which is based on the important concept of a configuration, denoted as a pair $(\delta, \sigma)$, where $\delta$ is a program (that remains to be executed) and $\sigma$ is a situation (of actions that have been performed). As presented in [De Giacomo et al., 2000], the transition semantics for Golog programs is defined by two predicates $Trans(\delta, \sigma, \delta', \sigma')$ and $Final(\delta, \sigma)$.

3 A concurrent epistemic situation calculus

In this section, we present a concurrent epistemic situation calculus with a strategy sort, which can be used to compactly represent incomplete information games where a fixed finite set of agents act simultaneously and instantly at each game step, and to reason about strategies explicitly.

We fix a set of agents $AG = \{1, \ldots, n\}$; we introduce an agent sort and $n$ agent constants, for which unique names and domain closure hold. We introduce two additional sorts: an action profile sort and a second-order strategy sort. Intuitively, an action profile is an $n$-ary vector of actions, one action for each agent. A strategy is a function from situations to actions. Let $G \subseteq AG$, a group strategy of coalition $G$ is a function from $G$ to strategies. A strategy profile is a group
strategy of AG. We use variables $d$, $d'$, . . . for action profiles, $f$, $f'$, . . . for strategies, $f_G$, $f'_G$, . . . for group strategies of $G$, and $G$ to represent $AG \setminus G$. We treat $f_G$ the same as the set of variables $\{f_i \mid i \in G\}$. When $G$ is a singleton $\{i\}$, we simply write $i$.

We introduce a function $join(a_1, \ldots, a_n)$ which maps $n$ actions into an action profile, and $n$ projection functions $pr_i(d), 1 \leq i \leq n$, which maps an action profile into its $i$-th component. For simplicity, we write $join(a_1, \ldots, a_n)$ as $(a_1, \ldots, a_n)$, and write $pr_i(d)$ as $d_i$. Situations are now sequences of action profiles; so the first argument of the function $do$ and the predicate $Poss$ is of the action profile sort.

The set $\Sigma$ of foundational axioms for situations is the same as before except that we replace each action variable with an action profile variable. We also add to $\Sigma$ the following axioms concerning action profiles:

- $\forall d \exists a_1, \ldots, a_n. d = (a_1, \ldots, a_n)$;
- $(a_1, \ldots, a_n) = (a'_1, \ldots, a'_n) \iff a_1 = a'_1 \wedge \ldots \wedge a_n = a'_n$;
- $pr_i((a_1, \ldots, a_n)) = a_i, i = 1, \ldots, n$.

Reiter [2001] presents an account of true concurrency where a concurrent action is modeled as a possibly infinite set of simple actions. Our account of concurrent actions as action profiles can be viewed as a special case of Reiter’s account.

To specify the preconditions axioms for action profiles, we introduce $n$ predicates $Poss_i(a, s)$, meaning that it is possible for agent $i$ to perform action $a$ in situation $s$. In general, the following holds: $Poss((a_1, \ldots, a_n), s) = \bigwedge_{i=1}^n Poss_i(a_i, s)$. However, the converse does not necessarily hold: two simple actions may each be possible, their preconditions may be jointly consistent, yet intuitively they should not be concurrently possible. This is the precondition interaction problem as discussed in [Reiter, 2001].

To model turn-based games, we introduce an action $wait$, and $n$ fluents $turn_i(s)$, meaning that it is agent $i$’s turn to make a move. We have $Poss_i(wait, s) = \neg turn_i(s)$.

To model agents’ observability about action profiles, we introduce a special predicate $A'(i, j, a', a, s)$, meaning that when agent $j$ performs action $a$ in situation $s$, agent $i$ considers it possible that agent $j$ does action $a'$. In general, we let $A(i, d, d, s) = \bigwedge_{i=1}^n A'(i, j, d', s)$. Let $f_G$ be a group strategy of coalition $G$. The abbreviation $s \subseteq f_G s'$ is used to represent the formula

\[ s \subseteq s' \wedge \forall s'' \forall d \exists d \subseteq do(d, s'') \subseteq s' \supset \bigwedge_{i \in G} d_i = f_i(s''). \]

Intuitively, this means that $s$ is a subhistory of $s'$, and on the way from $s'$ to $s'$, each agent $i$ in $G$ performs actions according to strategy $f_i$. Further, we introduce the abbreviation:

\[ s \subseteq f_A s' \iff s \subseteq f_A s' \wedge s \subseteq s'. \]

Let $\phi(f_A, s)$ be a situation calculus formula. We introduce the following abbreviations, where we use $f_A(s)$ to represent the action profile at situation $s$, i.e., $(f_1(s), \ldots, f_n(s))$.

