ECAI 2016
G.A. Kaminka et al. (Eds.)

doi:10.3233/978-1-61499-672-9-408

Situation Calculus Game Structures and GDL

Giuseppe De Giacomo[1] and **Yves Lespérance**[2] and **Adrian R. Pearce**[3]

Abstract. We present a situation calculus-based account of multi-players synchronous games in the style of general game playing. Such games can be represented as action theories of a special form, *situation calculus synchronous game structures (SCSGSs)*, in which we have a single action $tick$ whose effects depend on the combination of *moves* selected by the players. Then one can express properties of the game, e.g., winning conditions, playability, weak and strong winnability, etc. in a first-order alternating-time μ-calculus. We discuss verification in this framework considering computational effectiveness. We also show that SCSGSs can be considered as a first-order variant of the Game Description Language (GDL) that supports infinite domains and possibly non-terminating games. We do so by giving a translation of GDL specifications into SCSGSs and showing its correctness. Finally, we show how a player's possible moves can be specified in a Golog-like programming language.

1 Introduction

Many types of problems can be viewed as games, where one or more agents interact to ensure that certain objectives hold no matter how the environment and other agents behave, e.g., contingent planning, service orchestration, controller synthesis, etc. Moreover, general game playing [13], where artificial agents compete in games that are not known in advance, is an important emerging AI testbed. Logics for reasoning about game settings, e.g., [35, 16, 24], has been an active area, with Alternating-Time Temporal Logic (ATL) [1] a popular choice. Model checking techniques have been used to verify properties of games specified in ATL and to synthesize strategies that agents can use to force temporal properties to hold [21]. However, such logics are usually propositional or limited to finite domains. Moreover, the game settings are usually specified using low-level automata-like languages. One exception is the Game Description Language (GDL) [13, 22] developed for the general game playing competition, which is based on logic programming, and allows for a quite high level representation of games. Typically, however, GDL is intended to represent games with finite domains in a declarative way, with a semantics based on "negation as failure" [13, 22, 27].

Within the situation calculus (SitCalc) [23, 26], a well known formalism for reasoning about action based on first-order logic (FOL) (with a second-order axiom to specify the domain of situations), [11] proposes an expressive logical framework for specifying and solving game-like problems. Game settings are specified as a special kind of SitCalc action theory. It is assumed that in any given state, only one agent may act next, and thus the approach is concerned with *turn-taking* games. Complex temporal properties of games can be expressed in a first-order (FO) variant of alternating-time μ-calculus. Methods for verification and synthesis based on fixpoint approximation and regression are developed.

In this paper, inspired by [11], we develop a SitCalc-based specification and verification framework, which deals with multi-players *synchronous games*, and is similar in spirit to GDL. Games are represented as action theories of a special form called *situation calculus synchronous game structures (SCSGSs),* where we have a single action $tick$ whose effects depend on the combination of *moves* selected by the players (see Sec. 3). A FO variant of alternating-time μ-calculus is used to specify and verify properties of the game (see Sec. 5), including winning conditions, playability, weak and strong winnability, etc.

The paper's main contributions are:

1. We develop a truly first-order framework that can be used to specify games/systems that involve infinite domains and infinite sets of states.
2. Games can be specified at a high level, using SitCalc action theories [26].
3. SCSGSs amounts to a variant of GDL where states are represented by first-order theories: we give a translation of GDL specifications into SCSGSs and show its soundness and completeness (see Sec. 4).
4. Reasoning techniques developed for the SitCalc can be used to verify properties of games, which can help in analyzing them and developing better players. These includes sound but incomplete techniques that apply to the general setting [11, 17], and techniques that are sound and complete for the decidable "bounded fluent extension" setting [8] (see Sec. 5).
5. Also agent moves can be specified procedurally in a variant of the SitCalc-based programming language Golog [19] (see Sec. 6).
6. Recent verification techniques developed for Golog and ConGolog programs, e.g., [10], can be applied (see again Sec. 6).

Like the original GDL formalism, our account assumes that agents have full observability of the state and all past moves. Handling partial observability, as in GDL-II [33, 30], is left for future work.

2 Preliminaries

The *situation calculus* (SitCalc) is a sorted predicate logic language for representing and reasoning about dynamically changing worlds [23, 26]. It includes three sorts, *Actions*, *Situations* and *Objects*. All changes to the world are the result of *actions*, which are terms in the logic. A possible world history is represented by a term called a *situation*. The constant S_0 is used to denote the initial situation where no actions have yet been done. Sequences of actions are built using the function symbol do, where $do(a, s)$ denotes the successor situation resulting from performing action a in situation s. Predicates and

[1] Dip. di Ingegneria Informatica, Automatica e Gestionale, Sapienza – Università di Roma, Rome, Italy, email: degiacomo@dis.uniroma1.it
[2] Dept. of Electrical Engineering and Computer Science, York University, Toronto, ON, Canada, email: lesperan@cse.yorku.ca
[3] Dept. of Computer Science and Software Engineering, University of Melbourne, Victoria, Australia, email: adrianrp@unimelb.edu.au

functions whose value varies from situation to situation are called *fluents*, and are denoted by symbols taking a situation term as their last argument (e.g., $Holding(x, s)$). Actions and fluents (except for the last argument) can only take arguments of sort *Objects*. Notice that we allow the object domain to be infinite. Within this language, we can formulate action theories that describe how the world changes as the result of actions. Here, we concentrate on *basic action theories* as proposed in [26]. A *basic action theory* $\mathcal{D}$ is the union of the following disjoint sets: the foundational, domain independent, axioms of the SitCalc (Σ); unique name axioms for actions; precondition axioms stating when actions can be legally performed ($\mathcal{D}_{poss}$); successor state axioms describing how fluents change between situations ($\mathcal{D}_{ssa}$); and axioms describing the initial configuration of the world ($\mathcal{D}_{S_0}$). A special predicate $Poss(a, s)$ is used to state that action a is executable in situation s; precondition axioms in $\mathcal{D}_{poss}$ characterize this predicate. We say that a situation s (corresponding to a sequence of actions) is executable, written $Executable(s)$, if every action performed in reaching s is possible in the situation it occurred [26]. In turn, successor state axioms encode the causal laws of the domain; they take the place of the so-called effect axioms and provide a solution to the frame problem.

3 Synchronous Game Structures

We focus on games where there are n players/agents each of whom chooses a move at every time step. All such moves are executed *synchronously* and determine the next state of the game. At each time step, the state of the game is fully observable by all agents, as are all past moves of every agent. This is in agreement with the assumptions built into GDL [13, 22]. To represent such multi-player synchronous games, we define a special class of basic action theories, called *situation calculus synchronous game structures (SCSGSs)*, which are defined as follows.

Agents A SCSGS involves a finite set of n agents, and we introduce a subsort *Agents* of *Objects* which includes these finitely many agents $Ag_1, \ldots, Ag_n$, each denoted by a constant, and for which unique names $Ag_i \neq Ag_j$ for $i \neq j$ and domain closure $Agent(x) \equiv x = Ag_1 \vee \cdots \vee x = Ag_n$ hold.

Moves. We also introduce a second subsort *Moves* of *Objects*, representing the possible moves of the agents. These come in finitely many types, represented by function symbols $M_i(\vec{x})$, which are parametrized by objects $\vec{x}$ and we have $Move(m) \equiv \bigvee_i \exists \vec{x}.m = M_i(\vec{x})$. Given that the parameters range over *Objects*, each agent may have an infinite number of possible moves at each time step. We have unique name and domain closure axioms (parametrized by objects) for these functions $M_i(\vec{x}) \neq M_j(\vec{y})$ for $i \neq j$, and $M_i(\vec{x}) = M_i(\vec{y}) \supset \vec{x} = \vec{y}$.

Actions. In SCSGSs, there is only *one action type*, $tick(m_1, \ldots, m_n)$, which represents the execution of a joint move by all the agents at a given time step. The action $tick$ has exactly n parameters, $m_1, \ldots, m_n$, one per agent, which are of sort *Moves* and corresponds to the simultaneous choice of the move to perform by the n different agents.

