Verifying ConGolog Programs on Bounded Situation Calculus Theories*

Giuseppe De Giacomo
Sapienza Univ. Roma
Rome, Italy
degiacomo@dis.uniroma1.it

Yves Lespérance
York University
Toronto, ON, Canada
lesperan@cse.yorku.ca

Fabio Patrizi
Free Univ. Bozen/Bolzano
Bolzano, Italy
patrizi@inf.unibz.it

Sebastian Sardina
RMIT University
Melbourne, Australia
sebastian.sardina@rmit.edu.au

Abstract

We address verification of high-level programs over situation calculus action theories that have an infinite object domain, but bounded fluent extensions in each situation. We show that verification of μ-calculus temporal properties against ConGolog programs over such bounded theories is decidable in general. To do this, we reformulate the transition semantics of ConGolog to keep the bindings of "pick variables" into a separate variable environment whose size is naturally bounded by the number of variables. We also show that for situation-determined ConGolog programs, we can compile away the program into the action theory itself without loss of generality. This can also be done for arbitrary programs, but only to check certain properties, such as if a situation is the result of a program execution, not for μ-calculus verification.

Introduction

Most work on verification of agent systems/programs is restricted to finite state systems (Baier and Katoen 2008; Lomuscio, Qu, and Raimondi 2009). In AI, starting from the seminal work in (De Giacomo, Ternovskaia, and Reiter 1997) and (Claßen and Lakemeyer 2008), there has been growing interest in verifying agent programs with a first-order state representation as in the situation calculus. Recently, (De Giacomo, Lespérance, and Patrizi 2012) have shown that verification of μ-calculus temporal properties over *bounded action theories* in the situation calculus is decidable. Such theories have an infinite object domain, but the number of object tuples that belong to fluents in each situation remains bounded. Nonetheless, they deal with infinitely many objects over the course of an infinite execution.

On top of action theories, high level programming languages, such as ConGolog (De Giacomo, Lespérance, and Levesque 2000), have been introduced to express semantically rich agent behaviors. ConGolog programs include conditionals, loops, and concurrency as usual programming languages, but their atomic actions are specified in terms of preconditions and effects defined in a situation calculus action theory and their tests involve fluents whose changing value is specified by the theory. Notably, such programs may be highly nondeterministic and allow many possible executions. In particular, a program may nondeterministically pick an object from the infinite domain and execute some actions on it. The decidability results for verification of bounded action theories do not apply to ConGolog programs, since unlike domain objects, which are infinitely many but unstructured, programs are unbounded terms defined inductively. So a natural question is whether such results can be extended to deal with ConGolog programs as well, to check properties like termination, safety, total correctness, etc. In this paper, we show that this is indeed the case: *verification of first-order μ-calculus temporal properties against ConGolog programs over bounded action theories is decidable*.

To obtain this result we develop a new transition semantics for ConGolog programs, which is shown equivalent to the original one, that keeps bindings of nondeterministically picked object variables in a separate "program environment" whose size in terms of number of pick variables is naturally bounded. Differently from the original semantics, the set of remaining programs (without assignment to the pick variables, now separated) that can be produced in any execution of a given initial program is in fact finite, and can be viewed as program counter values. Thus there is no need to define a complex encoding of programs as terms in the situation calculus as in the original ConGolog semantics (De Giacomo, Lespérance, and Levesque 2000). Leveraging on this new semantics, we can adapt the approach of (De Giacomo, Lespérance, and Patrizi 2012; 2016) to show decidability of verification of temporal properties. With the new semantics it becomes clear that if the action theory is bounded, then every reachable program configuration (formed by the remaining program, the variable environment state, and the situation) is also bounded.

This main result is complemented by a second one: for programs that are "situation-determined" (De Giacomo, Lespérance, and Muise 2012), we can compile the program into the action theory itself without loss of generality, so that the executable situations become those that can be generated by executing the program. In this case, we can verify temporal properties of the program by using the original (De Giacomo, Lespérance, and Patrizi 2012; 2016) verification method on such a compiled theory. For non-situation-determined programs, while the compiled theory cannot be

*We acknowledge the support of Sapienza 2015 project "Immersive Cognitive Environments," the NSERC of Canada, Provincia Autonoma di Bolzano (under project VeriSynCoPateD), and a Sapienza 2014 Visiting Grant (for the last author).

used for μ-calculus verification, it can still be used to check properties such as existence of a terminating execution of the program as in (Fritz, Baier, and McIlraith 2008).

