Abstraction of Situation Calculus Concurrent Game Structures

Yves Lespérance[1], Giuseppe De Giacomo[2], Maryam Rostamigiv[3], Shakil M. Khan[3]
[1]York University, Toronto, ON, Canada
[2]University of Oxford, Oxford, UK
[3]University of Regina, Regina, SK, Canada
lesperan@eecs.yorku.ca, giuseppe.degiacomo@cs.ox.ac.uk, maryam.rostamigiv@uregina.ca, shakil.khan@uregina.ca

Abstract

We present a general framework for abstracting agent behavior in multi-agent synchronous games in the situation calculus, which provides a first-order representation of the state and allows us to model how plays depend on the data and objects involved. We represent such games as action theories of a special form called 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. In our framework, one specifies both an abstract SCSGS and a concrete SCSGS, as well as a refinement mapping that specifies how each abstract move is implemented by a Golog program defined over the concrete SCSGS. We define notions of sound and complete abstraction with respect to a mapping over such SCSGS. To express strategic properties on the abstract and concrete games we adopt a first-order variant of alternating-time μ-calculus μATL-FO. We show that we can exploit abstraction in verifying μATL-FO properties of SCSGSs under the assumption that agents can always execute abstract moves to completion even if not fully controlling their outcomes.

Introduction

Many multi-agent applications can be viewed *games* where some agents try to ensure that certain objectives hold no matter how the environment and other agents behave. Logics such as Alternating-Time Temporal Logic (ATL) (Alur, Henzinger, and Kupferman 2002), Coalition Logic (CL) (Pauly 2002), and Strategy Logic (SL) (Chatterjee, Henzinger, and Piterman 2010; Mogavero et al. 2014) have been defined to specify properties of such systems and verify them through techniques like model checking. However, as the game/system becomes more complex, it becomes very important to use *abstraction* to explain how the game evolves and do strategic reasoning more effectively.

(Banihashemi, De Giacomo, and Lespérance 2017) (BDL17) recently proposed a formal account of *agent abstraction* based on the situation calculus (McCarthy and Hayes 1969; Reiter 2001) and the ConGolog agent programming language (De Giacomo, Lespérance, and Levesque 2000). They assume that one has a high-level/abstract action theory, a low-level/concrete action theory, both representing the agent's behavior at different levels of detail, and a *refinement mapping* between the two. The refinement mapping specifies how each high-level action is implemented by a low-level ConGolog program and how each high-level fluent can be translated into a low-level formula. This work defines notions of abstraction between such action theories in terms of the existence of a suitable bisimulation relation (Milner 1971) between their respective models. Such abstractions have many useful properties that ensure that one can reason about the agent's actions (e.g., executability, projection, and planning) at the abstract level, and refine and concretely execute them at the low level. The framework can also be used to generate high-level explanations of low-level behavior. More recently, (Banihashemi, De Giacomo, and Lespérance 2023), building upon (De Giacomo and Lespérance 2021), extended their abstraction framework to agents operating in nondeterministic environments, where the agent does not fully control the outcome of its actions.

In this paper, we further generalize this abstraction framework to apply to *multi-agent synchronous games*. To represent such games, we follow (De Giacomo, Lespérance, and Pearce 2016) and use 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. The SCSGS specifies the legal moves available to each agent in each situation and how fluents change value when a $tick$ joint move by all the agents is performed. To express strategic properties on the abstract and concrete games we adopt a first-order variant of alternating-time μ-calculus μATL-FO.

In our game abstraction framework, one specifies both an abstract SCSGS and a concrete SCSGS, as well as a *refinement mapping* that specifies how each abstract move is implemented by a Golog (Levesque et al. 1997) program defined over the concrete SCSGS.

We show that we can exploit abstraction in verifying μATL-FO properties of SCSGSs under the assumption that agents can always execute abstract moves to completion even if not fully controlling their outcomes. Our framework is based on the situation calculus which provides a first-order representation of the state, allowing us to model how system processes depend on the data and objects involved. Our account is more general than (Banihashemi, De Giacomo, and Lespérance 2023), as we support multiple agents that act synchronously.

While this paper has a foundational nature, we note that our results are relevant for concrete applications where various agents need to be controlled and coordinated in a first-order state setting, such as those in the context of Smart Manufacturing (De Giacomo et al. 2022) and Smart Business Process Management (Marrella, Mecella, and Sardiña 2017).

Preliminaries

Situation Calculus. The *situation calculus* is a well known predicate logic language for representing and reasoning about dynamically changing worlds (McCarthy and Hayes 1969; Reiter 2001). All changes to the world are the result of *actions*, which are terms in the language. A possible world history is represented by a term called a *situation*. The constant S_0 is used to denote the initial situation. Sequences of actions are built using the function symbol do, such that $do(a, s)$ denotes the successor situation resulting from performing action a in situation s. Predicates and 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., $Open(Door1, s)$). The abbreviation $do([a_1, \ldots, a_n], s)$ stands for $do(a_n, do(a_{n-1}, \ldots, do(a_1, s) \ldots))$; for an action sequence $\vec{a}$, we write $do(\vec{a}, s)$ for $do([\vec{a}], s)$. In this language, a dynamic domain can be represented by a *basic action theory (BAT)*, where successor state axioms (SSA) represent the causal laws of the domain and and provide a solution to the frame problem (Reiter 2001). A special predicate $Poss(a, s)$ is used to state that action a is executable in situation s; the precondition axioms characterize this predicate. $Executable(s)$ means that every action done in reaching situation s was executable in the situation in which it occurred.

Synchronous Game Structures. Following (De Giacomo, Lespérance, and Pearce 2016), 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. To represent such multi-player synchronous games, we use a special class of BATs, called *situation calculus synchronous game structures (SCSGSs)*, which are defined as follows.

Agents A SCSGS $\mathcal{D}$ involves a finite set of n agents, and we use 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 use 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 parameterized by objects $\vec{x}$, with $Move(m) \equiv \bigvee_i \exists \vec{x}.m = M_i(\vec{x})$. 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 (parameterized 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. The *legal moves* available to each agent in a given situation are 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):

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

i.e., 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:

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

Thus the joint action by all agents $tick(m_1, \ldots, m_n)$ is executable 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.[1]

Initial situation description. Finally, the initial state of the game is axiomatized in the *initial situation description* $\mathcal{D}_{S_0}$ as usual, in a domain specific way.

High-Level Programs and Golog. To represent and reason about complex actions or processes obtained by suitably executing atomic actions, various so-called *high-level programming languages* have been defined. Here we concentrate on (a variant of) Golog (Levesque et al. 1997) that includes the following constructs:

$$\delta ::= \alpha \mid \varphi? \mid \delta_1; \delta_2 \mid \delta_1|\delta_2 \mid \pi x.\delta \mid \delta^*$$

In the above, α is an action term, possibly with parameters. φ is a situation-suppressed formula, i.e., a formula with all situation arguments in fluents suppressed. As usual, we denote by $\varphi[s]$ the formula obtained from φ by restoring the situation argument s into all fluents in φ. The sequence of program δ_1 followed by program δ_2 is denoted by $\delta_1; \delta_2$. 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 binding for object variable x

[1]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 (Reiter 2001; Pinto 1998).

(observe that such a choice is, in general, unbounded). δ^* performs δ zero or more times.

Formally, the semantics of Golog can be specified in terms of single-step transitions, using two predicates (De Giacomo, Lespérance, and Levesque 2000): *(i)* $Trans(\delta, s, \delta', s')$, which holds if one step of program δ in situation s may lead to situation s' with δ' remaining to be executed; and *(ii)* $Final(\delta, s)$, which holds if program δ may legally terminate in situation s. The definitions of $Trans$ and $Final$ that we use are as in (De Giacomo, Lespérance, and Pearce 2010); differently from (De Giacomo, Lespérance, and Levesque 2000), the test construct $\varphi?$ does not yield any transition, but is final when satisfied. Predicate $Do(\delta, s, s')$ means that program δ, when executed starting in situation s, has as a legal terminating situation s', and is defined as $Do(\delta, s, s') \doteq \exists \delta'.Trans^*(\delta, s, \delta', s') \wedge Final(\delta', s')$ where $Trans^*$ denotes the reflexive transitive closure of $Trans$.