- Next $\phi$: $\Box \phi \iff \phi(f_A, do(f_A(s), s))$;
- Eventually $\phi$: $\Diamond \phi \iff \exists s'. s \subseteq f_A s' \wedge \phi(f_A, s')$.

We say that a strategy $f$ is an executable strategy of agent $i$, if in any situation, $i$ knows the action required by $f$ and its executability. Formally, we have:

\[ EX(i, f) \equiv \forall s \exists a.TBel(i, f(now) = a \wedge Poss_i(a, now), s). \]

For a coalition $G$, we let $EX(G, f_G) = \bigwedge_{i \in G} EX(i, f_i)$. In fact, the notion of executable strategy coincides with that of uniform strategy in the literature, see [Jamroga and van der Hoek, 2004]: a strategy $f$ is uniform if for any situations $s$ and $s'$ indistinguishable for agent $i$, the values of $f$ at $s$ and $s'$ are the same. Formally, we have:

\[ \forall s, s'. B(i, s', s') \supset f(s) = f(s') \wedge Poss_i(f(s), s) \wedge Poss_i(f(s'), s'). \]

In the following, we illustrate the use of our situation calculus language with a variant of Levesque’s Squirrels World.

**Example 1** Squirrels and acorns live in a plane, and there is a fixed set of squirrels acting simultaneously at any time. Each squirrel and acorn is located at a cell $(p, q)$, where $p, q \in \mathbb{Z}$, the set of integers, and each cell can contain any number of acorns and squirrels.

Each squirrel can do actions below: pick up an acorn if he is located at the same cell as this acorn and does not hold any acorn in the current situation; drop an acorn that he is holding; move up, down, right and left a cell; also stay at the current location and do nothing (nil). A squirrel can observe the action of another squirrel within a distance of one, but if the action is a sensing action, the result is not observable.

There are only two squirrels 1 and 2. As shown in Figure 1, initially, squirrel 1 is located at the cell $(-1, -1)$ and 2 is located at the cell $(1, 1)$, and any squirrel knows the location of the other one; there is only one acorn in each of the cell $(-1, 1)$ and $(1, -1)$, and in other cells there are no acorns.

We use four ordinary fluents: in situation $s$, squirrel $i$ is holding an acorn $(hold(i, s))$; squirrel $i$ is at cell $(p, q)$ $(cell(i, p, q, s))$; there are $n$ acorns at cell $(p, q)$ $(acorn(p, q, n, s))$, the situation $s$ will be achieved after $m$ steps from the initial situation $S_0(step(s) = m)$.

For illustration purpose, we only present some axioms of the BAT of this game, denoted by $D_{sq}$:

\[ Poss_i(up, s) \equiv T, \quad i = 1, 2; \]
\[ Poss_i(pick, s) \equiv \neg hold(i, s) \wedge 3p, q, n.cell(i, p, q, s) \wedge acorn(p, q, n, s) \wedge n > 0, \quad i = 1, 2; \]
\[ Poss_i(pick, pick), s) \equiv \bigwedge_{i=1}^2 Poss_i(pick, s) \wedge \neg \exists p, q, 3n.cell(i, p, q, s) \wedge acorn(p, q, 1, s); \]
\[ hold(i, do(d, s)) \equiv d_i = pick \lor hold(i, s) \land d_i \neq drop \]

![Figure 1: The Squirrels World](image-url)
\[ A'(i, j, a, right, s) \equiv \exists p, p', q, q'.cell(i, p, q, s) \land \]
\[ cell(j, p', q', s) \land (|p-p'| + |q-q'| \leq 1 \land a = right) \land (|p-p'| + |q-q'| \geq 1 \land a = nil); \]
\[ TBel(2, cell(1, -1, -1), S_0) \land TBel(1, cell(2, 1, 1), S_0); \]
\[ \forall p, q.acorn(p, q, n, S_0) \land n > 0 \equiv \]
\[ p = -1 \land q = 1 \lor p = 1 \land q = -1; \]
\[ acorn(-1, 1, 1, S_0) \land acorn(1, -1, 1, S_0). \]

Finally, we show that our situation calculus can be used to reason about games described by GDL-II [2014]. For the purpose of General Game Playing, GDL (Game Description Language) has been developed as a high-level language for the specification of complete information games. GDL is based on the standard syntax and semantics of logic programming, characterized by a number of special keywords. GDL-II is an extension of GDL for describing imperfect/incomplete information games. It has two extra keywords: sees, which specifies the information that each player gets, and random, which denotes a special player who chooses moves randomly.