Legal moves. A key component of a SCSGS is a characterization of the *legal* moves available to each agent in a given situation. This is specified formally using a special predicate $LegalM$, which is defined by statements of the following form (one for each agent Ag_i and move type M_i):

$$LegalM(Ag_i, M_i(\vec{x}), s) \doteq \Phi_{Ag_i,M_i}(\vec{x}, s)$$

meaning that agent Ag_i can legally perform move $M_i(\vec{x})$ in situation s if and only if $\Phi_{Ag_i,M_i}(\vec{x}, s)$ holds. Technically $LegalM$ is an abbreviation for $\Phi_{Ag_i,M_i}(\vec{x}, s)$, which is a uniform formula (i.e., a formula that only refers to a single situation s).

Precondition axioms. The precondition axiom for the action $tick$ is fixed and specified in terms of $LegalM$ as follows:

$$Poss(tick(m_1, \ldots, m_n), s) \equiv \bigwedge_{i=1,\ldots,n} LegalM(Ag_i, m_i, s)$$

This states that action $tick(m_1, \ldots, m_n)$, denoting the joint move of all agents, can be performed if and only if each selected move m_i is a legal move for agent Ag_i in situation s. Since we only have one action type $tick$, this is the only precondition axiom in $\mathcal{D}_{poss}$.

Successor state axioms. We have *successor state axioms* $\mathcal{D}_{ssa}$, specifying the effects and frame conditions of the joint moves $tick(m_1, \ldots, m_n)$ on the fluents. Such axioms, as usual in basic action theories, are domain specific, and characterize the actual game under consideration. Within such axioms, the agent moves, which occur as parameters of $tick$, determine how fluents change as the result of joint moves.[4]

Initial situation description. Finally, the initial state of the game is axiomatized in the *initial situation description* $\mathcal{D}_0$ as usual, in a domain specific way.

Example 1 *Consider the following example drawn from [29]. There are two guard agents, Ag_1 and Ag_2, that cooperatively try to catch a third agent, Ag_3, who is trying to escape, in a 5×5 grid world. Ag_3 is initially at location $(5, 5)$ and can escape after reaching any of the other corners. Initially, Ag_1 is at location $(1, 1)$ and Ag_2 at $(1, 5)$. At each time step, the agents can all move synchronously to an adjacent square. Ag_3 is caught and loses the game if he ends up on the same square as one of the guards or if he crosses path with one of them in a simultaneous move. We can specify this game as follows. We have 3 possible moves, with the following definitions:*

$$\begin{aligned}
&LegalM(ag, move(d), s) \doteq \exists u, v, x, y. \neg Terminal(s) \wedge \\
&\quad At(ag, u, v, s) \wedge Adj(u, v, d, x, y) \\
&LegalM(ag, Stay, s) \doteq \exists x, y.\, At(ag, x, y, s) \\
&LegalM(ag, Exit, s) \doteq \\
&\quad ag = Ag_3 \wedge \neg Terminal(s) \wedge AtExit(Ag_3, s)
\end{aligned}$$

Thus an agent ag may perform move $move(d)$ in s to move one step in direction d provided that the game is not yet finished and moving in direction d is possible given ag's position in s. An agent may also perform move $Stay$ in a situation s to remain where he is provided that he is on the grid in s. Finally an agent may perform move $Exit$ in s to exit the grid provided he is Ag_3, the game is not yet finished, and he is at an exit position in s. $Terminal(s)$, meaning that the game is finished in situation s, holds if Ag_3 is at the same position as one of the other agents in s and is "captured", in which case Ag_1 and Ag_2 win, or if Ag_3 has "exited" the grid in s, in which case Ag_3 wins, and is defined as:

$$\begin{aligned}
&Terminal(s) \doteq \exists x, y. At(Ag_1, x, y, s) \wedge At(Ag_3, x, y, s) \vee \\
&\quad \exists x, y. At(Ag_2, x, y, s) \wedge At(Ag_3, x, y, s) \vee \\
&\quad \neg \exists x, y. At(Ag_3, x, y) \\
&Wins(ag, s) \doteq Terminal(s) \wedge \\
&\quad ag = Ag_3 \wedge \neg \exists x, y. At(Ag_3, x, y, s) \vee \\
&\quad ag \neq Ag_3 \wedge \exists x, y. At(Ag_3, x, y, s) \wedge At(ag, x, y, s)
\end{aligned}$$

[4] In many cases, moves don't interfere with each other and the effects are just the union of those of each move. One can also exploit previous work on axiomatizing parallel actions to generate successor state axioms [26, 25].

$$AtExit(ag, s) \doteq At(ag, 1, 1, s) \vee At(ag, 5, 1, s) \vee At(ag, 1, 5, s)$$

The following successor state axiom specifies how the game state changes:

$$At(ag, x, y, do(a, s)) \equiv \exists u, v. MovesTo(ag, u, v, x, y, a, s) \vee At(ag, x, y, s) \wedge \neg\exists u, v. MovesTo(ag, x, y, u, v, a, s) \wedge \neg\exists m_1, m_2.(a = tick(m_1, m_2, Exit) \wedge ag = Ag_3)$$

$$MovesTo(ag, u, v, x, y, a, s) \doteq \exists m_1, m_2, m_3, d. a = tick(m_1, m_2, m_3) \wedge At(ag, u, v, s) \wedge Adj(u, v, d, x, y) \wedge [ag = Ag_1 \wedge m_1 = move(d) \vee ag = Ag_2 \wedge m_2 = move(d) \vee ag = Ag_3 \wedge m_3 = move(d) \wedge \neg Capturing(Ag_3, m_3, Ag_1, m_1, s) \wedge \neg Capturing(Ag_3, m_3, Ag_2, m_2, s)]$$

$$Capturing(ag, m, ag', m', s) \doteq \exists x, y, u, v, d, d'. ag = Ag_3 \wedge At(Ag_3, x, y, s) \wedge (ag' = Ag_1 \vee ag' = Ag_2) \wedge At(ag', u, v, s) \wedge m = move(d) \wedge m' = move(d') \wedge Adj(x, y, d, u, v) \wedge Adj(u, v, d', x, y)$$

The successor state axiom for At essentially says that agent ag moves to position (x, y) in situation $do(a, s)$ if a is a tick joint move where ag performs a move in direction d, from a position (u, v), which is adjacent from (x, y) in direction d, and moreover if ag is Ag_3, then he is not "being captured" by one of the other agents (see below). Otherwise, ag remains at the position where he was in situation s except if ag is Ag_3 and he performs move Exit, in which case he will no longer be at any position on the grid. The axiom uses a defined fluent (an abbreviation), $Capturing(ag, m, ag', m', s)$, meaning that ag' performing move m' is capturing agent ag performing move m in situation s, which holds if and only if ag is Ag_3 and ag' is one of the other agents, and ag and ag' are performing moves that overlap, i.e., where the starting position of one move is the ending position of another.

The following axioms define the non-fluent predicates that we use:

$$Agent(ag) \equiv ag = Ag_1 \vee ag = Ag_2 \vee ag = Ag_3$$

$$Move(m) \equiv \exists d. Dir(d) \wedge m = move(d) \vee m = Stay \vee m = Exit$$

$$Dir(d) \equiv d = N \vee D = E \vee d = S \vee d = W \quad \text{— directions}$$

$$Co(x) \equiv x = 1 \vee x = 2 \vee x = 3 \vee x = 4 \vee x = 5 \quad \text{— coordinates}$$

$$Succ(x, y) \equiv \quad \text{— } y \text{ is successor of } x \quad x = 1 \wedge y = 2 \vee x = 2 \wedge y = 3 \vee x = 3 \wedge y = 4 \vee x = 4 \wedge y = 5$$

$$Adj(x, y, d, x', y') \equiv \quad \text{— } (x', y') \text{ is adjacent from } (x, y) \quad d = N \wedge x' = x \wedge Co(x) \wedge Succ(y, y') \vee \quad \text{in direction } d \quad d = S \wedge x' = x \wedge Co(x) \wedge Succ(y', y) \vee d = E \wedge y' = y \wedge Co(y) \wedge Succ(x, x') \vee d = W \wedge y' = y \wedge Co(y) \wedge Succ(x', x)$$

The initial state is specified as follows:

$$At(ag, x, y, S_0) \equiv ag = Ag_1 \wedge x = 1 \wedge y = 1 \vee ag = Ag_2 \wedge x = 5 \wedge y = 1 \vee ag = Ag_3 \wedge x = 1 \wedge y = 5$$

The precondition axiom for the tick action is as discussed earlier. We also have unique names for moves, agents, directions, and positions. □

Note that it easy to obtain an infinite states version of this game, for instance, by using an infinite grid, with positions (x, y) for all $x, y \in \mathbb{N}$. Then Ag_3 can run away to avoid getting caught and the others cannot corner her. We can then say that the game ends if the guards catch Ag_3 or if Ag_3 gets North and East of both guards past a given area (they can't catch up if he keeps going North-East).