For simplicity in this paper, we use a restricted class of Golog programs which are *situation-determined* (SD) (De Giacomo, Lespérance, and Muise 2012), i.e., for every sequence of actions, the remaining program is uniquely determined by the resulting situation: $SitDet(\delta, s) \doteq \forall s', \delta', \delta''. Trans^*(\delta, s, \delta', s') \wedge Trans^*(\delta, s, \delta'', s') \supset \delta' = \delta''$.

Move-Based Golog. In this paper, we will use "standard" Golog programs to specify tasks for the system or a coalition of agents in it; we call these *system programs*. But we will also use a special kind of Golog programs defined in (De Giacomo, Lespérance, and Pearce 2016) that we call *move-based* programs to specify the legal moves of an agent in a SCSGS as well as how high-level moves are implemented at the low level. Move-based programs are Golog programs where atomic actions are replaced by atomic moves. Apart from this change the syntax remains the same.

To define the semantics of such programs, they introduce the predicate $TransM(\delta, s, \delta', m)$ to mean that the program δ in situation s can perform move m leaving δ' as the remaining program to execute, and the predicate $FinalM(\delta, s)$ to mean that program δ can be considered terminated in situation s.

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

$$\begin{aligned}&LegalM(Ag_i, m, s) \doteq \\ &\qquad CurrProg(Ag_i, \delta_i, s) \wedge \exists \delta_i'. TransM(\delta_i, s, \delta_i', m)\end{aligned}$$

where the successor state axiom for $CurrProg$ is:[2]

$$\begin{aligned}&CurrProg(Ag_i, \delta_i', do(tick(m_1, \ldots, m_n), s)) \equiv \\ &\qquad CurrProg(Ag_i, \delta_i, s) \wedge TransM(\delta_i, s, \delta_i', m_i)\end{aligned}$$

That is, a move m_j is legal for agent Ag_j in situation s if her current remaining program δ_j in s can perform m_j, and when a joint move $tick(m_1, \ldots, m_n)$ 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. We use $\mathcal{C}$ to denote the axioms defining the Golog language, including $TransM$ and $FinalM$ for move-based programs.

[2]Here, we assume that move-based programs are *move determined*: $MoveDet(\delta, s) \doteq \forall m, \delta', \delta''. TransM(\delta, s, \delta', m) \wedge TransM(\delta, s, \delta'', m) \supset \delta' = \delta''$. Each agent's program should remain move determined in every game situation: $\forall s.\delta_i. Executable(s) \wedge CurrProg(Ag_i, \delta_i, s) \supset MoveDet(\delta_i, s)$.

Abstraction of SCSGSs

In this section, we show how we can extend the agent abstraction framework of (BDL17) to handle SCSGSs. As in (BDL17), we assume that there is a high-level/abstract (HL) action theory $\mathcal{D}_h$ and a low-level/concrete (LL) action theory $\mathcal{D}_l$ representing the agent's possible behaviors at different levels of detail. In (BDL17), these are standard BATs; here, we assume that they are both SCSGSs. $\mathcal{D}_h$ (resp. $\mathcal{D}_l$) involves a finite set of agents $Agents$, a finite set of move types $Moves_h$ (resp. $Moves_l$), a set of primitive actions $A_h = \{tick_h\}$ (resp. $A_l = \{tick_l\}$) , and a finite set of primitive fluent predicates $\mathcal{F}_h$ (resp. $\mathcal{F}_l$). The terms of object sort are assumed be a countably infinite set $\mathcal{N}$ of standard names for which we have the unique name assumption and domain closure. Also, $\mathcal{D}_h$ and $\mathcal{D}_l$ are assumed to share no domain specific symbols except for the set of standard names for objects $\mathcal{N}$. For simplicity and w.l.o.g., it is assumed that there are no functions other than constants and no non-fluent predicates.

Refinement Mapping. In (BDL17), one relates the HL and LL BATs by defining a *refinement mapping* that specifies how HL atomic actions are implemented at the LL and how HL fluents can be translated into LL state formulas. In a BAT, one can simply map each HL atomic action type to a LL program that the agent uses to implement the action. In a SCSGS however, we need to specify how each HL move is implemented at the LL. Thus we say that a *SCSGS refinement mapping* m is a triple $\langle m_m, m_a, m_f \rangle$ where m_m associates each HL move type m in $Moves_h$ to a move-determined (MD) move-based Golog program δ_m defined over the LL SCSGS theory that implements the move, i.e., $m_m(m(\vec{x})) = \delta_m(\vec{x})$, m_a maps the unique HL action $tick_h \in A_h$ to a Golog *system program* that executes the mapping of the moves involved synchronously in parallel, i.e., $m_a(tick(m_1, \ldots, m_n)) = sync(m_m(m_1), \ldots, m_m(m_n))$, and (as in (BDL17)) m_f maps each situation-suppressed HL fluent $F(\vec{x})$ in $\mathcal{F}_h$ to a situation-suppressed formula $\phi_F(\vec{x})$ defined over the LL theory that characterizes the concrete conditions under which $F(\vec{x})$ holds in a situation, i.e., $m_f(F(\vec{x})) = \phi_F(\vec{x})$.

We support the synchronous concurrency construct by extending the Golog semantics as follows:

$$\begin{aligned}&Trans(sync(\delta_{m_1}, \ldots, \delta_{m_n}), s, \delta', s') \equiv \\ &\exists \delta_{m_1}', \ldots \delta_{m_n}'. \exists m_1, \ldots, m_n. \delta' = sync(\delta_{m_1}', \ldots, \delta_{m_n}') \wedge \\ &TransM(\delta_{m_1}, s, \delta_{m_1}', m_1) \wedge \ldots \wedge TransM(\delta_{m_n}, s, \delta_{m_n}', m_n) \\ &\wedge s' = do(tick(m_1, \ldots, m_n), s)\end{aligned}$$

$$\begin{aligned}&Final(sync(\delta_{m_1}, \ldots, \delta_{m_n}), s) \equiv \\ &\quad FinalM(\delta_{m_1}, s) \wedge \ldots \wedge FinalM(\delta_{m_n}, s)\end{aligned}$$

We can extend a mapping to a sequence of system actions in the obvious way, i.e., $m_a(\alpha_1, \ldots, \alpha_n) \doteq$

$m_a(\alpha_1); \ldots ; m_a(\alpha_n)$ for $n \geq 1$ and $m_a(\epsilon) \doteq nil$. We also extend the notation so that $m_f(\phi)$ stands for the result of substituting every fluent $F(\vec{x})$ in situation-suppressed formula ϕ by $m_f(F(\vec{x}))$.