Schiffel and Thielchers present a full embedding of GDL-II into a multi-agent epistemic situation calculus, and formally prove that this provides a sound and complete reasoning method for players’ knowledge about game states as well as about the knowledge of the other players. In fact, the situation calculus they use is essentially our extended situation calculus without the strategy sort. If we let \( A(i, d, d', s) \equiv \)
\[ d_i = d'_i \land \forall P.Sees(i, P, d, s) \equiv Sees(i, P, d', s), \]
which intuitively means that agent \( i \) cannot distinguish between two joint actions \( d \) and \( d' \) if their own actions are the same and the information \( i \) gets about \( d \) is the same as that about \( d' \), then their SSA for the knowledge fluent \( K \) coincides with the SSA for our \( B \) fluent. Therefore, using their embedding, we get that GDL-II can be embedded into the concurrent epistemic situation calculus.

4 Individual strategic ability

In this section, we show that in our situation calculus, we can distinguish between different kinds of individual strategic abilities, including that studied by [Lespérance et al., 2000].

As discussed in the literature, say [Jamroga and Agotnes, 2007], by considering whether the strategies of agent \( i \) or other agents are executable, and whether agent \( i \) knows which strategy to ensure his goal, there are different notions of strategic abilities of agent \( i \). We can formalize these different abilities in our situation calculus as follows. Let \( \phi(f_{Ag}, s) \) be a situation calculus formula which serves as the goal for agent \( i \). Note that by our notation, \( f_i \) represents a group strategy of agents other than \( i \).

1. \( Can_1(\phi, s) \equiv \exists f_i.Bel(i, \forall f_i.\phi(f_i, f_i, now), s); \)
2. \( Can_2(\phi, s) \equiv Bel(i, \forall f_i.\phi(f_i, f_i, now), s); \)
3. \( Can_3(\phi, s) \equiv \exists f_i.EX(i, f_i) \land Bel(i, \forall f_i.\phi(f_i, f_i, now), s); \)
4. \( Can_4(\phi, s) \equiv Bel(i, \exists f_i.EX(i, f_i) \land \forall f_i.\phi(g_i, g_i, now), s). \)

In (1), \( i \) knows which strategy to ensure \( \phi \). In (2), \( i \) knows there exists a strategy to ensure \( \phi \). But in both (1) and (2), the strategy of agent \( i \) may not be executable. In (3), agent \( i \) knows which strategy to ensure the goal \( \phi \) and this strategy is executable for him, but other agents’ strategies may not be executable; in this case, agent \( i \) considers the worst case. In (4), \( i \) knows there exists an executable strategy to ensure \( \phi \), but does not know which strategy.

Example 1 Cont’d. We first give an abbreviation meaning that the distance between agents 1 and 2 in \( s \) is not more than \( k \):
\[ dist(1, 2, s) \leq k \equiv 3p, q, p', q'.cell(1, p, q, s) \land cell(2, p', q', s) \land |p-p'| + |q-q'| \leq k. \]

Also, we let \( S_1 = do(right, left), S_0 \). Since in \( S_0 \), any agent cannot see the action of the other one, in \( S_1 \), \( i \) no longer knows the location of the other one.

1. Let \( \phi_1 = O\{dist(1, 2) \leq 4 \} \). Then \( D_{S_0} \models Can_3(\phi_1, S_1). \)

This is because when agent 1 moves up, no matter what action agent 2 does, agent 1 believes that their distance is no more than 4.

2. Let \( \phi_2 = O\{p, q, p', q'.cell(2, p, q, s) \land \}
\[ [q < 3 \land -(p = 2 \land q = 2) \lor |p-p'| \leq 2 \land |q-q'| \leq 2]. \]

Then \( D_{S_0} \models Can_1(\phi_2, S_2) \land Can_3(\phi_2, S_1) \land Can_4(\phi_2, S_1). \) In \( S_1 \), after agent 1 does the same action as agent 2 does in \( S_0 \), he believes that \( \phi_2 \) holds. Thus we have \( Can_1(\phi_2, S_1) \). However, since agent 1 cannot observe agent 2’s action in \( S_0 \), this strategy is not executable. In fact, we cannot find an executable strategy of agent 1 which ensures \( \phi_2 \), hence \( Can_3(\phi_2, S_1) \) does not hold. On the other hand, for each possible action of agent 2 in \( S_0 \), when agent 1 does the same action in \( S_1 \), \( \phi_2 \) holds; thus we have \( Can_4(\phi_2, S_1) \).