Let's now consider another simple example which has infinite states. It is not a game in the traditional sense, but we can analyse what properties agents can enforce in it.

Example 2 *We have a repair shop where items arrive, are repaired, and then shipped. Items are denoted by a countably infinite set of constants $Item_1$, $Item_2$, ..., for which we have unique name axioms. Ag_1 represents the environment, Ag_2 is a repairing robot, and Ag_3 is a shipper agent. We have the following legal move axioms:*

$$LegalM(ag, Wait, s) \doteq True$$

$$LegalM(ag, arrive(i), s) \doteq ag = Ag_1 \wedge Item(i) \wedge \neg InShop(i, s)$$

$$LegalM(ag, repair(i), s) \doteq ag = Ag_2 \wedge InShop(i, s)$$

$$LegalM(ag, ship(i), s) \doteq ag = Ag_3 \wedge Repaired(i, s)$$

and the following successor state axioms:

$$InShop(i, do(a, s)) \equiv \exists m, m'. a = tick(arrive(i), m, m') \vee InShop(i, s) \wedge \neg\exists m, m'. a = tick(m, m', ship(i))$$

$$Repaired(i, do(a, s)) \equiv \exists m, m'. a = tick(m, repair(i), m') \vee Repaired(i, s) \wedge \neg\exists m, m'. a = tick(arrive(i), m, m')$$

$$Shipped(i, do(a, s)) \equiv \exists m, m'. a = tick(m, m', ship(i)) \vee Shipped(i, s) \wedge \neg\exists m, m'. a = tick(arrive(i), m, m')$$

We also have initial state axioms saying that initially no items are in the shop, or have been repaired or shipped. Clearly, the domain is infinite, as is the number of moves. □

4 Relationship with GDL

SCSGSs are closely related to GDL specifications. We show that GDL game descriptions where auxiliary predicates are "acyclic" or "hierarchical" (without direct or indirect recursion) [20] can be translated into SCSGSs. Notice that formalisms in which the state description is based on first-order logic (FOL), such as the situation calculus, cannot capture predicates on state defined recursively.

We first define a translation function $\tau_{a,s}$ for translating the bodies of the rules for defining the initial situation, next situation, and legal moves; only `true`, `does` atoms and auxiliary predicates $\texttt{aux}(\vec{x})$ can occur in bodies:

$$\begin{aligned}
\tau_{a,s}(\texttt{true}) &= \texttt{true}\\
\tau_{a,s}(\texttt{true}(F(\vec{t}))) &= F(\vec{t})[s]\\
\tau_{a,s}(\texttt{does}(R, M))) &= \exists m_1 \ldots \exists m_{R-1} \exists m_{R+1} \ldots \exists m_n\\
&\quad a = tick(m_1, \ldots, m_{R-1}, M, m_{R+1}, \ldots, m_n)\\
\tau_{a,s}(\texttt{aux}(\vec{x})) &= \exists \vec{y}. \tau_{a,s}(body_{\texttt{aux}}(\vec{x}, \vec{y}))\\
\tau_{a,s}(\alpha_1 \wedge \alpha_2) &= \tau_{a,s}(\alpha_1) \wedge \tau_{a,s}(\alpha_2)\\
\tau_{a,s}(\neg\alpha) &= \neg\tau_{a,s}(\alpha)
\end{aligned}$$

where $body_{\texttt{aux}}(\vec{x}, \vec{y})$ denotes the body of the rule for $\texttt{aux}(\vec{x})$ (which may involve disjunctions).

Initial situation. In GDL, the initial situation is specified by a set of clauses of the form $\texttt{init}(F(\vec{t})) \leftarrow body(\vec{t}, \vec{y})$, where $body(\vec{t}, \vec{y})$ includes only `true` atoms and auxiliary predicates (facts are represented as $\texttt{init}(F(\vec{t}) \leftarrow \texttt{true})$, involving terms $\vec{t}$ and additional existential variables $\vec{y}$. In the SitCalc, we capture this through a set of FOL formulas:

$$F(\vec{x}, S_0) \equiv \bigvee_{\texttt{init}(F(\vec{t})) \leftarrow body(\vec{t}, \vec{y})} \vec{x} = \vec{t} \wedge \exists \vec{y}. \tau_{\_, S_0}(body(\vec{t}, \vec{y}))$$

This is the familiar completion of the set of `init` clauses, which captures their semantics given that the set of clauses is acyclic. Note that since the bodies of these clauses cannot contain `does` atoms, the action parameter of τ is irrelevant. We have a complete specification, so there is a single model.

Effects. In GDL, the next state resulting from moves is specified by a set of clauses of the form $\texttt{next}(F(\vec{t})) \leftarrow body(\vec{t},\vec{y})$, where *body* includes only `true` and `does` atoms, and auxiliary predicates. In the SitCalc, we capture this description through successor state axioms of the form:

$$F(\vec{x}, do(a,s)) \equiv \bigvee_{\texttt{next}(F(\vec{t}))\leftarrow body(\vec{t},\vec{y})} \vec{x} = \vec{t} \wedge \exists \vec{y}.\tau_{a,s}(body(\vec{t},\vec{y}))$$

Preconditions and legality. In GDL, the legality conditions for a move M by a role R are expressed by a set of clauses of the form $\texttt{legal}(R, M(\vec{t})) \leftarrow body(\vec{t},\vec{y})$, where *body* contains only `true` and auxiliary predicates. In the SitCalc, we capture this through axioms of the form:

$$LegalM(R, M(\vec{x}), s) \equiv \bigvee_{\texttt{legal}(R,M(\vec{t}))\leftarrow body(\vec{t},\vec{y})} \vec{x} = \vec{t} \wedge \exists \vec{y}.\tau_{\_,s}(body(\vec{t},\vec{y}))$$

The preconditions of the $tick$ joint move action are specified by the action precondition axiom given earlier.

Goals and terminal states. For GDL goals, we have clauses of the form $\texttt{goal}(R, V) \leftarrow body(\vec{t},\vec{y})$, where *body* includes only `true` and auxiliary predicates. In the SitCalc, we have:

$$Goal(r, v, s) \equiv \bigvee_{\texttt{goal}(R,V)\leftarrow body(\vec{t},\vec{y})} r = R \wedge v = V \wedge \exists \vec{y}.\tau_{\_,s}(body(\vec{t},\vec{y}))$$

Similarly for defining termination, we have in GDL clauses of the form, $\texttt{terminal} \leftarrow body(\vec{y})$, where *body* includes only `true` and auxiliary predicates. So we have:

$$Terminal(s) \equiv \bigvee_{\texttt{terminal}\leftarrow body(\vec{y})} \exists \vec{y}.\tau_{\_,s}(body(\vec{y}))$$

Unique name and domain closure for objects. We additionally need to impose the unique name assumption and domain closure for the object sort in the SitCalc to conform to the GDL assumption that object terms are interpreted as themselves. In the SitCalc this corresponds to assuming we have standard names for objects [18].

We can now show that the above mapping is correct.

Theorem 3 *For any GDL specification that uses acyclic auxiliary predicates only, the above translation is correct, i.e., it produces a SCSGS whose only model is bisimilar to the transition system associated with the GDL specification.*

Proof (sketch). Notice that we have complete information. This means the resulting SCSGS $\mathcal{D}$ has only one SitCalc model $\mathcal{M}$ (up to isomorphism). We can associate to such a model $\mathcal{M}$ a *transition system* $T_{\mathcal{M}} = \langle \Delta, \mathcal{S}, S_0, \rightarrow_{\mathcal{M}}, L_{\mathcal{M}} \rangle$ *induced by* $\mathcal{M}$ where:

- Δ is the object domain of $\mathcal{M}$, which is isomorphic to the set of *all ground object terms* since we have unique name and domain closure for objects.
- $\mathcal{S}$ is the set of *possible states* formed by all situations;
- $S_0 \in \mathcal{S}$ is the *initial state*, where S_0 is the initial situation;
- $\rightarrow_{\mathcal{M}} \subseteq \mathcal{S} \times \mathcal{S}$ is the *transition relation* s.t. $s \rightarrow_{\mathcal{M}} s'$ iff there exists some a s.t. $s' = do^{\mathcal{M}}(a, s)$ and $(a, s) \in Poss^{\mathcal{M}}$; note that a will be some instantiation of the $tick$ action type for some move arguments;
- $L_{\mathcal{M}} : \mathcal{S} \mapsto Int^{\mathcal{M}}$ is the *labeling function* associating each state/situation s with a first-order (FO) interpretation $I = L_{\mathcal{M}}(s)$ s.t. $F^I = \{\vec{o} \mid \mathcal{M} \models F(\vec{o}, s)\}$, for every predicate fluent.