We also need to ensure that the mapping captures all the legal behaviors that agents can display at the LL. This is essential if we want to do strategic reasoning at the HL and be able to refine the resulting strategies into LL ones that achieve the objectives. This can be done analogously to how BATs for nondeterministic domains (NDBATs) are mapped in (Banihashemi, De Giacomo, and Lespérance 2023). But there is a new complication here. We need to ensure that at the LL, all agents start their HL moves at the same time and end them at the same time, otherwise ensuring bisimilarity with respect to (wrt) the mapping becomes very difficult. To get this, we impose the following constraint:

Constraint 1 *(Proper Refinement Mapping)*
For every high-level system action sequence $\vec{\alpha}$ and every agent $i \in Agents$, we have that:

$$\begin{array}{l}
\mathcal{D}_l \cup \mathcal{C} \models \forall s.(Do(m_a(\vec{\alpha}), S_0, s) \supset \\
\quad \forall m_i, s'.(Do(sync((\pi \vec{m}.\vec{m})^*, m_m(m_i), (\pi \vec{m}.\vec{m})^*), s, s') \supset \\
\quad\quad \exists m_1, \ldots, m_{i-1}, m_{i+1}, \ldots, m_n. \\
\quad\quad\quad Do(m_a(tick(m_1, \ldots, m_{i-1}, m_i, m_{i+1}, \ldots, m_n)), s, s')))
\end{array}$$

This ensures that for every situation s' that can be reached by an agent i executing a refinement of a HL move m_i with other agents executing arbitrary legal LL moves, there exist HL legal moves by the other agents that make the system reach s'. To guarantee this, we have to ensure that at the LL, agents only perform moves that refine HL moves, and that all agents begin and end refinements of HL moves at the same time. We discuss below how we can specify the legal moves to respect this. We say that a SCSGS refinement mapping m is *proper with respect to low-level SCSGS $\mathcal{D}_l$* if this constraint holds.

m-Bisimulation. To relate the HL and LL models/theories, (BDL17) define a variant of bisimulation (Milner 1971, 1989). Let M_h be a model of the HL theory $\mathcal{D}_h$, and M_l a model of the LL theory $\mathcal{D}_l \cup \mathcal{C}$. We say that situation s_h in M_h is *m-isomorphic* to situation s_l in M_l, written $s_h \simeq_m^{M_h,M_l} s_l$, if and only if

> $M_h, v[s/s_h] \models F(\vec{x}, s)$ iff $M_l, v[s/s_l] \models m(F(\vec{x}))[s]$ for every high-level primitive fluent $F(\vec{x})$ in $\mathcal{F}_h$ and every variable assignment v ($v[x/e]$ stands for the assignment that is like v except that x is mapped to e).

A relation $B \subseteq \Delta_S^{M_h} \times \Delta_S^{M_l}$ (where Δ_S^M stands for the situation domain of M) is an *m-bisimulation relation between M_h and M_l* if $\langle s_h, s_l \rangle \in B$ implies that:

1. $s_h \simeq_m^{M_h,M_l} s_l$, i.e., s_h and s_l evaluate HL fluents the same;
2. for every HL primitive action type A in A_h, if there exists s'_h such that $M_h, v[s/s_h, s'/s'_h] \models Poss(A(\vec{x}), s) \wedge s' = do(A(\vec{x}), s)$, then there exists s'_l such that $M_l, v[s/s_l, s'/s'_l] \models Do(m_a(A(\vec{x})), s, s')$ and $\langle s'_h, s'_l \rangle \in B$; and
3. for every HL primitive action type A in A_h, if there exists s'_l such that $M_l, v[s/s_l, s'/s'_l] \models Do(m_a(A(\vec{x})), s, s')$, then there exists s'_h such that $M_h, v[s/s_h, s'/s'_h] \models Poss(A(\vec{x}), s) \wedge s' = do(A(\vec{x}), s)$ and $\langle s'_h, s'_l \rangle \in B$.

M_h is *m-bisimilar* to M_l, written $M_h \sim_m M_l$, if and only if there exists an m-bisimulation relation B between M_h and M_l such that $(S_0^{M_h}, S_0^{M_l}) \in B$. A situation s_h in M_h is *m-bisimilar* to situation s_l in M_l, written $s_h \sim_m^{M_h,M_l} s_l$, if and only if there exists an m-bisimulation relation B between M_h and M_l and $(s_h, s_l) \in B$.

The definition of m-bisimulation for SCSGSs can remain as in (BDL17) where conditions (ii) and (iii) are applied to the high-level primitive action $tick_h$ and its mapping m_a. The definition of m-bisimulation ensures that performing a HL $tick$ action results in m-bisimilar situations.

(BDL17) use m-bisimulation to define notions of sound/complete abstraction. $\mathcal{D}_h$ is a *sound abstraction of $\mathcal{D}_l$ relative to refinement mapping m* if and only if, for all models M_l of $\mathcal{D}_l \cup \mathcal{C}$, there exists a model M_h of $\mathcal{D}_h$ such that $M_h \sim_m M_l$. With a sound abstraction, whenever the HL theory *entails* that a sequence of actions is executable and achieves a certain condition, then the LL must also entail that there exists an executable refinement of the sequence such that the mapped condition holds afterwards. Moreover, whenever the LL takes the executability of a refinement of a HL action to be satisfiable, then the HL does as well. A dual notion is also defined: $\mathcal{D}_h$ is a *complete abstraction of $\mathcal{D}_l$ relative to refinement mapping m* if and only if, for all models M_h of $\mathcal{D}_h$, there exists a model M_l of $\mathcal{D}_l \cup \mathcal{C}$ such that $M_l \sim_m M_h$.

Example. Our running example involves a repair shop where a set of agents collaborate to repair items and ship them to customers. The set of agents consists of two repair robots RR_1 and RR_2, a dispatcher $Disp$ that assigns items to either of these, a shipper Sh, and an agent representing the customers $Cust$. We assume that items arrive at the shop, are assigned to one of the repair robots by the dispatcher, then are repaired, and finally are shipped by the shipper.

The HL SCSGS $\mathcal{D}_h^{rs}$ has the following $LegalM$ axioms:

$$\begin{array}{l}
LegalM(ag, arrive(i), s) \doteq ag = Cust \wedge \neg Arrived(i, s) \\
LegalM(ag, assign(i, ag'), s) \doteq \\
\quad ag = Disp \wedge (ag' = RR_1 \vee ag' = RR_2) \wedge \\
\quad Arrived(i, s) \wedge \neg \exists ag''.Assigned(i, ag'', s) \\
LegalM(ag, repair(i), s) \doteq (ag = RR_1 \vee ag = RR_2) \wedge \\
\quad Assigned(i, ag, s) \wedge \neg Repaired(i, s) \\
LegalM(ag, ship(i), s) \doteq \\
\quad ag = Sh \wedge Repaired(i, s) \wedge \neg Shipped(i, s) \\
LegalM(ag, wait, s) \doteq True
\end{array}$$

$\mathcal{D}_h^{rs}$ also includes the following SSAs (we use the shorthand notation $m \in a$ to mean that move m is one of the moves in the joint move $a = tick(m_1, \ldots, m_n)$):

$$\begin{array}{l}
Arrived(i, do(a, s)) \equiv arrive(i) \in a \vee Arrived(i, s) \\
Assigned(i, ag, do(a, s)) \equiv \\
\quad assign(i, ag) \in a \vee Assigned(i, ag, s) \\
Repaired(i, do(a, s)) \equiv repair(i) \in a \vee Repaired(i, s) \\
Shipped(i, do(a, s)) \equiv ship(i) \in a \vee Shipped(i, s)
\end{array}$$

$\mathcal{D}_h^{rs}$ also contains the following initial state axioms:

$$\forall i.Arrived(i, S_0) \equiv i = 1 \qquad \forall i, ag.\neg Assigned(i, ag, S_0)$$
$$\forall i.\neg Repaired(i, S_0) \qquad \forall i.\neg Shipped(i, S_0)$$

Before presenting the LL theory, let us define the refinement mapping m^{rs} (we use subscripts to disambiguate HL and LL moves with the same name):

$m^{rs}(arrive_{HL}(i)) = \neg Arrived_{LL}(i)?; arrive_{LL}(i); wait_{LL}$
$m^{rs}(assign_{HL}(i, ag)) = ((ag = RR_1 \vee ag = RR_2)$
$\quad \wedge Arrived_{LL}(i) \wedge \neg\exists ag'.Assigned_{LL}(i, ag'))?;$
$\quad assign_{LL}(i, ag); wait_{LL}$
$m^{rs}(repair(i)) = (\exists ag.Assigned_{LL}(i, ag) \wedge$
$\quad \neg(Diagnosed(i) \wedge Fixed(i)))?; diagnose(i); fix(i)$
$m^{rs}(ship(i)) = (Diagnosed(i) \wedge Fixed(i)$
$\quad \wedge\neg(Packed(i) \wedge DroppedOff(i)))?; pack(i); dropOff(i)$
$m^{rs}(wait_{HL}) = wait_{LL}; wait_{LL}$
$m^{rs}(Arrived_{HL}(i)) = Arrived_{LL}(i)$
$m^{rs}(Assigned_{HL}(i, ag)) = Assigned_{LL}(i, ag)$
$m^{rs}(Repaired(i)) = Diagnosed(i) \wedge Fixed(i)$
$m^{rs}(Shipped(i)) = Packed(i) \wedge DroppedOff(i)$