Now we show that we can represent in our framework the notion of ability \( Can(\varphi, s) \) defined by Lespérance et al. (2000) in the single-agent case, where \( \varphi(s) \) is a formula about situation \( s \):

- \( \text{OnPath}(f, s, s') \equiv s \leq s' \land \forall a.s' \subseteq s' \supset f(s') = a; \)

This is the same as our \( s \leq f.s' \).

- \( \text{CanGet}(\varphi, f, s) \equiv \exists s'.(\text{OnPath}(f, s, s') \land \text{Bel}(\varphi, s') \land \forall s'. s \subseteq s' \subseteq s' \supset \exists a.\text{Bel}(f(now) = a, s')); \)

Here the second line requires that \( g \) be uniform on the way from \( s \) to \( s' \).

- \( \text{Can}(\varphi, s) \equiv \exists f.\text{Bel}(\text{CanGet}(\varphi, f, now), s). \)

Recall that we use \( \Sigma \) to denote the set of foundational axioms for our extended situation calculus. Then we get

\[ \Sigma \models Can(\varphi, s) \equiv Can_3(\Diamond\text{Bel}(\varphi, now), s). \]

5 A strategy programming language: SGolog

In this section, we propose a strategy programming language SGolog for specifying collective strategies of coalitions.

We first consider a single-step fragment (SSF) of Golog, which is used to specify an agent’s possible choice to perform in one step. A situation-suppressed formula \( \varphi \) is a situation calculus formula with all situation arguments suppressed, and \( \varphi[s] \) denotes the formula obtained from \( \varphi \) by taking \( s \) as the situation arguments of all fluent mentions in \( \varphi \).
Definition 1 SSF programs are defined inductively as follows: $\delta := (\varphi?; \alpha) \mid (\delta_1 \mid \delta_2) \mid (\pi.a.(x))$, where $\varphi$ is a situation-suppressed formula, and $\alpha$ is an action term.

The formal semantics of SSF programs is defined by an abbreviation $\text{Does}(\delta, a, s)$, which intuitively means action $\alpha$ forms a legal execution of $\delta$ in situation $s$.

Definition 2 $\text{Does}(\delta, a, s)$ is defined inductively as:

- $\text{Does}(\varphi?; a, s, a) \equiv \varphi[s] \land a = \alpha$;
- $\text{Does}(\delta_1 \mid \delta_2, a, s) \equiv \text{Does}(\delta_1, a, s) \lor \text{Does}(\delta_2, a, s)$;
- $\text{Does}(\pi.a.(x), a, s) \equiv \exists x.\text{Does}(\delta(x), a, s)$.

We use $\top$ (resp. $\bot$) to represent $\text{true}$ (resp. $\text{false}$). For convenience, we let * be an extra SSF program, which intuitively means taking an arbitrary action, and define $\text{Does}(*, s, a) \equiv \top$. Next, we define SGolog programs, which can be used to represent the collective behavior/strategy of a coalition or the protocol of a multi-agent game. A primitive SGolog program is an $n$-ary vector of SSF programs. We use $\pi$ to denote the $n$-ary vector of $*s$. For a primitive program $\theta$, we use $\theta_i$ to denote its $i$th component.

Definition 3 SGolog programs are defined as follows:

$\rho := \emptyset \mid (\varphi?; \rho_1; \rho_2) \mid (\pi.a.(x)) \mid \rho^* \mid \text{while } \varphi \text{ do } \rho \mid \rho_1 \vee \rho_2 \vee \ldots \vee \rho_m \mid \rho_1 \land \rho_2 \land \ldots \land \rho_m$,

where $\theta$ is a primitive program, and $\varphi$ is a situation-suppressed formula.

A conditional [if $\varphi$ then $\rho_1$ else $\rho_2$] is defined as abbreviation for the program $[\varphi?; \rho_1; \rho_2]$. The prioritized disjunction and conjunction operators $\vee$ and $\land$ are inspired by the work of Zhang and Thielscher (2015). Intuitively, $\rho_1 \vee \rho_2 \vee \ldots \vee \rho_m$ means that if the program $\rho_1$ can be executed in situation $s$, then only $\rho_1$ is executed and the other programs are discarded, else if $\rho_2$ can be executed, then execute $\rho_2$, $\ldots$, $\rho_i$ has higher priority than $\rho_{i+1}$. The program $\rho_1 \land \rho_2 \land \ldots \land \rho_m$ means that if all of $\rho_1$, $\ldots$, and $\rho_m$ can be applied together in $s$, then all of them are applied, else we discard $\rho_m$ and consider whether the remaining $\rho_1$s can be applied together.