On the other hand, one can use the techniques in [29] to generate a transition system for the GDL specification G. We can associate to such a game description G a transition system $T_G = \langle \Delta, Q, q_0, \rightarrow_G, L_G \rangle$ where:

- Δ is the set of *all ground object terms*.
- Q is the set of *possible states* formed by all possible finite subsets of ground "fluent" atoms;
- $q_0 = \{F(\vec{t}) | G \models \texttt{init}(F(\vec{t}))\}$;
- $\rightarrow_G \subseteq Q \times Q$ is the *transition relation* s.t. $q \rightarrow_G q'$ iff there exists some ground move terms $M_1, \ldots M_n$ s.t. $G \cup q \models \texttt{does}(Ag_i, M_i)$ (for $i = 1, \ldots, n$) and $q' = \{F(\vec{t}) | G \cup q \cup \{\texttt{does}(Ag_1, M_1), \ldots, \texttt{does}(Ag_n, M_m)\} \models \texttt{next}(F(\vec{t}))\}$;
- $L_G : Q \mapsto Int^G$ is the *labeling function* associating each state q with a FO interpretation $I = L_G(q)$ s.t.
  - $Goal^I = \{(Ag_i, v) \mid G \cup q \models \texttt{goal}(Ag_i, v)\}$,
  - $Terminal^I = \texttt{true}$ iff $G \cup q \models \texttt{terminal}\}$, and
  - $F^I = \{\vec{t} \mid G \cup q \models F(\vec{t})\}$ for all other predicates.

The two transition systems $T_{\mathcal{M}}$ and $\mathcal{T}_G$ are bisimilar. Indeed there is a relation $\mathcal{B}$ including (S_0, q_O) such that if $(s, q) \in \mathcal{B}$ then: *(i)* $L_{\mathcal{M}}(s)$ is isomorphic to $L_G(q)$; *(ii)* for all s' such that $s \rightarrow_{\mathcal{M}} s'$, there exists a q' such that $q \rightarrow_G q'$ and $(s', q') \in \mathcal{B}$; *(iii)* for all q' such that $q \rightarrow_G q'$, there exists a s' such that $s \rightarrow_{\mathcal{M}} s'$ and $(s', q') \in \mathcal{B}$. One can check that one such relation is the isomorphism between state labeling of the transitions systems: i.e. $\mathcal{B} = \{(s, q) \mid L_{\mathcal{M}}(s)$ is isomorphic to $L_G(q)\}$. Given the bisimilarity-invariance of the μ-calculus, we get that the two transition systems satisfy the same μATL-FO formulas (see Sec. 5). □

Notice that [34] shows trace-equivalence between GDL specifications and their translation into the $\mathcal{C}^+$ action language [14]. Remember that bisimilarity implies trace-equivalence. GDL also requires that game specifications be stratified, "allowed", and satisfy some restrictions on recursion that ensure that the specification is equivalent to a finite set of ground clauses [34]. In principle, when the game is finite state as assumed in [13], we could drop the acyclicity restriction and capture GDL in its entirety at the cost of compositionality.

5 Verification

To express properties about SCSGSs, we introduce a specific logic μATL-FO, inspired by alternating-time μ-calculus, μATL, which is a well-known generalization of ATL [1]. Our logic is a first-order variant of the μ-calculus [2] that works on games, by suitably considering coalitions acting towards the realization of a temporally extended goal, as in μATL. The key building block in these kinds of logics is the so-called *force-next* operator, which in our case is:

$$\begin{aligned}&\langle\langle G\rangle\rangle \bigcirc \varphi \equiv \\ &\quad \exists m_{g_1}, \ldots, m_{g_k} \cdot \textstyle\bigwedge_{\{g_i,\ldots,g_k\}=G} LegalM(g_i, m_{g_i}, now) \wedge \\ &\quad \exists m_{g_{k+1}}, \ldots, m_{g_n} \cdot \textstyle\bigwedge_{\{g_{k+1},\ldots,g_n\}=\overline{G}} LegalM(g_i, m_{g_i}, now) \wedge \\ &\quad \forall m_{g_{k+1}}, \ldots, m_{g_n} \cdot \textstyle\bigwedge_{\{g_{k+1},\ldots,g_n\}=\overline{G}} LegalM(g_i, m_{g_i}, now) \\ &\qquad \supset \varphi(do(tick(m_{g_1}, \ldots, m_{g_n}), now))\end{aligned}$$

Above, φ is a situation suppressed formula, i.e., one with situation arguments in fluents suppressed (syntactically replaced by a placeholder now). We denote by $\varphi[s]$ the formula obtained by restoring

the suppressed situation argument s into all fluents in φ. Here, we quantify existentially on legal moves when the agent is in the coalition G, and universally when it is not. We are looking for some move for each agent in the coalition G, such that for all moves by the agents not in the coalition, φ becomes true next. Notice that in any case both agents in the coalition and agents outside it must have a legal move.

Then, following [11], we define the logic μATL-FO as:

$$\Psi \leftarrow \varphi \mid Z \mid \neg\Psi \mid \Psi_1 \wedge \Psi_2 \mid \exists x.\Psi \mid \langle\langle G\rangle\rangle \bigcirc \Psi \mid \mu Z.\Psi(Z)$$

where φ is an arbitrary, possibly open, situation-suppressed SitCalc uniform formula, Z is a predicate variable of a given arity, and $\langle\langle G\rangle\rangle \bigcirc \Psi$ is as defined above. $\mu Z.\Psi(Z)$ is the *least fixpoint* construct from the μ-calculus, which denotes the least fixpoint of the formula $\Psi(Z)$ (we use this notation to emphasize that Z may occur free, i.e., not quantified by μ in Ψ). Similarly $\nu Z.\Psi(Z)$, defined as $\neg\mu Z.\neg\Phi[Z/\neg Z]$ (where we denote with $\Phi[Z/\neg Z]$ the formula obtained from Φ by substituting each occurrence of Z with $\neg Z$), denotes the *greatest fixpoint* of $\Psi(Z)$. We also use the usual abbreviations for first-order logic such as disjunction ($\vee$) and universal quantification $\forall$. Moreover we denote by $[[G]]\bigcirc\Psi$ the dual of $\langle\langle G\rangle\rangle\bigcirc\Psi$, i.e., $[[G]] \bigcirc \Psi \doteq \neg\langle\langle G\rangle\rangle \bigcirc \neg\Psi$.

As usual in the μ-calculus, formulas of the form $\mu Z.\Psi(Z)$ (and $\nu Z.\Psi(Z)$) must obey the *syntactic monotonicity* of $\Psi(\cdot)$ w.r.t. Z, which states that every occurrence of the second-order variable Z in $\Psi(Z)$ must be within the scope of an even number of negation symbols. This ensures that both the least fixpoint $\mu Z.\Psi(Z)$ and the greatest fixpoint $\nu Z.\Psi(Z)$ always exist.

The least fixpoint formula $\mu Z.\Psi$ is true in a situation if and only if it belongs to the least set of situations Z that satisfy the temporal formula $\Psi(Z)$, where Z is a second-order predicate variable ranging over sets of situations (a formal semantics is given below). Similarly, the greatest fixpoint formula $\nu Z.\Psi$ holds in a situation if it belongs to the largest set of situations Z that satisfy $\Psi(Z)$. Using these least and greatest fixpoint constructs, we can express the ability of *forcing* arbitrary temporal and dynamic properties. For instance, to say that group G has a strategy to force achieving $\varphi(\vec{x})$ eventually, where $\varphi(\vec{x})$ is a situation suppressed formula with free variables $\vec{x}$, we use the following least fixpoint formula:

$$\mu Z.\ \varphi(\vec{x}) \vee \langle\langle G\rangle\rangle \bigcirc Z$$

In a first-order ATL, this could be expressed as $\langle\langle G\rangle\rangle\Diamond\varphi(\vec{x})$. Similarly, we use the greatest fixpoint construct to express the ability of a coalition G to force maintaining property φ:

$$\nu Z.\varphi(\vec{x}) \wedge \langle\langle G\rangle\rangle \bigcirc Z$$

In a first-order ATL, this could be expressed as $\langle\langle G\rangle\rangle\Box\varphi(\vec{x})$.