We specify agents' LL legal moves procedurally using the move-based Golog language discussed earlier. Thus, in the LL SCSGS $\mathcal{D}_l^{rs}$, we have the following initial state axioms specifying the agents' initial move-based programs:

$CurrProg(Cust, \delta_i, S_0) \equiv$
$\delta_i = ((\pi i.m^{rs}(arrive_{HL}(i)))|m^{rs}(wait))^*$
$CurrProg(Disp, \delta_i, S_0) \equiv$
$\delta_i = ((\pi i, ag.m^{rs}(assign_{HL}(i, ag)))|m^{rs}(wait))^*$
$CurrProg(ag, \delta_i, S_0) \equiv (ag = RR_1 \vee ag = RR_2) \wedge$
$\delta_i = ((\pi i.m^{rs}(repair(i)))|m^{rs}(wait))^*$
$CurrProg(Sh, \delta_i, S_0) \equiv \delta = ((\pi i.m^{rs}(ship(i)))|m^{rs}(wait))^*$

This says that agents can perform any sequence of refinements of their HL moves as specified by the mapping.

We also have the standard legal move axiom for procedural move-based Golog legal move specifications, as well as the standard SSA for $CurrentProg$, as discussed earlier. $\mathcal{D}_l^{rs}$ also includes the following SSAs:

$Arrived(i, do(a, s)) \equiv arrive(i) \in a \vee Arrived(i, s)$
$Assigned(i, ag, do(a, s)) \equiv$
$\quad assign(i, ag) \in a \vee Assigned(i, ag, s)$
$Diagnosed(i, do(a, s)) \equiv diagnose(i) \in a \vee Diagnosed(i, s)$
$Fixed(i, do(a, s)) \equiv fix(i) \in a \vee Fixed(i, s)$
$Packed(i, do(a, s)) \equiv pack(i) \in a \vee Packed(i, s)$
$DroppedOff(i, do(a, s)) \equiv dropOff(i) \in a \vee DroppedOff(i, s)$

$\mathcal{D}_l^{rs}$ also contains the following initial state axioms:

$$\forall i.Arrived(i, S_0) \equiv i = 1 \quad \forall i, ag.\neg Assigned(i, ag, S_0)$$
$$\forall i.\neg Diagnosed(i, S_0) \quad \forall i.\neg Fixed(i, S_0)$$
$$\forall i.\neg Packed(i, S_0) \quad \forall i.\neg DroppedOff(iS_0)$$

In general, to ensure that a LL SCSGS satisfies Constraint 1, we need to make sure that agents begin and end refinements of HL moves at the same time. A simple way to ensure this is to have all HL moves perform the same number of LL moves, as we did above. But a more flexible way to do this is to add synchronization moves at the LL.[3] As well as ensuring that refinements of HL moves begin and end at the same time, we must make sure that the LL move sequences produced match those allowed by the HL move mapping. If we specify agents' legal moves at the LL procedurally using Golog programs as above, this is easy to do. If instead we want to provide a declarative specification, then we need to tailor the LL legal move conditions to ensure that moves can only occur as allowed by the mapping.

We can show that:

Proposition 2 *SCSGS $\mathcal{D}_h^{rs}$ is a sound and complete abstraction of SCSGS $\mathcal{D}_l^{rs}$ wrt refinement mapping m^{rs}.*

Proof Sketch We use Theorems 9 and 12 from (BDL17) which identify a number of properties that must be entailed by the low-level theory to have sound and complete abstraction wrt a mapping. First, we must show that after any refinement of a HL action sequence, a refinement of a HL action is executable if and only if the mapped precondition of the HL action holds. Second, we must show that after any refinement of a HL action sequence, the mapped HL successor state axioms hold over any refinement of a HL action. Finally, we must show that the initial situations are m-isomorphic (as we have complete information about the initial state here). $\square$

Proposition 3 *SCSGS refinement mapping m^{rs} is proper wrt SCSGS $\mathcal{D}_l^{rs}$.*

Proof Sketch We prove this by induction on the length of $\vec{\alpha}$. $\mathcal{D}_l^{rs}$ uses a procedural specification of legal moves, so the proof is easy: the only legal moves at the LL are those that are produced by a sequence of refinements of the agent's HL moves. $\square$

Notice that our framework supports synchronous moves by the agents, not just turn-based games. In our example, we can have a joint move where a repair robot repairs an item at the same time as the shipper agent ships another item. Furthermore, the effects of a joint move can depend on the moves of several agents and their interaction, e.g., we can represent a domain where two robots are able to lift a heavy object only if they both make a lift move synchronously.

Abstraction in Verifying Strategic Properties

μATL-FO. To express properties about SCSGSs, (De Giacomo, Lespérance, and Pearce 2016) introduces the logic μATL-FO, inspired by alternating-time μ-calculus, μATL, a well-known generalization of ATL (Alur, Henzinger, and

[3]We can do this by introducing a new fluent $DoneHLmove(agt, s)$ and new moves $setDoneHLmove(agt)$ and $unsetDoneHLmove(agt)$ that toggle it. We then modify the implementation of moves to ensure that after completing a refinement of their HL move, agents first do $setDoneHLmove(self)$ and then repeatedly do a $wait$ move until all agents have set their $DoneHLmove$; also, before starting a refinement of a HL move, all agents must do $unsetDoneHLmove(self)$.

Kupferman 2002). This logic is a first-order variant of the μ-calculus (Bradfield and Stirling 2007) that works on games, by suitably considering coalitions acting towards the realization of a temporally extended goal, as in μATL.

We have the following syntax for μATL-FO formulas:

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

In the above, φ is an arbitrary, possibly open, situation-suppressed situation calculus uniform formula and Z is a predicate variable of a given arity. $\langle\langle G\rangle\rangle \bigcirc \Psi$ means that coalition G can force Ψ to hold next, i.e., there is a vector of legal moves for the agents in G such that for all legal moves by the other agents, Ψ holds afterwards. $\mu Z.\Psi(Z)$ 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.

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 $\mu Z.\ \varphi(\vec{x}) \vee \langle\langle G\rangle\rangle \bigcirc Z$; in a first-order ATL, this could be expressed as $\langle\langle G\rangle\rangle \Diamond \varphi(\vec{x})$. Other ATL constructs such as G can force always $\varphi(\vec{x})$, $\langle\langle G\rangle\rangle \Box \varphi(\vec{x})$, and G can force $\varphi(\vec{x})$ to hold until $\varphi'(\vec{y})$, $\langle\langle G\rangle\rangle \varphi(\vec{x})\, \mathcal{U}\, \varphi'(\vec{y})$, can also be expressed.

The formal semantics of μATL-FO is based on characterizing how to evaluate μATL-FO formulas in a situation calculus model 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 M, v, and V an *extension function* $(\cdot)^M_{v,V}$, which maps formulas to subsets of $\mathcal{S}$, and is defined inductively as follows:

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

$$\begin{array}{l}
\text{where } Pre(G,\mathcal{E}) = \{s \in \mathcal{S} \mid \\
\exists m_1,...,m_k. \bigwedge_{g_i \in \{g_1,...,g_k\}=G} (M \models LegalM(g_i, m_i, s)) \wedge \\
\exists m_{k+1},...,m_n. \bigwedge_{g_j \in \{g_{k+1},...,g_n\}=\overline{G}} (M \models LegalM(g_j, m_j, s)) \\
\wedge\ \forall m_{k+1},...,m_n. \\
\quad \bigwedge_{g_j \in \{g_{k+1},...,g_n\}=\overline{G}} (M \models LegalM(g_j, m_j, s)) \\
\quad \supset do(tick(m_1,...,m_n), s) \in \mathcal{E}\}
\end{array}$$

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 $\mathcal{E}$, and similarly for v. Notice also that when a μATL-FO formula Ψ is closed (w.r.t. individual and predicate variables), its extension $(\Psi)^M_{v,V}$ does not depend on the valuations v and V, and we denote the extension of Ψ simply by $(\Psi)^M$. We say that a closed formula Ψ holds in the situation calculus model M, denoted by $M \models \Psi$, if $S_0 \in (\Psi)^M$.