Preliminaries

The *situation calculus* (McCarthy and Hayes 1969; Reiter 2001) is a logical language for representing and reasoning about dynamic worlds with three sorts: objects, actions, and situations. All changes to the world are the result of *actions*, which are terms in the logic. A *situation* term denotes a sequence of actions: the constant S_0 denotes the initial situation (no action has yet been done), whereas term $do(a, s)$ denotes the successor situation resulting from performing action a in situation s. Predicates whose extension vary from situation to situation are called *fluents*, and are denoted by symbols taking a situation term as their last argument (e.g., $Holding(x, s)$), while the other arguments are of sort object. We assume that there are no functions other than constants and no predicates other then fluents.

Within this language, one can formulate action theories to describe how the world changes as a result of actions. A well studied and popular type of such theories are *basic action theories* (Reiter 2001). A basic action theory $\mathcal{D}$ is a collection of first-order axioms (plus a domain independent second-order characterization of situation terms) conveniently specifying (in terms of size and computational properties): *(i)* actions' preconditions, by characterizing special predicate $Poss(a, s)$ through *precondition axioms*, capturing when action a is executable in situation s; *(ii)* actions' effects and non-effects (i.e., frame problem) by the so-called *successor state axioms*; and *(iii)* the world's initial state. We assume $\mathcal{D}$ to have a finite number of action types, each of which takes a tuple of objects as arguments, and to have countably infinitely many object constants, on which we adopt the unique name assumption.[1] Notice that the latter implies an infinite object domain.

To represent and reason about complex actions or processes obtained by executing atomic actions, *high-level programming languages* have been defined. Here we concentrate on ConGolog (De Giacomo, Lespérance, and Levesque 2000), which includes the following constructs:

$$\delta ::= \alpha \mid \varphi? \mid \delta_1; \delta_2 \mid \textbf{if } \varphi \textbf{ then } \delta_1 \textbf{ else } \delta_2 \mid \textbf{while } \varphi \textbf{ do } \delta$$
$$\delta_1 | \delta_2 \mid \pi x.\delta \mid \delta_1 \| \delta_2$$

In the above, α is an action term, possibly with parameters, and φ is situation-suppressed formula, that is, one with all situation arguments in fluents suppressed. As usual, we denote by $\varphi[s]$ the situation calculus formula obtained from φ by restoring the situation argument s into all fluents in φ. Program $\delta_1 | \delta_2$ nondeterministically chooses between programs δ_1 and δ_2. Program $\pi z.\delta(z)$ nondeterministically "picks" an object d to bind to variable z and then executes program $\delta(z)$ with z assigned to d;[2] e.g., the program **while** $\exists x.\neg OnTable(x)$ **do** $\pi z.\neg OnTable(z)?; table(z)$ repeatedly picks a block that is not on the table and tables it, until all blocks are on the table. Program δ^* performs δ zero or more times. $\delta_1 \| \delta_2$ expresses the concurrent execution (interpreted as interleaving) of programs δ_1 and δ_2. In this paper we do not allow for recursive procedures, though we allow for a form of (tail) recursion through $*$ and **while**. We also leave out concurrent iteration $\delta^{||}$. Both of these constructs require handling unbounded information (stack for recursive procedures and iterated duplication of the program terms for concurrent iteration) even without data.

The semantics of ConGolog is specified in terms of single-step transitions, using two predicates: *(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. Both are defined inductively by axioms, e.g., $Trans(\pi z.\delta(z), s, \delta', s') \equiv \exists z.Trans(\delta(z), s, \delta', s')$. Here, we follow (Claßen and Lakemeyer 2008; De Giacomo, Lespérance, and Pearce 2010), in which the test construct $\varphi?$ yields no transition and is final when satisfied. This results in a *synchronous test* construct which does not allow interleaving (every transition involves the execution of an action). Given this, **if** and **while** can be treated as abbreviations (**if** ϕ **then** δ_1 **else** δ_2) $\doteq (\phi?; \delta_1 \mid \neg\phi?; \delta_2)$ and (**while** ϕ **do** δ) $\doteq ((\phi?; \delta)^*; \neg\phi?)$. Below, we denote by $\mathcal{C}$ the axioms defining the ConGolog programming language.