The formal semantics of μATL-FO is based on characterizing how to evaluate μATL-FO formulas in a SitCalc model $\mathcal{M}$. To do so, since μATL-FO contains formulas with both individual and predicate free variables, we need to introduce an individual variable valuation v, and a predicate variable valuation V, i.e., a mapping from predicate variables Z to subsets of the set of all situations $\mathcal{S}$. Then, we assign meaning to formulas by associating to $\mathcal{M}$, v, and V an *extension function* $(\cdot)^{\mathcal{M}}_{v,V}$, which maps formulas to subsets of $\mathcal{S}$, and is defined inductively as follows:

$$\begin{array}{lcl}
(\varphi)^{\mathcal{M}}_{v,V} & = & \{s \in \mathcal{S} \mid \mathcal{M} \models \varphi[s]\} \\
(\neg\Psi)^{\mathcal{M}}_{v,V} & = & \mathcal{S} - (\Psi)^{\mathcal{M}}_{v,V} \\
(\Psi_1 \wedge \Psi_2)^{\mathcal{M}}_{v,V} & = & (\Psi_1)^{\mathcal{M}}_{v,V} \cap (\Psi_2)^{\mathcal{M}}_{v,V} \\
(\exists x.\Psi)^{\mathcal{M}}_{v,V} & = & \{s \in \mathcal{S} \mid \text{ exists } t \text{ s.t. } s \in (\Psi)^{\mathcal{M}}_{v[x/t],V}\} \\
(\langle\langle G\rangle\rangle \bigcirc \Psi)^{\mathcal{M}}_{v,V} & = & \{s \in \mathcal{S} \mid s \in Pre(G, (\Psi)^{\mathcal{M}}_{v,V})\} \\
(Z(\vec{t})^{\mathcal{M}}_{v,V}) & = & V(Z) \\
(\mu Z.\Psi)^{\mathcal{M}}_{v,V} & = & \bigcap\{\mathcal{E} \subseteq \mathcal{S} \mid (\Psi)^{\mathcal{M}}_{v,V[Z/\mathcal{E}]} \subseteq \mathcal{E}\}
\end{array}$$

where:

$$\begin{array}{l}
Pre(G,\mathcal{E}) = \{s \in \mathcal{S} \mid \\
\exists m_{g_1},\ldots,m_{g_k} \cdot \bigwedge_{\{g_i,\ldots,g_k\}=G}(\mathcal{M} \models LegalM(g_i,m_{g_i},s)) \wedge \\
\exists m_{g_{k+1}},\ldots,m_{g_n} \cdot \bigwedge_{\{g_{k+1},\ldots,g_n\}=\overline{G}}(\mathcal{M} \models LegalM(g_i,m_{g_i},s)) \wedge \\
\forall m_{g_{k+1}},\ldots,m_{g_n} \cdot \bigwedge_{\{g_{k+1},\ldots,g_n\}=\overline{G}}(\mathcal{M} \models LegalM(g_i,m_{g_i},s)) \\
\qquad \supset do(tick(m_{g_1},\ldots,m_{g_n}),s) \in \mathcal{E}\}
\end{array}$$

Note that given a valuation V and a predicate variable Z and a set of situations $\mathcal{E}$ we denote by $V[Z/\mathcal{E}]$ the valuation obtained from V by changing the value of Z to $\mathcal{E}$. Similarly for v. Notice also that when a μATL-FO formula Ψ is closed (w.r.t. individual and predicate variables), its extension $(\Psi)^{\mathcal{M}}_{v,V}$ does not depend on the valuations v and V, and we denote the extension of Ψ simply by $(\Psi)^{\mathcal{M}}$. We say that a closed formula Ψ holds in the SitCalc model $\mathcal{M}$, denoted by $\mathcal{M} \models \Psi$, if $S_0 \in (\Psi)^{\mathcal{M}}$.

Example 4 *Several key properties of games [13] can easily be expressed in* μATL-FO*, for example:*

- Playability*, i.e., at every step which is not terminal there exists a legal joint move:*

$$\nu Z.\ Terminal \vee \langle\langle ALL\rangle\rangle \bigcirc Z$$

- Termination*, i.e., there is a way of playing the game that eventually leeds to termination:*

$$\mu Z.\ Terminal \vee \langle\langle ALL\rangle\rangle \bigcirc Z$$

- Weak Winnability (by agent Ag)*, i.e., there is a way for agent Ag to win if the others cooperate:*

$$\mu Z.\ Terminal \wedge \exists v.Goal(Ag,v) \wedge (\textstyle\bigwedge_{Ag' \neq Ag} \exists v'.Goal(Ag',v') \wedge v' \leq v) \vee \langle\langle ALL\rangle\rangle \bigcirc Z$$

- Strong Winnability (by agent Ag)*, i.e., there is a way for agent Ag to win no matter what the others do:*

$$\mu Z.\ Terminal \wedge \exists v.Goal(Ag,v) \wedge (\textstyle\bigwedge_{Ag' \neq Ag} \exists v'.Goal(Ag',v') \wedge v' \leq v) \vee \langle\langle \{Ag\}\rangle\rangle \bigcirc Z$$

- Well-formed*: if terminating, playable and weakly winnable.*

In Example 1, one can check that the game is weakly winnable for all agents, and that it becomes strongly winnable for either Ag_3 or for the coalition $\{Ag_1, Ag_2\}$ starting from certain initial configurations (where we change the initial position of some players). In Example 2, one can check that Ag_2 can ensure that all items that arrive are eventually repaired, and that Ag_2 and Ag_3 together can ensure that all are eventually shipped. □

Let us now discuss how one can effectively verify μATL-FO formulas against a SCSGS in three key cases.

Propositional case. The propositional case is the one where the object domain is assumed to be finite. If this is the case, actions are also finite (we have finite moves types and only one action type). Hence, the only domain that remains infinite is that of situations (though now the situation tree is only finitely branching). However successor state axioms ensure that fluents in a given situation depend only on the values of the fluents in the previous situation (not the history). Hence in the presence of a finite object domain, one can abstract situations into "states", which are the interpretation of the fluents for that situation [32]. Consequently one can show that there is a finite transition system bisimilar to the SitCalc model, for example along the lines of [8]. Given the invariance with respect to bisimulation of the μ-calculus, one can use such a finite transition system for the evaluation, or model checking, of the μATL-FO formulas (notice also that first-order quantification can be eliminated because of the finite object domain). Thus we have the following result:

Theorem 5 *Let $\mathcal{D}$ be a SCSGS with a finite object domain and Ψ a μATL-FO formula. Then checking whether $\mathcal{D} \models \Psi$ is decidable.*

In practice μATL-FO reduces to standard alternating-time μ-calculus (μATL) [1], and one can use standard algoritms and tools for the verification. In fact such tools can also be used for synthesis by considering that strategies can be extracted from the existential choices in the $\langle\langle G\rangle\rangle \bigcirc \varphi$ operators as discussed in [1]. Verification techniques for propositional GDL descriptions have been proposed in [27].