Strategic Ability to Execute Programs. Adapting (De Giacomo and Lespérance 2021), we define an agent strategy as a function from situations to (instantiated) agent moves. That is, $f(s) = M(\vec{t})$ denotes that the strategy f applied to situation s returns $M(\vec{t})$ as the agent's next move. The special agent move *stop* (with no effects and always legal) may be returned when the strategy wishes to stop. Given an agent strategy f_i for every agent i in a coalition C, we denote the joint strategy where every agent i in C follows agent strategy f_i by $\vec{f_C}$, where $\vec{f_C}(i) = f_i$ for all $i \in C$.

To represent the ability of a coalition C to execute a system program/task δ in a SCSGS domain, we write $CanForceBy(C, \delta, s, \vec{f_C})$, which can be viewed as an adversarial version of Do in presence of agents outside the coalition. This predicate states that joint strategy $\vec{f_C}$ executes SD Golog system program δ in situation s considering its nondeterminism angelic, as in the standard Do, but also considering the nondeterminism of agents outside the coalition devilish/adversarial:

$$\begin{array}{l}
CanForceBy(C, \delta, s, \vec{f_C}) \doteq \forall P.[\ldots \supset P(\delta, s)] \\
\ \text{where} \ldots \text{stands for} \\
[((\forall i \in C. f_i(s) = stop) \wedge Final(\delta, s)) \supset P(\delta, s)] \wedge \\
[\forall i \in C. \exists m_i. (f_i(s) = m_i \neq stop \wedge \forall j \in \overline{C}. \\
(\exists m_j. Poss(tick(\vec{m_i}\vec{m_j}), s)) \wedge \\
\forall m_j. Poss(tick(\vec{m_i}\vec{m_j}), s) \supset \\
\quad \exists \delta'. Trans(\delta, s, \delta', do(tick(\vec{m_i}\vec{m_j}), s)) \wedge \\
\qquad P(\delta', do(tick(\vec{m_i}\vec{m_j}), s))) \\
\supset P(\delta, s)]
\end{array}$$

Note that the coalition's joint move $\vec{m_i}$ selected by its strategy must successfully complete the task for every legal joint move $\vec{m_j}$ by agents outside the coalition. We also say that predicate $CanForce(C, \delta, s)$ holds if and only if there exists a joint strategy $\vec{f_C}$ for agents in coalition C such that $CanForceBy(C, \delta, s, \vec{f_c})$ holds.

Strategic reasoning in executing refinements of HL moves. To exploit abstraction in verifying strategic properties, we first need to consider how much strategic reasoning an agent (resp. a coalition) needs to do to execute a HL atomic move (resp. joint move) at the LL. A given HL agent move $M(\vec{x})$ is mapped to a LL agent program $m_m(M(\vec{x}))$ that implements it. In m-bisimilar models, if a coalition C has a vector of legal moves $\vec{m}_C$ and the agents outside the coalition also have legal moves $\vec{m}_{\overline{C}}$, then there exists a terminating execution of $m_a(tick(\vec{m}_C\vec{m}_{\overline{C}}))$ at the LL for any vector of legal moves $\vec{m}_{\overline{C}}$ by agents outside the coalition. But this does not mean that all executions of $m_m(\vec{m}_C)$ terminate when executed together with legal moves by the agents outside the coalition $m_m(\vec{m}_{\overline{C}})$, as some executions may block or diverge, due to choices of the agents in the coalition or that of agents outside it. In general, the agents in the coalition must do strategic reasoning to ensure that the execution of $m_m(\vec{m}_C)$ terminates (and the agents outside the coalition may need to cooperate as well). But we can impose further constraints on the mapping of HL moves to avoid this or ensure that an execution strategy exists. Note that ensuring that the execution of $m_m(\vec{m}_C)$ terminates *does not mean that the agents in C control the joint move's outcome;* the outcome still depends on the moves that the agents outside the coalition select.

One simple approach to ensure that we can refine HL strategies into LL ones is to constrain the mapping of HL moves to ensure that the implementation program always terminates and no LL strategic reasoning is required to ensure termination. But here we follow a less restrictive approach that requires that for any HL move by an agent that is possibly executable at the LL, the agent has a strategy to execute it to termination no matter how the other agents act (even if not controlling the outcome). Formally:

Constraint 4 *(Agents Can Always Execute HL Moves) For every high-level system action sequence $\vec{\alpha}$ and every agent $i \in Agents$, we have that:*

$$\begin{aligned}&\mathcal{D}_l \cup \mathcal{C} \models \forall s.(Do(m_a(\vec{\alpha}), S_0, s) \supset\\ &\quad \forall m_i.(\exists s'.Do(sync((\pi \vec{m}.m)^*, m_m(m_i), (\pi \vec{m}.m)^*), s, s') \supset\\ &\qquad CanForce(\{i\}, sync((\pi \vec{m}.m)^*, m_m(m_i), (\pi \vec{m}.m)^*), s)))\end{aligned}$$

Given this constraint, it follows that for any HL joint action by a non-empty coalition that is possibly executable at the LL, the coalition has a strategy to execute it to termination no matter how the agents outside the coalition behave.

Proposition 5 *Constraint 4 holds for SCSGS $\mathcal{D}_l^{rs}$ and refinement mapping m^{rs}.*

Proof Sketch We prove this by induction on the length of $\vec{\alpha}$. According to $\mathcal{D}_l^{rs}$, the only legal moves at the LL are those that are produced by a sequence of refinements of the agent's HL moves. We can show that the implementations of the HL moves that can be executed synchronously by the agents never interfere with each other. The only non-determinism in the implementation of the HL moves themselves is the initial choice of item (as well as the agent for $assign(i, agt')$); when the initial test is satisfied, the body can always be executed successfully. □

Using abstraction in Verifying μATL-FO Properties. We would like to show that if we have m-bisimilar models and a μATL-FO property Ψ holds in some situation s_h at the HL, then at the LL the mapped version of Ψ holds in any situation s_l that is m-bisimilar to s_h. The first question we face is how to map μATL-FO formulas, in particular $\langle\langle G\rangle\rangle \bigcirc \Psi$. Since HL moves are mapped to programs, if a condition Ψ holds at the next instant at the HL, then the mapped condition will hold not at the next LL instant but in the one where the refinements of the HL moves have finished executing, at the next LL state where a HL joint move has been completed.

First, following (BDL17), we will impose a constraint on the refinements of high-level joint moves/actions:

Constraint 6 *For any distinct ground high-level system action terms α and α' we have that:*

(a) $\mathcal{D}_l \cup \mathcal{C} \models \forall s, s'.Do(m_a(\alpha), s, s') \supset \neg\exists\delta.Trans^*(m_a(\alpha'), s, \delta, s')$

(b) $\mathcal{D}_l \cup \mathcal{C} \models \forall s, s'.Do(m_a(\alpha), s, s') \supset \neg\exists a\exists\delta.Trans^*(m_a(\alpha), s, \delta, do(a, s'))$

(c) $\mathcal{D}_l \cup \mathcal{C} \models \forall s, s'.Do(m_a(\alpha), s, s') \supset s < s'$

Part (a) ensures that different HL primitive system actions have disjoint sets of refinements; (b) ensures that once a refinement of a HL primitive system action is complete, it cannot be extended further; and (c) ensures that a refinement of a HL primitive system action will produce at least one LL action. As shown in (BLD17), these restrictions ensure that we can map a LL system action sequence back to a unique HL system action sequence that produced it.