The formal semantics of SGolog programs is defined by two predicates: $\text{Trans}(\rho, s, \rho', s')$, which holds if executing one step of program $\rho$ in situation $s$ may lead to situation $s'$ with $\rho'$ remaining to be executed; and $\text{Final}(\rho, s)$, which holds if program $\rho$ may legally terminate in situation $s$. The two predicates are inductively defined as follows.

1. $\text{Trans}(\emptyset, s, \rho', s') \equiv \exists d.\text{Poss}(d, s) \land s' = do(d, s) \land \\land_{i=1}^n \text{Does}(\theta_i, d_i, s) \land \rho' = \top$;
2. $\text{Trans}(\varphi?, s, \rho', s') \equiv \top$;
3. $\text{Trans}(\rho_1; \rho_2, s, \rho', s') \equiv (\exists \rho_1'.\text{Trans}(\rho_1, s, \rho_1', s') \land \rho' = \rho_1'; \rho_2) \lor \text{Final}(\rho, s) \land \text{Trans}(\rho_2, s, \rho', s')$;
4. $\text{Trans}(\pi.a.(x), s, \rho', s') \equiv \text{Trans}(\pi.a.(x), s, \rho, s') \lor \text{Trans}(\rho_2, s, \rho', s')$;
5. $\text{Trans}(\varphi?, s, \rho', s') \equiv \exists x.\text{Trans}(\varphi, s, \rho, s')$;
6. $\text{Trans}(\rho^*, s, \rho', s') \equiv \exists \rho'.\text{Trans}(\rho, s, \rho', s') \land \rho' = \rho^*; \rho^*$;
7. $\text{Trans}(\text{while } \varphi \text{ do } \rho, s, \rho', s') \equiv \exists \rho'.\text{Trans}(\varphi(s), s, \rho, s') \land \rho' = \rho''; (\text{while } \varphi \text{ do } \rho)$;
8. $\text{Trans}(\rho_1 \lor \ldots \lor \rho_m, s, \rho', s') \equiv \text{Trans}(\rho_1, s, \rho', s') \lor \ldots \lor \text{Trans}(\rho_m, s, \rho', s')$;
9. $\text{Trans}(\rho_1 \land \ldots \land \rho_m, s, \rho', s') \equiv (\bigwedge_{k=1}^\infty \exists \rho_k'.\text{Trans}(\rho_k, s, \rho_k', s') \land \rho' = \rho_1 \land \ldots \land \rho_m \lor (\bigwedge_{k=1}^\infty \exists \rho_k'.\text{Trans}(\rho_k, s, \rho_k', s') \land \rho' = \rho_1 \land \ldots \land \rho_m - 1, s', s'))$;

10. $\text{Final}(\emptyset, s) \equiv \top$;
11. $\text{Final}(\varphi?, s) \equiv \varphi[s]$;
12. $\text{Final}(\rho_1; \rho_2, s) \equiv \text{Final}(\rho_1, s) \land \text{Final}(\rho_2, s)$;
13. $\text{Final}(\rho_1 \lor \rho_2, s) \equiv \text{Final}(\rho_1, s) \lor \text{Final}(\rho_2, s)$;
14. $\text{Final}(\rho, s) \equiv \top$;
15. $\text{Final}(\rho^*, s) \equiv \top$;
16. $\text{Final}(\text{while } \varphi \text{ do } \rho, s, \rho', s') \equiv \varphi[s] \land \text{Final}(\rho', s')$;
17. $\text{Final}(\rho^*, s) \equiv \top$;

Example 1 Cont.d.

- Let $\rho_1 = (\langle \text{up}, *; \rangle; \langle \text{pick}, *; \rangle) \lor (\langle \text{right}, *; \rangle; \langle \text{pick}, *; \rangle)$, then $\rho_1$ is a strategy for agent 1 in situation $S_0$ such that he prefers the acorn in position $(-1, 1)$.
- Let $\rho_2 = (\langle \text{left}, *; \rangle; \langle \text{up}, *; \rangle) \lor (\langle \text{right}, *; \rangle; \langle \text{down}, *; \rangle)$. Since this is a prioritized conjunction, when $\rho_2$ is performed in $S_0$, agent 1 should move up twice.
- Let $\rho_3 = (\langle \text{up}, *; \rangle; \langle \text{right}, \text{left}, *; \rangle)$. Then $\rho_3$ is a group strategy for the two agents in situation $S_0$ to achieve the goal of meeting at $(0, 0)$.