Bounded first-order case. We say that a SitCalc theory is bounded if in spite of having an infinite object domain, it allows only a bounded number of object tuples in the extension of fluents in each situation [8]. Intuitively this is like saying that we have a bookshelf of a fixed size in which we can freely add, remove and replace books as long as we remain within the fixed size of the bookshelf. For instance, we can obtain an infinite-states bounded version of Example 2 as follows: we make the $arrive(i)$ move illegal if there are already k items in the shop; we make $Repaired$ become false when an item is shipped, so it is also bounded by k; finally we replace $Shipped(i, s)$ by $JustShipped(i, s)$, which only holds in the situation that follows the $ship(i)$ action, i.e., for at most one item ($Move(m)$ and $Item(i)$ can be viewed as abbreviations, the latter standing for anything that is not an agent or move). Such bounded action theories are known to be decidable for model checking a first-order variant of the μ-calculus without first-order quantification across situations [8] as well as with quantification across [8, 15, 3]. Such results can be adapted to show that μATL-FO model checking against SCSGSs is decidable:

Theorem 6 *Let $\mathcal{D}$ be a SCSGS that is a bounded action theory and Ψ a μATL-FO formula. Then checking whether $\mathcal{D} \models \Psi$ is decidable.*

Proof (sketch). If Ψ does not include first-order quantification across situations, we can apply the techniques in [8] which allow for building a finite transition system that is bisimilar to the one induced by the SitCalc theory model (which is essentially unique if we assume complete information). If quantification across is allowed, we cannot use the techniques in [8] in general, but we can still use a similar finite faithful abstraction if the quantification is over objects in the *active domain* and is restricted to be *persistence-preserving* [9]. Finally if we allow unrestricted quantification, we can still generate a finite faithful abstraction, which however, in this case depends on the number of variables in Ψ as well [3, 4]. The key to adapting the original proofs to our case is to suitably reformulate the preimage construction used in evaluating μATL-FO formulas, so has to handle the coalition existentially and the adversaries universally. □

Synthesis can be done as in the propositional case.

General first-order case. In the general case, model checking of μATL-FO in SCSGSs is undecidable. For example it is immediate to reconstruct the undecidability result in [15] using very simple SCSGSs and μATL-FO formulas. In this case, we can adopt the approach of [11], and base the verification method on two main ingredients: *(i) regression* [26], and *(ii) fixpoint approximates* and the classical Knaster and Tarski results [31].

Regarding regression, note that with $LegalM$ defined as in Section 3, if φ is regressable then $\langle\langle G\rangle\rangle \bigcirc \varphi$ is also regressable, and in fact its (one step) regression is:

$$\begin{array}{l}\mathcal{R}(\langle\langle G\rangle\rangle \bigcirc \varphi) \doteq \\ \quad \exists m_{g_1}, \ldots, m_{g_k} \cdot \bigwedge_{\{g_i,\ldots,g_k\}=G} LegalM(g_i, m_{g_i}, now)) \wedge \\ \quad \exists m_{g_{k+1}}, \ldots, m_{g_n} \cdot \bigwedge_{\{g_{k+1},\ldots,g_n\}=\overline{G}} LegalM(g_i, m_{g_i}, now) \wedge \\ \quad \forall m_{g_{k+1}}, \ldots, m_{g_n} \cdot \bigwedge_{\{g_{k+1},\ldots,g_n\}=\overline{G}} LegalM(g_i, m_{g_i}, now) \\ \qquad \supset \mathcal{R}(\varphi(do(tick(m_{g_1}, \ldots, m_{g_n}), now)))\end{array}$$

The second element is the ability, in some cases, to compute fixpoint approximates. Suppose that we want to verify a least fixpoint formula $\mu Z.\Psi(Z)$, where Z occurs free in Ψ. We can try to evaluate this formula using the general technique of iterated fixpoint approximates, which guarantees that for some transfinite ordinal we get the fixpoint [31]. The technique goes as follows. The approximates for a least fixpoint of the form $\mu Z.\Psi(Z)$ are as follows:

$$\begin{array}{l} Z_0 \doteq \Psi(False) \\ Z_1 \doteq \Psi(Z_0) \\ Z_2 \doteq \Psi(Z_1) \\ \ldots \end{array}$$

Observe that all of these formulas Z_i are situation suppressed which means that they all talk about the same situation, say now.

At limit transfinite ordinals ω we have that:

$$Z_\omega = \bigvee_i Z_i$$

Notice that in order to express this approximate we need infinitary disjunction (for least fixpoint as here, and conjunctions for greatest fixpoint).[5] However, this technique becomes effective only when such a fixpoint can be reached within a finite number of iterations. For an in-depth discussion, see [11].

6 Golog-Based Players

We can also use programs to specify the possible behaviors of the agents playing the game. In particular, we can assume that each agent Ag_i is following a program δ_i specifying her possible moves at each step. For this, we use programs in a variant of the Golog programming language [19] where instead of atomic actions, we use *moves*. Such programs cannot be run in isolation; they must be executed concurrently with all agents moving synchronously. Programs constructs are the following:

m	atomic move
$\varphi?$	test for a condition
$\delta_1; \delta_2$	sequence
if φ **then** δ_1 **else** δ_2	conditional
while φ **do** δ	while loop
$\delta_1 \vert \delta_2$	nondeterministic branch
$\pi x.\delta$	nondeterministic choice of argument
δ^*	nondeterministic iteration

[5] By the way, notice that the fixpoint formulas in the μ-calculus are not continuous, so going above limit ordinals is in general necessary.

$$\mathit{TransM}(m, s, \delta', m') \equiv m' = m \wedge \delta' = \mathit{nil}$$

$$\mathit{TransM}(\varphi?, s, \delta', m') \equiv \mathit{False}$$

$$\mathit{TransM}(\delta_1; \delta_2, s, \delta', m') \equiv \exists \delta_1'.\mathit{TransM}(\delta_1, s, \delta_1', m') \wedge \delta' = \delta_1'; \delta_2 \vee \mathit{FinalM}(\delta_1, s) \wedge \mathit{TransM}(\delta_2, s, \delta', m')$$

$$\mathit{TransM}(\textbf{if } \varphi \textbf{ then } \delta_1 \textbf{ else } \delta_2, s, \delta', m') \equiv \varphi[s] \wedge \mathit{TransM}(\delta_1, s, \delta', m') \vee \neg\varphi[s] \wedge \mathit{TransM}(\delta_2, s, \delta', m')$$

$$\mathit{TransM}(\textbf{while } \varphi \textbf{ do } \delta, s, \delta', m') \equiv \varphi[s] \wedge \exists \delta''.\mathit{TransM}(\delta, s, \delta'', m') \wedge \delta' = \delta''; (\textbf{while } \varphi \textbf{ do } \delta)$$

$$\mathit{TransM}(\delta_1|\delta_2, s, \delta', m') \equiv \mathit{TransM}(\delta_1, s, \delta', m') \vee \mathit{TransM}(\delta_2, s, \delta', m')$$

$$\mathit{TransM}(\pi z.\delta, s, \delta', m') \equiv \exists z.\mathit{TransM}(\delta, s, \delta', m')$$

$$\mathit{TransM}(\delta^*, s, \delta', m') \equiv \exists \delta''.\mathit{TransM}(\delta, s, \delta'', m') \wedge \delta' = \delta''; \delta^*$$

$$\mathit{TransM}(\mathit{nil}, s, \delta', m') \equiv \mathit{False}$$

$$\mathit{FinalM}(m, s) \equiv \mathit{False}$$

$$\mathit{FinalM}(\varphi?, s) \equiv \varphi[s]$$

$$\mathit{FinalM}(\delta_1; \delta_2, s) \equiv \mathit{FinalM}(\delta_1, s) \wedge \mathit{FinalM}(\delta_2, s)$$

$$\mathit{FinalM}(\textbf{if } \varphi \textbf{ then } \delta_1 \textbf{ else } \delta_2, s) \equiv \varphi[s] \wedge \mathit{FinalM}(\delta_1, s) \vee \neg\varphi[s] \wedge \mathit{FinalM}(\delta_2, s)$$

$$\mathit{FinalM}(\textbf{while } \varphi \textbf{ do } \delta, s) \equiv \varphi[s] \wedge \mathit{FinalM}(\delta, s) \vee \neg\varphi[s]$$

$$\mathit{FinalM}(\delta_1|\delta_2, s) \equiv \mathit{FinalM}(\delta_1, s) \vee \mathit{FinalM}(\delta_2, s)$$

$$\mathit{FinalM}(\pi x.\delta, s) \equiv \exists x.\mathit{FinalM}(\delta, s)$$

$$\mathit{FinalM}(\delta^*, s) \equiv \mathit{True}$$

$$\mathit{FinalM}(\mathit{nil}, s) \equiv \mathit{True}$$

Figure 1. Axioms specifying *TransM* and *FinalM*

In the above, m is a term that represents a move, possibly with parameters, and φ is situation-suppressed SitCalc formula. Program $\delta_1|\delta_2$ allows for the nondeterministic choice between programs δ_1 and δ_2, while $\pi x.\delta$ executes program δ for *some* nondeterministic choice of a legal binding for variable x (observe that such a choice is, in general, unbounded). δ^* performs δ zero or more times. Note that we leave out recursive procedures.