Proposition 7 *Constraint 6 holds for SCSGS $\mathcal{D}_l^{rs}$ and refinement mapping m^{rs}.*

Proof Sketch The result follows easily from the fact that the HL moves are mapped to programs that each contains a distinctive LL move. Also, they all perform exactly two LL moves. □

We will also assume that the LL model/theory tracks when refinements of HL actions start and end through a state formula $Hlc(s)$, which means that a HL action sequence has just completed in situation s:

Constraint 8 *$\mathcal{D}_l \cup \mathcal{C} \models Hlc(s)$ if and only if there exists a HL system action sequence $\vec{\alpha}$ such that $\mathcal{D}_l \cup \mathcal{C} \models Do(\vec{\alpha}, S_0, s)$.*

Hlc can be defined in various ways. If all refinements of HL actions have the same number of LL actions k, we can add a fluent ctr that counts how many LL actions have occurred since that last HL action completed and define $Hlc(s) \doteq ctr \ mod \ k = 0$. If we add synchronization moves and fluents at the low level as discussed in the previous section, we can use them to define Hlc (e.g., $Hlc(s) \doteq \forall agt.DoneHLmove(agt, s)$).

Proposition 9 *Constraint 8 holds for SCSGS $\mathcal{D}_l^{rs}$ and refinement mapping m^{rs}.*

Proof Sketch We can introduce a counter fluent ctr and define $Hlc(s) \doteq ctr \ mod \ 2 = 0$ as discussed above. The result can then be shown easily by induction on the length of $\vec{\alpha}$. □

Using this, we can introduce an abbreviation $\langle\langle G\rangle\rangle \bigcirc_h \Psi$, meaning that coalition G can force Ψ to hold next after the execution of one HL action:

$$\begin{aligned}\langle\langle G\rangle\rangle \bigcirc_h \Psi &\doteq \langle\langle G\rangle\rangle \bigcirc (\neg Hlc\,\mathcal{U}\,(Hlc \wedge \Psi))\\ &\doteq \langle\langle G\rangle\rangle \bigcirc \mu Z.((Hlc \wedge \Psi) \vee (\neg Hlc \wedge \langle\langle G\rangle\rangle \bigcirc Z))\end{aligned}$$

Using this, we extend the mapping to μATL-FO formulas:

$$\begin{array}{lcl} m_l(\varphi) & \doteq & m_f(\varphi)\\ m_l(\neg\Psi) & \doteq & \neg m_l(\Psi)\\ m_l(\Psi_1 \wedge \Psi_2) & \doteq & m_l(\Psi_1) \wedge m_l(\Psi_2)\\ m_l(\exists x.\Psi) & \doteq & \exists x.m_l(\Psi)\\ m_l(\langle\langle G\rangle\rangle \bigcirc \Psi) & \doteq & \langle\langle G\rangle\rangle \bigcirc_h m_l(\Psi)\\ m_l(Z(\vec{t})) & \doteq & Z(\vec{t})\\ m_l(\mu Z.\Psi) & \doteq & \mu Z.m_l(\Psi)\end{array}$$

We can now show that we can exploit abstraction in verifying μATL-FO properties. First, we will show this for the sublanguage of μATL-FO without predicate variables and the μ operator, call this ATL-FO:

Lemma 10
For any ATL-FO *formula Ψ, if $M_h \sim_m M_l$, Constraints 4, 6, and 8 hold, $s_h \sim_m^{M_h,M_l} s_l$, and $s_h \in (\Psi)_v^{M_h}$, then $s_l \in (m_l(\Psi))_v^{M_l}$.*

Proof By induction on the structure of Ψ.

Base case: If Ψ is a situation-suppressed situation calculus formula, the result follows immediately by Lemma 1 in (BDL17).

Inductive step: If Ψ is $\neg\Psi'$, $\Psi_1 \wedge \Psi_2$, or $\exists x.\Psi$, the result follows by the induction hypothesis and the definition of the extension function.

If Ψ is $\langle\langle G\rangle\rangle \bigcirc \Psi'$, then at the high level, by the definition of of the extension function, we have that there exists a HL joint move $\vec{m}_C$ for agents in the coalition such that there exists a joint move $\vec{m}_{\overline{C}}$ by agents outside the coalition such that $M_h, v[s/s_h] \models Poss(tick(\vec{m}_C\vec{m}_{\overline{C}}), s)$, and for all joint moves $\vec{m}_{\overline{C}}$ by agents outside the coalition, if $M_h, v[s/s_h] \models Poss(tick(\vec{m}_C\vec{m}_{\overline{C}}), s)$ then $do(tick(\vec{m}_C\vec{m}_{\overline{C}}), s) \in (\Psi')_{v,V}^{M_h}$.

Since Constraint 4 holds, the coalition always knows how to execute joint high-level moves to completion at the low level, and thus there exists a low-level strategy $\vec{g_l}$ for the coalition such that $M_l, v[s/s_l] \models CanForce(C, sync(m_m(\vec{m_C}), (\pi \vec{m}.m)^*), s, \vec{g_l})$.

Take an arbitrary s_l' such that $M_l, v[s/s_l, s'/s_l'] \models Do(sync(\vec{g_l}, (\pi \vec{m}.m)^*), s, s')$. By Constraint 1, we have that there exist some vector of HL moves for the agents outside the coalition $\vec{m}_{\overline{C}}$ such that $M_l, v[s/s_l, s'/s_l'] \models Do(m_a(tick(\vec{m}_C\vec{m}_{\overline{C}})), s, s')$. Since $s_h \sim_m^{M_h,M_l} s_l$, it then follows that there exists s_h' such that $M_h, v[s/s_h, s'/s_h'] \models Do(tick((\vec{m}_C\vec{m}_{\overline{C}}), s, s')$ and $s_h' \sim_m^{M_h,M_l} s_l'$. Then by the induction hypothesis, we have that $s_l' \in (m_l(\Psi'))_{v,V}^{M_l}$. By Constraint 6, it must be the case that at least one LL action occurs between s_l and s_l' and that no refinement of a HL action is complete between them. Then by Constraint 8, we have that $M_l, v[s/s_l, s'/s_l'] \models Hlc(s') \wedge \forall s''.s < s'' < s' \supset \neg Hlc(s'')$. Thus, we also have that $s_l \in (\langle\langle G\rangle\rangle \bigcirc_h m_l(\Psi))_{v,V}^{M_l}$. □

We can now extend the result to all of μATL-FO:

Theorem 11
*For any μ*ATL-FO *formula Ψ, if $M_h \sim_m M_l$, Constraints 4, 6, and 8 hold, $s_h \sim_m^{M_h,M_l} s_l$, and $s_h \in (\Psi)_{v,V}^{M_h}$, then $s_l \in (m_l(\Psi))_{v,V}^{M_l}$.*

Proof We prove the theorem in two steps. First, we show that Lemma 10 can be extended to the infinitary version of ATL-FO that supports arbitrary infinite disjunction of formulas sharing the same free variables (van Benthem 1983). Then, we recall that fixpoints can be translated into this infinitary logic, thus guaranteeing that the result extends to the whole μATL-FO logic.

Let Γ be a possibly infinite set of open ATL-FO formulas. The semantics of $\bigvee\Gamma$ is $s_h \in (\bigvee\Gamma)_v^{M_h}$ if and only if $s_h \in (\Psi')_v^{M_h}$ for some $\Psi' \in \Gamma$. Arbitrary infinite conjunction is obtained for free through negation. Lemma 10 extends to this arbitrary infinite disjunction. By the induction hypothesis, we have that if $s_h \in (\Psi)_v^{M_h}$, then $s_l \in (m_l(\Psi))_v^{M_l}$. Given the semantics of $\bigvee\Gamma$ above, this implies that if $s_h \in (\bigvee\Gamma)_v^{M_h}$, then $s_l \in (m_l(\bigvee\Gamma))_v^{M_l}$ (we assume that $m_l(\bigvee\{\Psi, \ldots, \Psi'\}) \doteq \bigvee\{m_l(\Psi), \ldots, m_l(\Psi')\}$).