Trans and *Final* with Variable Environments

Since we have infinitely many objects to bind pick variables with, the number of remaining programs for $\pi x.\delta(x)$ is typically infinite. We can better understand the nature of such remaining programs by separating the program terms themselves from the assignments to pick variables. Let δ_0 be the initial program and assume wlog that all pick variables are renamed apart. The number n of such variables is indeed finite. For convenience let's assume a predefined ordering on such variables. We can then introduce an *environment* term $\vec{x} = \langle x_1, \ldots, x_n \rangle$, consisting of a tuple of object terms in which each component i stores the current value x_i of the i-th pick variable of δ_0. The tuple will change over time as the program executes, and its initial content is arbitrary given that δ_0 is always closed wrt pick variables. Importantly, the environment $\vec{x}$ can denote infinitely many tuples of values along a computation; however, at each moment of the computation, the environment term, consisting of a single tuple of arity n, maintains only a bounded number (smaller that the size of δ_0) of values.

By keeping the values assigned to pick variables in the environment, we can avoid substituting them in the program itself. As a result, the set of all possible remaining programs is finite. Specifically, it is possible to define inductively the *syntactic closure* Γ_{δ_0} of the program δ_0, as follows: (1) $\delta_0, nil \in \Gamma_{\delta_0}$; (2) if $\delta_1; \delta_2 \in \Gamma_{\delta_0}$ and $\delta_1' \in \Gamma_{\delta_1}$, then $\delta_1'; \delta_2 \in \Gamma_{\delta_0}$ and $\Gamma_{\delta_2} \subseteq \Gamma_{\delta_0}$; (3) if $\delta_1 | \delta_2 \in \Gamma_{\delta_0}$, then $\Gamma_{\delta_1}, \Gamma_{\delta_2} \subseteq \Gamma_{\delta_0}$; (4) if $\pi z.\delta \in \Gamma_{\delta_0}$, then $\Gamma_\delta \subseteq \Gamma_{\delta_0}$; (5) if $\delta^* \in \Gamma_{\delta_0}$, then $\delta; \delta^* \in \Gamma_{\delta_0}$; (6) if $\delta_1 \| \delta_2 \in \Gamma_{\delta_0}$ and $\delta_1' \in \Gamma_{\delta_1}$ and $\delta_2' \in \Gamma_{\delta_2}$, then $\delta_1' \| \delta_2' \in \Gamma_{\delta_0}$.

Theorem 1 *The syntactic closure* Γ_{δ_0} *of a ConGolog pro-*

[1] In (De Giacomo, Lespérance, and Patrizi 2012), standard names were assumed. This assumption is dropped in (De Giacomo, Lesperance, and Patrizi 2016).

[2] Given that we have finitely many action types, wlog, we disallow pick variables to range over actions directly.

gram δ_0 is linear in the size of δ_0 if the concurrency operator does not occur, and exponential otherwise.

Note that the finite set of "program strings" in Γ_{δ_0} can be viewed as values of a program counter over program δ_0.

Given an initial program δ_0, a complete configuration is now formed by a triple $(\delta, \vec{x}, s)$, where $\delta \in \Gamma_{\delta_0}$, $\vec{x}$ is an environment for δ_0, and s is the current situation. We can inductively define *Trans* and *Final* over such configurations:

$$\begin{array}{l}
\mathit{Trans}(\alpha, \vec{x}, s, \delta', \vec{x}', s') \equiv \\
\quad s' = do(\alpha[\vec{x}], s) \wedge \mathit{Poss}(\alpha[\vec{x}], s) \wedge \delta' = nil \wedge \vec{x}' = \vec{x} \\
\mathit{Trans}(\varphi?, \vec{x}, s, \delta', \vec{x}', s') \equiv False \\
\mathit{Trans}(\delta_1; \delta_2, \vec{x}, s, \delta', \vec{x}', s') \equiv \\
\quad \mathit{Trans}(\delta_1, \vec{x}, s, \delta_1', \vec{x}', s') \wedge \delta' = \delta_1'; \delta_2 \vee \\
\quad \mathit{Final}(\delta_1, \vec{x}, s) \wedge \mathit{Trans}(\delta_2, \vec{x}, s, \delta', \vec{x}', s') \\
\mathit{Trans}(\delta_1 | \delta_2, \vec{x}, s, \delta', \vec{x}', s') \equiv \\
\quad \mathit{Trans}(\delta_1, \vec{x}, s, \delta', \vec{x}', s') \vee \mathit{Trans}(\delta_2, \vec{x}, s, \delta', \vec{x}', s') \\
\mathit{Trans}(\pi z.\delta, \vec{x}, s, \delta', \vec{x}', s') \equiv \exists d.\mathit{Trans}(\delta, \vec{x}_d^z, s, \delta', \vec{x}', s') \\
\mathit{Trans}(\delta^*, \vec{x}, s, \delta', \vec{x}', s') \equiv \mathit{Trans}(\delta, \vec{x}, s, \delta'', \vec{x}', s') \wedge \delta' = \delta''; \delta^* \\
\mathit{Trans}(\delta_1 \| \delta_2, \vec{x}, s, \delta', \vec{x}', s') \equiv \\
\quad \mathit{Trans}(\delta_1, \vec{x}, s, \delta_1', \vec{x}', s') \wedge \delta' = \delta_1' \| \delta_2 \vee \\
\quad \mathit{Trans}(\delta_2, \vec{x}, s, \delta_2', \vec{x}', s') \wedge \delta' = \delta_1 \| \delta_2' \\
\mathit{Trans}(nil, \vec{x}, s, \delta', \vec{x}', s') \equiv False
\end{array}$$

$$\begin{array}{l}
\mathit{Final}(\alpha, \vec{x}, s) \equiv False \\
\mathit{Final}(\varphi?, \vec{x}, s) \equiv \varphi[\vec{x}][s] \\
\mathit{Final}(\delta_1; \delta_2, \vec{x}, s) \equiv \mathit{Final}(\delta_1, \vec{x}, s) \wedge \mathit{Final}(\delta_2, \vec{x}, s) \\
\mathit{Final}(\delta_1 | \delta_2, \vec{x}, s) \equiv \mathit{Final}(\delta_1, \vec{x}, s) \vee \mathit{Final}(\delta_2, \vec{x}, s) \\
\mathit{Final}(\pi z.\delta, \vec{x}, s) \equiv \exists d.\mathit{Final}(\delta, \vec{x}_d^z, s) \\
\mathit{Final}(\delta^*, \vec{x}, s) \equiv True \\
\mathit{Final}(\delta_1 \| \delta_2, \vec{x}, s) \equiv \mathit{Final}(\delta_1, \vec{x}, s) \wedge \mathit{Final}(\delta_2, \vec{x}, s) \\
\mathit{Final}(nil, \vec{x}, s) \equiv True
\end{array}$$

Above, we use the notation $\alpha[\vec{x}]$, $\varphi[\vec{x}]$, and $\delta[\vec{x}]$ to denote the action term, formula, and program, resp., obtained from the "strings" α, φ, and δ, resp., by replacing all *free* pick variables with the object terms to which they are bound in environment $\vec{x}$. We use $\vec{x}_d^z$ to denote the environment term obtained from $\vec{x}$ by replacing the element corresponding to pick variable z with d. Note how the new axiomatization does not substitute values of the pick variables into the remaining program but keeps them in the environment, e.g., we might have $\mathit{Trans}(\pi x.\pi y.A(x,y); B(x,y), \langle O_{25}, O_{13}\rangle, S_0, nil; B(x,y), \langle O_2, O_5\rangle, do(A(O_2, O_5), S_0))$.

Let $\mathcal{C}_{new}$ be the new axioms above for ConGolog.

Theorem 2 *Let $\mathcal{D}$ be a situation calculus action theory, δ_0 a ConGolog program, and Γ_{δ_0} its syntactic closure. Then, for every model M of $\mathcal{D} \cup \mathcal{C}_{new}$ and program $\delta \in \Gamma_{\delta_0}$:*

$$\{\delta' \mid M \models \exists \vec{x}, \vec{x}', s, s'.\mathit{Trans}(\delta, \vec{x}, s, \delta', \vec{x}', s')\} \subseteq \Gamma_{\delta_0}.$$

This theorem guarantees that all programs that δ_0 can evolve into, according to the new environment-based definition of *Trans* and *Final*, must be in its syntactic closure Γ_{δ_0}. It follows then that we only need a fixed number of program symbols (depending only on the original program δ_0) to denote all programs in a computation. This is very different from the standard definition of *Trans* and *Final* discussed in the preliminaries, which relies on a complex second-order encoding of programs as terms (De Giacomo, Lespérance, and Levesque 2000). Nonetheless, the new semantics is equivalent to it:

Theorem 3 *Let $\mathcal{D}$ be a situation calculus action theory and δ_0 a ConGolog program. Then, for every $\delta, \delta' \in \Gamma_{\delta_0}$, theory $\mathcal{D} \cup \mathcal{C} \cup \mathcal{C}_{new}$ entails the following equivalences:*

$$\mathit{Trans}(\delta, \vec{x}, s, \delta', \vec{x}', s') \equiv \mathit{Trans}(\delta[\vec{x}], s, \delta'[\vec{x}']s'),$$
$$\mathit{Final}(\delta, \vec{x}, s) \equiv \mathit{Final}(\delta[\vec{x}], s).$$

To fully appreciate this theorem one should recall Theorem 2 of (De Giacomo, Lespérance, and Levesque 2000) that says that every model M of $\mathcal{D}$ can be univocally extended to a model of $\mathcal{D} \cup \mathcal{C}$. The same is true for $\mathcal{C}_{new}$. Hence the sequences of transitions that δ_0 can perform from S_0 are fully determined and coincide in the two semantics.

We finally observe that by unfolding recursively the new *Trans* and *Final* we get pure first-order situation calculus formulas in which program terms and enviroment terms disappear. Notably, this means that the new *Trans* and *Final* could be treated as abbreviations and the introduction of program and enviroment terms could be avoided altogether, analogously to what we have for *Do* in Golog (Levesque et al. 1997; Fritz, Baier, and McIlraith 2008). Moreover $\mathit{Final}(\delta, \vec{x}, s)$ results into a formula that is uniform in situation s. This is also the case for $\mathit{Trans}(\delta, \vec{x}, s, \delta', \vec{x}', s')$ when s' is instantiated to a situation term of the form $do(a, s)$; indeed s' only occurs in equalities of the form $s' = do(\alpha, s)$, whose instantiation becomes equivalent to $a = \alpha$. We exploit this observation in the last part of the paper.

Program Verification for Bounded Theories

The verification logic. For expressing temporal properties, we adopt a variant of the μ-calculus, one of the most powerful temporal logics, subsuming both linear time logics, such as LTL, and branching time logics such as CTL and CTL* (Emerson 1996). In particular we use a first-order variant of it, called $\mu\mathcal{L}$, analogous to that in (De Giacomo, Lespérance, and Patrizi 2012) with the difference of having an explicit predicate for *Final*:

$$\Phi ::= \mathit{Final} \mid \varphi \mid \neg\Phi \mid \Phi_1 \wedge \Phi_2 \mid \langle - \rangle \Phi \mid Z \mid \mu Z.\Phi$$

where φ is an arbitrary closed uniform *situation-suppressed* (i.e., with all situation arguments in fluents suppressed) situation calculus FO formula, and Z is a second-order (0-ary) predicate variable.[3] We shall use the standard abbreviations, namely, $\Phi_1 \vee \Phi_2 = \neg(\neg\Phi_1 \wedge \neg\Phi_2)$, $[-]\Phi = \neg\langle - \rangle\neg\Phi$, and $\nu Z.\Phi = \neg\mu Z.\neg\Phi[Z/\neg Z]$. Formula $\langle - \rangle\Phi$ states that there is a successor state where Φ holds and, hence, $[-]\Phi$ that Φ holds in all successor states. The *fixpoint formulas* $\mu Z.\Phi$ and $\nu Z.\Phi$ denote respectively the *least* and the *greatest fixpoint* of the formula Φ, seen as a predicate transformer $\lambda Z.\Phi$ (their existence is guaranteed by the syntactic monotonicity

[3] We assume, wlog, that φ does not mention action terms. In uniform formulas, actions can only appear in equality atoms, which, assuming finite action types, can be replaced with equalities over the action arguments, see (De Giacomo, Lespérance, and Patrizi 2012). As usual in the μ-calculus, formulas of the form $\mu Z.\Phi$ must satisfy *syntactic monotonicity*, i.e., every occurrence of Z in Φ must be within the scope of an even number of negation symbols.

of Φ), and are used to express typical program properties; e.g., $\mu Z.\mathit{Final} \vee [-]Z$ expresses termination of the program on all possible runs, while $\mu Z.\mathit{Final} \vee \langle - \rangle Z$ expresses possible termination of the program by suitably making nondeterministic choices. Instead $\nu Z.\varphi \wedge [-]Z$ expresses safety, i.e., that φ always holds along the execution of the program, and $\nu Z.(\mathit{Final} \supset \varphi) \wedge [-]Z$ expresses a form of partial correctness: when the program terminates, φ holds; by further requiring termination, we get total correctness.