To assign semantics to such programs we use notions analogous to *Trans* and *Final* from ConGolog's transition semantics [6]. In particular we introduce the predicate $\mathit{TransM}(\delta, s, \delta', m)$ to mean that the program δ in situation s can perform move m leaving δ' as the remaning program to execute, and the predicate $\mathit{FinalM}(\delta, s)$ to mean that program δ can be considered terminated in situation s. The definiton of these predicates appears in Figure 1. We can read these axioms as follows: A program consisting of an atomic move m can only perform move m with the remaining program being the "empty" program nil. A test program $\varphi?$ can never perform a move. A sequence $\delta_1; \delta_2$ can perform move m in situation s if δ_1 can perform it with the remaining program being what remains of δ_1 followed by δ_2 or if δ_1 can be considered completed in s and δ_2 can perform move m in s with the remaining program being what remains of δ_2. An **if** φ **then** δ_1 **else** δ_2 can perform move m in s if δ_1 can when φ is true, and if δ_2 can when φ is false; the remaining program is what remains of the selected branch. A "while" program can perform move m in s if its condition is true in s and its body can perform m; the remaining program is what remains of the body followed by the "while" program itself. A nondeterministic branch can perform move m in s if either one of its branches can, the remaining program being what remains of the chosen branch. A nondeterministic choice of argument $\pi x.\delta$ can perform move m in s if δ can perform it for some value of variable x, which may occur in δ; the remaining program is what remains of δ for this value. A nondeterministic iteration δ^* can perform move m in s if δ can perform it; the remaining program is what remains of δ followed by δ^* again. Finally, the empty program nil can never perform a move.

For *FinalM* we have the following: A program consisting of an atomic move m is never considered terminated. A test program $\varphi?$ is considered terminated in situation s if and only if φ holds in s. A sequence $\delta_1; \delta_2$ is considered terminated in situation s if both δ_1 and δ_2 are terminated in s. An **if** φ **then** δ_1 **else** δ_2 is terminated if δ_1 is when φ is true, and if δ_2 is when φ is false. A "while" program is terminated if its condition is false or if its condition is true and its body is terminated. A nondeterministic branch is terminated if either one of its branches is. A nondeterministic choice of argument $\pi x.\delta$ is terminated if δ is terminated for some value of variable x, which may occur in δ. A nondeterministic iteration δ^* can always be considered terminated, as it can execute 0 times. Lastly, the empty program nil is always considered terminated.

Actually, we require that the agents' behavior programs be *move determined*.[6] That is, we require that the step-by-step execution of such programs be fully determined at each step by the selected move. In other words, we cannot have nondeterminism in the program once the move is selected. E.g., program $m_1; (m_2 \mid m_3)$ is move determined, but $(m_1; m_2) \mid (m_1; m_3)$ is not; with the latter, the remaining program after performing m_1 could be either m_2 or m_3. We impose this requirement because we use programs to specify *the* set of available moves for each agent in every game state. Using *TransM* we can formalize that a program δ is *move determined in a situation* s as:

$$\mathit{MoveDet}(\delta, s) \doteq \forall m, \delta', \delta''.\mathit{TransM}(\delta, s, \delta', m) \wedge \mathit{TransM}(\delta, s, \delta'', m) \supset \delta' = \delta''$$

An agent Ag_i is *move determined in a game* if its (remaining) program is move determined in every situation that the game can reach.[7]

We can then define $LegalM$ in terms of such programs by introducing a special fluent $CurrProg(Ag_i, \delta_i, s)$ that stores the remaning program of each agent in the situation:

$$\mathit{LegalM}(Ag_i, m, s) \doteq \mathit{CurrProg}(Ag_i, \delta_i, s) \wedge \exists \delta_i'.\mathit{TransM}(\delta_i, s, \delta_i', m)$$

where the successor state axiom for $CurrProg$ is as follows:

$$\mathit{CurrProg}(Ag_i, \delta_i', do(tick(m_i, \ldots, m_n), s)) \equiv \mathit{CurrProg}(Ag_i, \delta_i, s) \wedge \mathit{TransM}(\delta_i, s, \delta_i', m_i)$$

[6] The notion of move-determined program is similar to that of situation-determined program from [7].

[7] Using $CurrProg$ introduced here, this can be specified as follows: $\forall s.\forall \delta_i.\mathit{Executable}(s) \wedge \mathit{CurrProg}(Ag_i, \delta_i, s) \supset \mathit{MoveDet}(\delta_i, s)$.

That is, a move m is legal for agent Ag_i in situation s if her current remaining program δ_i in s can perform m, and when a joint move $tick(m_i, \ldots, m_n)$ is performed, the current remaining program of each agent Ag_i is updated to be what remains of her current program after her move m_i.

Example 7 *For the game of Example 1, we can define the legal moves of Ag_3 using the following program:*

$CurrProg(Ag_3, \delta, S_0) \equiv \delta = BehaviorAg_3$
where $BehaviorAg_3 \doteq$
while $\neg Terminal$ **do**
$([\exists x, y.At(ag, x, y)?; Stay] \mid$
$[AtExit(Ag_3)?; Exit] \mid$
$[\pi d.\exists u, v, x, y.At(Ag_3, u, v, s) \wedge Adj(u, v, d, x, y)?; move(d)])$

For the guard agents Ag_1 and Ag_2, the program is similar, except that the Exit move is not allowed:

while $\neg Terminal$ **do**
$([\exists x, y.At(ag, x, y)?; Stay] \mid$
$[\pi d.\exists u, v, x, y.At(ag, u, v) \wedge Adj(u, v, d, x, y)?; move(d)])$

We can also specify more constrained behaviors/strategies, e.g., one where Ag_3 never moves in a direction where he may be captured:

$BehaviorAg_3 \doteq$
while $\neg Terminal$ **do**
$([\exists x, y.At(Ag_3, x, y)?; Stay] \mid$
$[AtExit(Ag_3)?; Exit] \mid$
$[\pi d.\exists u, v, x, y.At(Ag_3, u, v) \wedge Adj(u, v, d, x, y) \wedge$
$\neg\exists m.Capturing(Ag_3, move(d), Ag_1, m) \wedge$
$\neg\exists m.Capturing(Ag_3, move(d), Ag_2, m)?;$
$move(d)])$

When we do this, we verify properties of the system under the assumption that agents behave as specified. We should of course ensure that all moves allowed by such a specialized behavior are legal and that the agent always has some move it can make until the game ends. The latter is just playability, which we discussed earlier. The former requires establishing a simulation relation between the transition systems induced by the specialized and the orignal program, cf. [28]; we leave this for future work.

The verification techniques of Sec. 5 can be adapted to handle $LegalM$ defined through programs.

Propositional case. The propositional case is straightforward.

Theorem 8 *Let $\mathcal{D}$ be a SCSGS with a finite object domain, with $LegalM$ and $CurrProg$ defined through programs. Then, for every μATL-FO formula Ψ, checking whether $\mathcal{D} \models \Psi$ is decidable.*

Proof (sketch). Since the domain is bounded, the number of possible remaining programs within every computation is finite. Hence every fluent, including $LegalM$ and $CurrProg$ has a finite extension (and hence can be represented propositionally). Thus we can define a finite transition system that is bisimilar to the model of the action theory and check the property Ψ over it. □

Bounded first-order case. For the bounded first-order case, we can leverage on recent work [10]. The difficulty when the domain is infinite is that the number of possible remaining programs within a computation is infinite in general. Moreover their inductive structure guides the definition of *TransM* and *FinalM* and hence ultimately of $LegalM$ and $CurrProg$. The results in [10] however, show that in the absence of recursion, the source of having infinitely many program terms is the pick operator π. Now, when the number of action types is finite (and in our framework, moves (and hence actions) come in finitely many move types), we can capture such programs as a pair formed by a program schema (with pick variables uninstantiated) and a separate set of variable substitutions, one for each pick variable in the original program, which in turn can be assumed to range over objects only w.l.o.g. In this way, the number of remaining program schemas that are generated during a computation is finite (they act as a program counter) while the number of possible substitutions is infinite (they can get all possible values from the infinite object domain). The point is that the number of pick variables in a program is syntactically determined by the original program alone (not the remaining programs that can be generated) and hence is naturally bounded.