In order to extend the result to the whole μATL-FO, we translate μ-calculus *approximates* into the infinitary ATL-FO (see (Bradfield and Stirling 2007; van Benthem 1983)), where the approximate of index α is denoted by $\mu^\alpha Z.\Psi$ for least fixpoint formulas $\mu Z.\Psi$ and $\nu^\alpha Z.\Psi$ for greatest fixpoint formulas $\nu Z.\Psi$. This is a standard result that holds also for μATL-FO. In particular, such approximates are as follows:

$$\begin{array}{rclrcl} \mu^0 Z.\Psi & = & False & \nu^0 Z.\Psi & = & True\\ \mu^{\beta+1} Z.\Psi & = & \Psi[Z/\mu^\beta Z.\Psi] & \nu^{\beta+1} Z.\Psi & = & \Phi[Z/\nu^\beta Z.\Psi]\\ \mu^\lambda Z.\Psi & = & \bigvee_{\beta<\lambda} \mu^\beta Z.\Psi & \nu^\lambda Z.\Psi & = & \bigwedge_{\beta<\lambda} \nu^\beta Z.\Psi\end{array}$$

where λ is a limit ordinal, and the notation $\Psi[Z/\nu^\beta Z.\Psi]$ denotes the formula obtained from Ψ by replacing each occurrence of Z by $\nu^\beta Z.\Psi$. By the Tarski and Knaster Theorem (Tarski 1955), the fixpoints and their approximates are connected by the following properties:

- $s \in (\mu Z.\Psi)_{v,V}^M$ if and only if there exists an ordinal α such that $s \in (\mu^\alpha Z.\Psi)_{v,V}^M$ and, for every $\beta < \alpha$, it holds that $s \notin (\mu^\beta Z.\Psi)_{v,V}^M$;
- $s \notin (\nu Z.\Psi)_{v,V}^M$ if and only if there exists an ordinal α such that $s \notin (\nu^\alpha Z.\Psi)_{v,V}^M$ and, for every $\beta < \alpha$, it holds that $s \in (\nu^\beta Z.\Psi)_{v,V}^M$.

Since each approximate, including the ones corresponding exactly to the least and greatest fixpoints, can be written as an infinitary ATL-FO formula, we get the thesis. □

Notice that the HL game structure is usually much smaller than the LL one, so verifying that a μATL-FO property holds at that HL would typically be much easier than doing so at the LL. By the above theorem, if a μATL-FO property holds at the HL, then it immediately follows that the mapped property also holds at the LL, provided that the constraints are satisfied.

Example Cont. Using μATL-FO, we can express a wide range of strategic properties involving temporally extended goals. For a first example, consider the first-order ATL^* property that the dispatcher and first repair robot can ensure that some item is eventually repaired $\langle\langle\{Disp, RR_1\}\rangle\rangle\Diamond\exists i.Repaired(i)$, which can be expressed in μATL-FO as $\mu Z.\ \exists i.Repaired(i) \vee \langle\langle\{Disp, RR_1\}\rangle\rangle \bigcirc Z$, call this Ψ_h^{Fr}. This holds at the HL, i.e., $S_0 \in (\Psi_h^{Fr})^{M_h^{rs}}$, because item 1 has arrived initially, and the joint strategy where $Disp$ assigns the item to RR_1 who then repairs it achieves the goal no matter what other agents do. Since Constraints 4, 6, and 8 hold, by Theorem 11 it follows that the mapped property holds at the LL as well, i.e., $S_0 \in (m_l(\Psi_h^{Fr}))^{M_l^{rs}}$

For a second example, consider the first-order ATL^* property stating that the coalition of all the repair shop agents $C = \{Disp, RR_1, RR_2, Sh\}$ has a joint strategy to ensure that all the items that arrive are eventually shipped $\langle\langle C\rangle\rangle\, \forall i.\Box(Arrived(i) \supset \Diamond Shipped(i))$, which can be expressed in μATL-FO as

$$\begin{aligned}&\forall i.\nu X.(Arrived(i) \supset\\ &\qquad \mu Y.Shipped(i) \vee \langle\langle C\rangle\rangle \bigcirc Y) \wedge \langle\langle C\rangle\rangle \bigcirc X\end{aligned}$$

We can show that this property, call it Ψ_h^{GaFs} holds at the HL, i.e., $S_0 \in (\Psi_h^{GaFs})^{M_h^{rs}}$.[4] By Theorem 11, it follows that the mapped property also holds at the LL: $S_0 \in (m_l(\Psi_h^{GaFs}))^{M_l^{rs}}$.

Discussion and Conclusion

In this paper, we presented a general first-order framework for abstraction of synchronous multi-agent game structures. We showed that we can exploit abstraction in verifying strategic properties expressed in μATL-FO over SCSGSs if we assume that agents can always execute abstract moves to completion even if not fully controlling their outcomes.

Note that our approach is quite different from the usual abstraction techniques employed in model checking (Clarke et al. 2000). The latter apply "automatic" abstraction and refinement methods to address state-explosion in the verification of system specifications. Such techniques have also been proposed for concurrent game structures and strategy logics, e.g., (Belardinelli, Ferrando, and Malvone 2023). The abstractions that these methods generate need not be meaningful to users of the system. Our abstractions involve new HL fluents and moves that are meaningful to the users and can be used to express many HL temporally extended goals. They can be used to speed up strategic reasoning and verifying strategic properties, and also to give HL explanations of system behavior.

In future work, we would like to extend μATL-FO to support the use of programs to express dynamic properties. In the propositional setting our programs could be captured by Linear Dynamic Logic on Finite Traces (LDL$_f$) (De Giacomo and Vardi 2013), for which verification and synthesis have been shown to be decidable and effective (De Giacomo and Vardi 2015; De Giacomo and Rubin 2018; Camacho et al. 2018). However, when we move to the (first-order) situation calculus, reasoning becomes undecidable in general. Interestingly, conditions for decidability are known both for verification (De Giacomo, Lespérance, and Patrizi 2016; Calvanese et al. 2022) and synthesis (De Giacomo et al. 2022). It would be worthwhile to investigate how to leverage these results in our context.

We are also interested in developing techniques for automated generation of abstractions that serve users' purposes. (Luo et al. 2020) shows that one can use the notion of *forgetting* (of LL fluent and action symbols) to automatically obtain a sound and complete HL abstraction of a LL BAT *for a given mapping* under certain conditions; they also show that such an abstraction is computable in the propositional case. (Luo 2023) studies automated verification of the existence of a propositional agent abstraction given a LL finite-state labeled transition system and a refinement mapping where HL actions are mapped into loop-free LL programs. He shows that the problem can be reduced to a CTLK (an extension of CTL with an epistemic operator) model checking problem, which can be solved in PTIME. But in general, there are many different abstractions of a LL theory, each of which may be useful for a different purpose. So defining an abstract language and mapping for a domain is non-trivial. Some human input is likely to be required, e.g., the modeler might specify the goals of the abstraction, or which details can be considered unimportant.

We also want to generalize our framework for imperfect information game settings. There has been a lot of work on strategy logics for such settings including the case where one wants to ensure that each agent knows her strategy (van der Hoek and Wooldridge 2002; Jamroga and van der Hoek 2004; van der Hoek, Jamroga, and Wooldridge 2005; Herzig, Lorini, and Walther 2013; Xiong and Liu 2016; Berthon et al. 2021). The Game Description Language has also been extended for them (Thielscher 2010; Schiffel and Thielscher 2014; Engesser et al. 2021) and there is also related work in multi-agent epistemic planning (Muise et al. 2015; Le et al. 2018; Bolander et al. 2020). Finally, we plan to investigate the use of multi-tier game models/structures (Ciolek et al. 2020) to support flexible strategic reasoning, where one only considers unlikely contingencies when necessary, as well as reasoning about cooperative strategies where agents delegate subgoals to others without knowing how these agents will achieve the delegated subgoals.

Acknowledgements

This work is partially supported by the National Science and Engineering Research Council of Canada, by the ERC Advanced Grant WhiteMech (No. 834228), by the EU ICT-48

[4] A suitable strategy is for the dispatcher to assign the items to a repair agent in the order in which they arrive, for each repair agent to repair the items in the order in which they are assigned to them, and for the shipper to ship the items in the order they are repaired.