We now formally define the notion that a group strategy satisfies an SGolog program. First, we introduce an abbreviation $\text{Sat}(f_G, \rho, s)$, which intuitively means that group strategy $f_G$ complies with program $\rho$ in one step from $s$. It is inductively defined as follows:

1. $\text{Sat}(f_G, \emptyset, s) \equiv \bigwedge_{e \in G} \text{Does}(\theta_1, f_1(s), s)$;
2. $\text{Sat}(f_G, \varphi?, s) \equiv \top$;
3. $\text{Sat}(f_G, \rho_1; \rho_2, s) \equiv \text{Sat}(f_G, \rho_1, s) \land \text{Sat}(f_G, \rho_2, s)$;
4. $\text{Sat}(f_G, \rho_1 \lor \rho_2, s) \equiv \text{Sat}(f_G, \rho_1, s) \lor \text{Sat}(f_G, \rho_2, s)$;
5. $\text{Sat}(f_G, \rho^*, s) \equiv \text{Sat}(f_G, \rho, s)$;
6. $\text{Sat}(f_G, \text{while } \varphi \text{ do } \rho, s) \equiv \varphi[s] \land \text{Sat}(f_G, \rho, s)$;
7. $\text{Sat}(f_G, \text{while } \varphi \text{ do } \rho, s) \equiv \varphi[s] \land \text{Sat}(f_G, \rho, s)$;
8. $\text{Sat}(f_G, \rho_1 \lor \ldots \lor \rho_m, s) \equiv (\exists \rho'.\text{Sat}(\varphi(s), s, \rho', s') \land \rho' = \rho''; (\text{while } \varphi \text{ do } \rho))$;
9. $\text{Sat}(f_G, \rho_1 \land \ldots \land \rho_m, s) \equiv (\exists \rho'.\bigwedge_{k=1}^\infty \exists \rho_k'.\text{Sat}(\rho_k, s, \rho_k', s') \land s' = do(d, s) \land \\d_G = f_G(s) \lor (\exists s'.\bigwedge_{k=1}^\infty \exists \rho_k'.\text{Sat}(\rho_k, s, \rho_k', s') \land \text{Sat}(f_G, \rho_1 \land \ldots \land \rho_m - 1, s'))$.
As for \( Sat_1(f_G, \rho, s) \), we only consider one step from situation \( s \). To represent a collective strategy of a coalition by an SGolog program, we specify how a group strategy completely complies with this program from a situation.

**Definition 4** \( Sat(f_G, \rho, s) \) is defined as:
\[
\forall s', s \leq f_G \; s' \land s' \neq s \rightarrow \exists \rho', s'. Trans^+(\rho, s, \rho', s') \land (s' = s' \lor Final(\rho', s') \land s' \subset s').
\]

Here \( Trans^+ \) is the transitive closure of \( Trans \). Intuitively, \( Sat(f_G, \rho, s) \) means that any situation \( s' \) reachable from \( s \) following \( f_G \) is either a situation resulting from the execution of \( \rho \), or an situation following a final situation of \( \rho \).

Obviously, we can prove that \( \Sigma \models Sat(f_G, \rho, s) \Rightarrow Sat_1(f_G, \rho, s) \), that is, if \( f_G \) complies with \( \rho \) from \( s \) on, then it must comply with \( \rho \) in one step. In fact, any non-final configuration \((\rho', s')\) which comes from configuration \((\rho, s)\) according to \( f_G \) will conform with the remaining \( \rho' \) in \( s' \) at the next situation. That is,

**Proposition 1** \( \Sigma \models Sat(f_G, \rho, s) \Rightarrow \forall s', s' \leq f_G \; s' \land Trans^+(\rho, s, \rho', s') \land \neg Final(\rho', s') \land s' \subset s' \) \( \supset \) \( Sat_1(f_G, \rho, s') \).