As a final result [10] shows that for *situation-determined programs*, execution, as defined by *Trans* and *Final*, can be captured using new suitable predicate fluents, which are bounded. The same kind of reasoning can indeed be applied in our case to show that for *move-determined programs*, *TransM* and *FinalM* can be captured using new predicate fluents, which are bounded. This, in turn, makes the extension of $LegalM$ and $CurrProg$ bounded and hence we can apply Theorem 6 to get decidability. Thus we get:

Theorem 9 *Let $\mathcal{D}$ be a SCSGS that is a bounded action theory except for LegalM and CurrProg defined through programs as above. Then for every μATL-FO formula Ψ, checking whether $\mathcal{D} \models \Psi$ is decidable.*

General first-order case. For the general first-order case, we can only obtain sound (but generally incomplete) methods, e.g., resorting to the techniques in [11] based on program characteristic graphs [5] or compilation techniques such as [12] and in [10] for situation-determined (in our case move-determined) programs.

7 Conclusion

In this paper, we have defined a logical framework, SCSGSs, for representing synchronous games-like systems and verifying temporal properties over them. We have also shown that under some common assumptions, GDL games can be represented as SCSGSs. Perhaps more significantly our framework allows for representing first-order GDL games in standard situation calculus, and thus allows one to leverage on the wide literature on such a formalism for analyzing the game, in particular for FO-temporal verification. Indeed, a key point is that our framework is truly first-order and can be used to specify games/systems that involve infinite domains and an infinite set of states. Further, when the SCSGS is "propositional" or "bounded", verification of large classes of temporal formulas are decidable. Even in the general case, reasoners for our framework can be developed. A prototype verifier that uses the iterated fixpoint approximation technique is discussed in [17]. Such reasoners could be used to verify properties of interest in general game playing. One important assumption in our framework is that the game state is fully observable and no agent can any have private information. We would like to generalize it to accomodate partially observable game settings.

ACKNOWLEDGEMENTS

We acknowledge the support of Sapienza 2015 project "Immersive Cognitive Environments" and the National Science and Engineering Research Council of Canada.

REFERENCES

[1] Rajeev Alur, Thomas A. Henzinger, and Orna Kupferman, 'Alternating-time temporal logic', *J. ACM*, **49**(5), 672–713, (2002).

[2] Julien Bradfield and Colin Stirling, 'Modal mu-calculi', in *Handbook of Modal Logic*, volume 3, 721–756, Elsevier, (2007).

[3] Diego Calvenese, Giuseppe De Giacomo, , Marco Montali, and Fabio Patrizi, 'On first-order μ-calculus over situation calculus action theories', in *Proc. of KR*, (2016).

[4] Diego Calvenese, Giuseppe De Giacomo, Marco Montali, and Fabio Patrizi, 'First-order μ-calculus over generic transition systems and applications to the situation calculus'. Submitted.

[5] Jens Claßen and Gerhard Lakemeyer, 'A logic for non-terminating Golog programs', in *Proc. of KR*, pp. 589–599, (2008).

[6] Giuseppe De Giacomo, Yves Lespérance, and Hector J. Levesque, 'ConGolog, a concurrent programming language based on the situation calculus', *Artificial Intelligence*, **121**(1–2), 109–169, (2000).

[7] Giuseppe De Giacomo, Yves Lespérance, and Christian J. Muise, 'On supervising agents in situation-determined ConGolog', in *Proc. of AAMAS*, pp. 1031–1038, (2012).

[8] Giuseppe De Giacomo, Yves Lespérance, and Fabio Patrizi, 'Bounded situation calculus action theories and decidable verification', in *Proc. of KR*, (2012).

[9] Giuseppe De Giacomo, Yves Lespérance, and Fabio Patrizi, 'Bounded situation calculus action theories', *Artificail Intelligence*, **237**, 172–203, (2016).

[10] Giuseppe De Giacomo, Yves Lespérance, Fabio Patrizi, and Sebastian Sardina, 'Verifying ConGolog programs on bounded situation calculus theories', in *Proc. of AAAI*, (2016).

[11] Giuseppe De Giacomo, Yves Lespérance, and Adrian R Pearce, 'Situation calculus based programs for representing and reasoning about game structures.', in *Proc. of KR*, (2010).

[12] Christian Fritz, Jorge A. Baier, and Sheila A. McIlraith, 'ConGolog, Sin Trans: Compiling ConGolog into basic action theories for planning and beyond', in *Proc. of KR*, pp. 600–610, (2008).

[13] Michael R. Genesereth, Nathaniel Love, and Barney Pell, 'General game playing: Overview of the AAAI competition', *AI Magazine*, **26**(2), 62–72, (2005).

[14] Enrico Giunchiglia, Joohyung Lee, Vladimir Lifschitz, Norman McCain, and Hudson Turner, 'Nonmonotonic causal theories', *Artificial Intelligence*, **153**(1–2), 49–104, (2004).

[15] Babak Bagheri Hariri, Diego Calvanese, Giuseppe De Giacomo, Alin Deutsch, and Marco Montali, 'Verification of relational data-centric dynamic systems with external services', in *Proc. of PODS*, pp. 163–174, (2013).

[16] Andreas Herzig, Emiliano Lorini, Frédéric Moisan, and Nicolas Troquard, 'A dynamic logic of normative systems', in *Proc. of IJCAI*, (2011).

[17] Slawomir Kmiec and Yves Lespérance, 'Infinite states verification in game-theoretic logics: Case studies and implementation', in *Proc. of EMAS*. Springer, (2014).

[18] Hector J. Levesque and Gerhard Lakemeyer, *The Logic of Knowledge Bases*, MIT Press, 2001.

[19] Hector J. Levesque, Ray Reiter, Yves Lespérance, Fangzhen Lin, and Richard B. Scherl, 'GOLOG: A logic programming language for dynamic domains', *J. Logic Programming*, **31**, 59–84, (1997).

[20] John W. Lloyd, *Foundations of Logic Programming, 2nd Edition*, Springer, 1987.

[21] Alessio Lomuscio, Hongyang Qu, and Franco Raimondi, 'MCMAS: A model checker for the verification of multi-agent systems', in *Proc. of CAV*, pp. 682–688, (2009).

[22] Nathaniel Love, Timothy Hinrichs, David Haley, Eric Schkufza, and Michael Genesereth, 'General game playing: Game description language specification'. Tech. Rept. LG-2006-01, Stanford University, 2006.

[23] J. McCarthy and P. J. Hayes, 'Some Philosophical Problems From the Standpoint of Artificial Intelligence', *Machine Intelligence*, **4**, 463–502, (1969).

[24] Fabio Mogavero, Aniello Murano, and Moshe Y. Vardi, 'Reasoning about strategies', in *Proc. of FSTTCS*, pp. 133–144, (2010).

[25] Javier Pinto, 'Concurrent actions and interacting effects', in *Proc. of KR*, pp. 292–303, (1998).

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

[27] Ji Ruan, Wiebe van der Hoek, and Michael Wooldridge, 'Verification of games in the game description language', *J. Log. Comput.*, **19**(6), 1127–1156, (2009).

[28] Sebastian Sardina and Giuseppe De Giacomo, 'Composition of ConGolog programs', in *Proc. of IJCAI*, pp. 904–910, (2009).

[29] Stephan Schiffel and Michael Thielscher, 'A multiagent semantics for the game description language', in *Agents and Artificial Intelligence*, volume 67 of *CCIS*, pp. 44–55. Springer, (2010).

[30] Stephan Schiffel and Michael Thielscher, 'Representing and reasoning about the rules of general games with imperfect information', *J. Artif. Intell. Res. (JAIR)*, **49**, 171–206, (2014).

[31] Alfred Tarski, 'A lattice-theoretical fixpoint theorem and its applications', *Pacific J. of Mathematics*, **5**(2), 285–309, (1955).

[32] Eugenia Ternovskaia, 'Automata theory for reasoning about actions', in *Proc. of IJCAI*, pp. 153–159, (1999).

[33] Michael Thielscher, 'A general game description language for incomplete information games', in *Proc. of AAAI*, (2010).

[34] Michael Thielscher, 'Translating general game descriptions into an action language', in *Logic Programming, Knowledge Representation, and Nonmonotonic Reasoning*, volume 6565 of *LNCS*, pp. 300–314. Springer, (2011).

[35] Wiebe van der Hoek and Michael Wooldridge, 'On the logic of cooperation and propositional control', *Artif. Intell.*, **164**(1-2), 81–119, (2005).