Project TAILOR (No. 952215), by York University, and by the University of Regina.

References

Alur, R.; Henzinger, T. A.; and Kupferman, O. 2002. Alternating-time temporal logic. *J. ACM*, 49(5): 672–713.

Banihashemi, B.; De Giacomo, G.; and Lespérance, Y. 2017. Abstraction in Situation Calculus Action Theories. In *AAAI*, 1048–1055.

Banihashemi, B.; De Giacomo, G.; and Lespérance, Y. 2023. Abstraction of Nondeterministic Situation Calculus Action Theories. In *IJCAI*, 3112–3122.

Belardinelli, F.; Ferrando, A.; and Malvone, V. 2023. An abstraction-refinement framework for verifying strategic properties in multi-agent systems with imperfect information. *Artif. Intell.*, 316: 103847.

Berthon, R.; Maubert, B.; Murano, A.; Rubin, S.; and Vardi, M. Y. 2021. Strategy Logic with Imperfect Information. *ACM Trans. Comput. Log.*, 22(1): 5:1–5:51.

Bolander, T.; Charrier, T.; Pinchinat, S.; and Schwarzentruber, F. 2020. DEL-based epistemic planning: Decidability and complexity. *Artif. Intell.*, 287: 103304.

Bradfield, J.; and Stirling, C. 2007. Modal mu-calculi. In *Handbook of Modal Logic*, volume 3, 721–756.

Calvanese, D.; De Giacomo, G.; Montali, M.; and Patrizi, F. 2022. Verification and Monitoring for First-Order LTL with Persistence-Preserving Quantification over Finite and Infinite Traces. In *IJCAI*, 2553–2560.

Camacho, A.; Baier, J. A.; Muise, C. J.; and McIlraith, S. A. 2018. Finite LTL Synthesis as Planning. In *ICAPS*, 29–38.

Chatterjee, K.; Henzinger, T. A.; and Piterman, N. 2010. Strategy logic. *Inf. Comput.*, 208(6): 677–693.

Ciolek, D. A.; D'Ippolito, N.; Pozanco, A.; and Sardiña, S. 2020. Multi-Tier Automated Planning for Adaptive Behavior. In *ICAPS*, 66–74.

Clarke, E. M.; Grumberg, O.; Jha, S.; Lu, Y.; and Veith, H. 2000. Counterexample-Guided Abstraction Refinement. In *CAV*, volume 1855 of *Lecture Notes in Computer Science*, 154–169.

De Giacomo, G.; Felli, P.; Logan, B.; Patrizi, F.; and Sardiña, S. 2022. Situation calculus for controller synthesis in manufacturing systems with first-order state representation. *Artif. Intell.*, 302: 103598.

De Giacomo, G.; and Lespérance, Y. 2021. The Nondeterministic Situation Calculus. In *KR*, 216–226.

De Giacomo, G.; Lespérance, Y.; and Levesque, H. J. 2000. ConGolog, A Concurrent Programming Language Based on the Situation Calculus. *Artif. Intell*, 121(1–2): 109–169.

De Giacomo, G.; Lespérance, Y.; and Muise, C. J. 2012. On supervising agents in situation-determined ConGolog. In *AAMAS*, 1031–1038.

De Giacomo, G.; Lespérance, Y.; and Patrizi, F. 2016. Bounded Situation Calculus Action Theories. *Artif. Intell*, 237: 172–203.

De Giacomo, G.; Lespérance, Y.; and Pearce, A. R. 2010. Situation Calculus Based Programs for Representing and Reasoning about Game Structures. In *KR*.

De Giacomo, G.; Lespérance, Y.; and Pearce, A. R. 2016. Situation Calculus Game Structures and GDL. In *ECAI*, 408–416.

De Giacomo, G.; and Rubin, S. 2018. Automata-Theoretic Foundations of FOND Planning for LTLf and LDLf Goals. In *IJCAI*, 4729–4735.

De Giacomo, G.; and Vardi, M. Y. 2013. Linear Temporal Logic and Linear Dynamic Logic on Finite Traces. In *IJCAI*, 854–860.

De Giacomo, G.; and Vardi, M. Y. 2015. Synthesis for LTL and LDL on Finite Traces. In *IJCAI*, 1558–1564.

Engesser, T.; Mattmüller, R.; Nebel, B.; and Thielscher, M. 2021. Game description language and dynamic epistemic logic compared. *Artif. Intell.*, 292: 103433.

Herzig, A.; Lorini, E.; and Walther, D. 2013. Reasoning about Actions Meets Strategic Logics. In *LORI*, volume 8196 of *Lecture Notes in Computer Science*, 162–175.

Jamroga, W.; and van der Hoek, W. 2004. Agents that Know How to Play. *Fundam. Informaticae*, 63(2-3): 185–219.

Le, T.; Fabiano, F.; Son, T. C.; and Pontelli, E. 2018. EFP and PG-EFP: Epistemic Forward Search Planners in Multi-Agent Domains. In *ICAPS*, 161–170.

Levesque, H. J.; Reiter, R.; Lespérance, Y.; Lin, F.; and Scherl, R. B. 1997. GOLOG: A Logic Programming Language for Dynamic Domains. *J. Logic Programming*, 31: 59–84.

Luo, K. 2023. Automated Verification of Propositional Agent Abstraction for Classical Planning via CTLK Model Checking. In *AAAI*, 6475–6482.

Luo, K.; Liu, Y.; Lespérance, Y.; and Lin, Z. 2020. Agent Abstraction via Forgetting in the Situation Calculus. In *ECAI*, volume 325, 809–816.

Marrella, A.; Mecella, M.; and Sardiña, S. 2017. Intelligent Process Adaptation in the SmartPM System. *ACM Trans. Intell. Syst. Technol.*, 8(2): 25:1–25:43.

McCarthy, J.; and Hayes, P. J. 1969. Some Philosophical Problems From the Standpoint of Artificial Intelligence. *Machine Intelligence*, 4: 463–502.

Milner, R. 1971. An Algebraic Definition of Simulation Between Programs. In *IJCAI*, 481–489.

Milner, R. 1989. *Communication and concurrency*. PHI Series in computer science. ISBN 978-0-13-115007-2.

Mogavero, F.; Murano, A.; Perelli, G.; and Vardi, M. Y. 2014. Reasoning About Strategies: On the Model-Checking Problem. *ACM Trans. Comput. Log.*, 15(4): 34:1–34:47.

Muise, C. J.; Belle, V.; Felli, P.; McIlraith, S. A.; Miller, T.; Pearce, A. R.; and Sonenberg, L. 2015. Planning Over Multi-Agent Epistemic States: A Classical Planning Approach. In *AAAI*, 3327–3334.

Pauly, M. 2002. A Modal Logic for Coalitional Power in Games. *J. Log. Comput.*, 12(1): 149–166.

Pinto, J. 1998. Concurrent Actions and Interacting Effects. In *KR*, 292–303.

Reiter, R. 2001. *Knowledge in Action. Logical Foundations for Specifying and Implementing Dynamical Systems*.

Schiffel, S.; and Thielscher, M. 2014. Representing and Reasoning About the Rules of General Games With Imperfect Information. *J. Artif. Intell. Res.*, 49: 171–206.

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

Thielscher, M. 2010. A General Game Description Language for Incomplete Information Games. In *AAAI*.

van Benthem, J. 1983. *Modal Logic and Classical Logic*.

van der Hoek, W.; Jamroga, W.; and Wooldridge, M. J. 2005. A logic for strategic reasoning. In *AAMAS*, 157–164.

van der Hoek, W.; and Wooldridge, M. J. 2002. Tractable multiagent planning for epistemic goals. In *AAMAS*, 1167–1174.

Xiong, L.; and Liu, Y. 2016. Strategy Representation and Reasoning for Incomplete Information Concurrent Games in the Situation Calculus. In *IJCAI*, 1322–1329.