Therefore, we can say that program \( \rho \) completely determines a collective strategy \( f_G \) from a situation \( s \), if we have \( Sat(f_G, \rho, s) \land \forall f_G \supset s'. Sat(f_G, \rho, s) \land s \leq f_G \; s' \supset f_G(s') = f_G(s') \). According to the above results, we can also have:

**Proposition 2**
- \( \Sigma \models Sat(f_G, \varphi?; \rho, s) \equiv \varphi[s] \land Sat(f_G, \rho, s) \).
- \( \Sigma \models [\forall i \in G Poss_i(f_i(s), s)] \supset Sat(f_G, \rho, s) \), where \( \rho \) while \( T \) do \( \bar{s} \).

6 Joint ability under an SGolog program

In this section, we adapt the approach of [Ghaderi et al., 2007] to formalize joint ability of coalitions under commitments to strategy programs.

We first define the preferred strategies of agent \( i \) under an SGolog program \( \rho \) in situation \( s \) by the approach of iterated elimination of dominated strategies. Given an agent \( i \), a group \( G \), a strategy \( f_i \), a program \( \rho \), a situation \( s \), and a goal \( \phi \),

\[
Pref(i, G, f_i, \rho, \phi, s) \equiv \forall n. Keep(i, G, n, f_i, \rho, \phi, s).
\]

So agent \( i \) prefers strategy \( f_i \) for goal \( \phi \) under the program \( \rho \) if it is kept for all levels \( n \), where \( n \) is a natural number.

There are two cases for the definition of \( Keep \):

1. For any \( n \),
\[
Keep(i, G, n, f_i, \rho, \phi, s) = EX(f_i, s) \land Sat(f_i, s).
\]

For so agents outside the coalition, we keep those executable strategies satisfying \( \rho \).

2. \( i \in G \) if \( Keep \) is inductively defined as follows:
\[
Keep(i, G, n + 1, f_i, \rho, \phi, s) = Keep(i, G, n, f_i, \rho, \phi, s) \land \neg \exists f_i'. Keep(i, G, n, f_i', \rho, \phi, s) \land GTE(i, G, n, f_i, \rho, \phi, s) \land GTE(i, G, n, f_i, \rho, \phi, s).
\]

For agent \( i \), the strategies kept at level \( n + 1 \) are those kept at level \( n \) for which there is not a better one at level \( n \).

**Theorem 2** 1. Let \( \rho = while T do \bar{s}, then \)
\[
\Sigma \models JCan(\phi, s) \equiv EX(\bar{f}_i, EX(G, f_G, s)).
\]

In fact, those specifications of abilities defined by Ghaderi et al. (2007) and in ATL can be viewed as joint abilities under some special SGolog programs. First, we formalize them in our framework with minor differences:

- \( JCan(\phi, s) \equiv \forall f_G \exists f_i. EX(AG, f_G, s) \Rightarrow \phi(f_G, s) \)
- \( \exists \forall f_G \exists f_i. EX(AG, f_G, s) \Rightarrow \phi(f_G, s) \).

- \( \exists \forall f_G \exists f_i. EX(AG, f_G, s) \Rightarrow \phi(f_G, s) \).

- \( \exists \forall f_G \exists f_i. EX(AG, f_G, s) \Rightarrow \phi(f_G, s) \).

Intuitively, the above two cases are two extremes of SGolog programs. For the former, no agent has any knowledge about the other agents’ strategies. For the latter, the group strategy \( f_G \) is the common knowledge of group \( G \).

From our definition of joint ability, we can see that SGolog programs serve as the common knowledge of all agents. Thus we can use SGolog programs to specify rough collective strategies, protocols, conventions, or social laws.

**Example 1** Cont’d. Suppose the goal of the two squirrels is that each squirrel will hold an acorn in at most 4 steps...
from $S_0$. This can be represented with $\phi_{sq}: \Diamond (\text{step} \leq 4 \land \text{hold}(1) \land \text{hold}(2))$. We now consider whether the two squirrels have joint abilities to achieve the goal under different SLogol programs:

- Let $\rho_1 = \text{while } \top \text{ do } (\ast, \ast)$. Then $D_{sq} \not\models SA(\{1,2\}, \rho_1, \phi_{sq}, S_0)$. Under this program, for squirrel 1 (resp. 2), any of his preferred strategies should move up (resp. down) or right (resp. left) in the first step. However, if squirrel 1 moves up and squirrel 2 moves left in the first step, no matter how they will act in the later steps, their goal cannot be achieved.

- Let $\theta_1 = (\text{right}, \text{left}), \theta_2 = (\text{up}, \text{down})$, and $\rho_2$ be $\theta_1$ or $\theta_2$. Then $D_{sq} \models SA(\{1,2\}, \rho_2, \phi_{sq}, S_0)$. We explain using the example of $\theta_1$. Under this program, the preferred strategy for squirrel 1 must be $\text{right}; \text{right}; \text{pick}$; and for squirrel 2 it must be $\text{left}; \text{left}; \text{pick}$. Therefore, any combination of the preferred strategies of the two squirrels can ensure the goal.

- Let $\rho_3 = \theta_1 \lor \theta_2$. Then $D_{sq} \models SA(\{1,2\}, \rho_3, \phi_{sq}, S_0)$. Since $\theta_1$ can be executed in $S_0$, it will be executed due to higher priority.

- Let $\rho_4 = (\ast, \text{down}); (\ast, \text{down})$. Then $D_{sq} \models SA(\{1,2\}, \rho_4, \phi_{sq}, S_0) \land \neg SA_0(\{1,2\}, \rho, \phi_{sq}, S_0)$. For goal $\phi_{sq}$, initially, agent 1 keeps all those executable strategies, and agent 2 keeps those executable strategies whose first two steps both are down. When agent 1 moves right in the first step, the two agents cannot achieve their goal. However, at level 1, agent 1 should consider those kept strategies of agent 2 at level 0, then be only keeps those strategies whose first 3 steps are up, up, pick. Then any combination of the preferred strategies of the two agents can achieve the goal.

**Example 2** We now consider a traffic example. As shown in Figure 2, there are two cars (1 and 2) driving in opposite directions on a road in which there are four lanes numbered as 1, 2, 3, 4. There is only one action move($k$) which is always possible, here $k$ is a lane. For simplicity, we assume the velocities of the two cars are the same, that is, when a car executes move($k$), then it moves one unit forward. For $i = 1, 2$, there is a fluent $loc_i(m, k, s)$, which means car $i$ is on lane $k$ with coordinate $m$ in situation $s$. Two cars collide when they are at the same location. In this game, we assume that neither car can see the other car (for instance, when two cars are driving in the night with broken lights).

We have the following axioms:

$$\text{Poss}_i(\text{move}(k), s) \equiv \top, i = 1, 2;$$

$$\text{loc}_1(m, k, \text{do}(d, s)) \equiv \exists k'. \text{loc}_1(m - 1, k', s) \land d_1 = \text{move}(k);$$

$$\text{loc}_2(m, k, \text{do}(d, s)) \equiv \exists k'. \text{loc}_2(m + 1, k', s) \land d_2 = \text{move}(k);$$

$$\exists k. \text{loc}_1(0, k, S_0) \land \exists k'. \text{loc}_2(n, k', S_0).$$

We denote the BAT of this game as $D_{tr}$, and the goal formula is $\varphi = \neg \Diamond m, k. \text{loc}_1(m, k) \land \text{loc}_2(m, k)$, meaning that two cars will never collide.

Let $\rho = \text{while } \top \text{ do } (\ast, \ast)$. Under $\rho$, for any agent $i$, any strategy is a preferred strategy. However, if both agents choose to always perform move(1), they must collide. Thus $D_{tr} \not\models SA(\{1,2\}, \rho, \varphi, S_0)$. However, consider $\rho' = \text{while } \top \text{ do } (\text{move}(1), \text{move}(2), \text{move}(3) \mid \text{move}(4))$. Intuitively, $\rho'$ requires that both agents should always drive on the right. It is easy to see that under $\rho'$, the two cars have joint ability to avoid collision. Thus $D_{tr} \models SA(\{1,2\}, \rho', \varphi, S_0)$.

7 Conclusions

In this paper, by a simple extension of a variant of multi-agent epistemic situation calculus with a strategy sort, we have developed a general framework for strategy representation and reasoning for incomplete information concurrent games. The framework can be used to compactly represent the structure of such games, distinguish between different kinds of individual strategic abilities, specify collective strategies of coalitions, and reason about joint abilities of coalitions under commitments to collective strategy specifications. Both strategic abilities in ATL and joint abilities of Ghaderi et al. can be considered as joint abilities under special strategy specifications in our framework.

In our current framework, a strategy of an agent is a function from situations to actions, thus we have made the assumption that all agents have perfect recall. But in reality, agents may have limited memory, and are even memory-free. In the future, we would like to extend our framework to accommodate bounded-memory agents. Another future work is to identify decidable fragments of our framework based on existing work, e.g., [De Giacomo et al., 2013].

Acknowledgments

We thank the anonymous reviewers for helpful comments. This work received support from the Natural Science Foundation of China under Grant Nos. 61572535 and 61463044.

References