Formulas of $\mu\mathcal{L}$ are interpreted on *transition systems (TS)*, over the situation-suppressed fluents of a basic action theory $\mathcal{D}$, of the form $T = \langle \Delta, Q, q_0, \rightarrow, L, Q_F \rangle$, where: (1) Δ is the *object domain*; (2) Q is the set of *states*; (3) $q_0 \in Q$ is the *initial state*; (4) $\rightarrow \subseteq Q \times Q$ is the *transition relation*; (5) L is a *labeling function* mapping states in Q into a FO *interpretation* of the situation-suppressed fluents of $\mathcal{D}$ over Δ, i.e., for $q \in Q$, $L(q) = \langle \Delta, \cdot^{L(q)} \rangle$, with Δ being the *interpretation domain* and $\cdot^{L(q)}$ a function assigning an extension $F^{L(q)}$ over Δ to every fluent F of $\mathcal{D}$; and (6) $Q_F \subseteq Q$ is the set of *final states*. To evaluate formulas with predicate free variables, we introduce a predicate variable valuation v, mapping a predicate variable Z into subsets of Q. The *extension function* $(\cdot)^T_v$, defined inductively, maps $\mu\mathcal{L}$ formulas into subsets of Q (those where the formula is true):

$$
\begin{array}{lcl}
(\mathit{Final})^T_v & = & Q_F; \\
(\varphi)^T_v & = & \{q \in Q \mid L(q) \models \varphi\}; \\
(\neg\Phi)^T_v & = & Q - (\Phi)^T_v; \\
(\Phi_1 \wedge \Phi_2)^T_v & = & (\Phi_1)^T_v \cap (\Phi_2)^T_v \\
(\langle - \rangle \Phi)^T_v & = & \{q \in Q \mid q' \in Q, q \rightarrow q', q' \in (\Phi)^T_v\}; \\
(Z)^T_v & = & v(Z); \\
(\mu Z.\Phi)^T_v & = & \bigcap \{\mathcal{E} \subseteq Q \mid (\Phi)^T_{v[Z/\mathcal{E}]} \subseteq \mathcal{E}\}.
\end{array}
$$

Here, $v[Z/\mathcal{E}]$ denotes the valuation obtained from v by assigning the set $\mathcal{E}$ to Z. Note that the extension of closed (wrt predicate variables) $\mu\mathcal{L}$ formulas does not depend on v. A TS T *satisfies* a closed $\mu\mathcal{L}$-formula Φ, i.e., $T \models \Phi$, if $q_0 \in (\Phi)^T_v$, for any v. From now on we will consider only closed $\mu\mathcal{L}$ formulas, the ones of interest in verification.

A fundamental property of the μ-calculus is *bisimulation invariance*, saying that *bisimilar* TSs are indistinguishable through μ-formulas (Stirling 2001). In our case bisimulation can be defined as follows. Given two TSs $T_1 = \langle \Delta_1, Q_1, q_{01}, \rightarrow_1, L_1, Q_{F1} \rangle$ and $T_2 = \langle \Delta_2, Q_2, q_{02}, \rightarrow_2, L_2, Q_{F2} \rangle$ over the fluents and the constants of a basic action theory $\mathcal{D}$, we say that a relation $B \subseteq Q_1 \times Q_2$ is a *bisimulation* between T_1 and T_2, if $\langle q_1, q_2 \rangle \in B$ implies that: (1) $q_1 \in Q_{F1}$ iff $q_2 \in Q_{F2}$; (2) $L_1(q_1)$ and $L_2(q_2)$ are isomorphic wrt the interpretation of fluents and constants (written $L_1(q_1) \sim L_2(q_2)$); (3) for every transition $q_1 \rightarrow_1 q_1'$ of T_1, there exists a transition $q_2 \rightarrow_2 q_2'$ of T_2 s.t. $\langle q_1', q_2' \rangle \in B$; and (4) for every transition $q_2 \rightarrow_2 q_2'$ of T_2, there exists a transition $q_1 \rightarrow_1 q_1'$ of T_1 s.t. $\langle q_1', q_2' \rangle \in B$. We say that a state $q_1 \in Q_1$ *bisimulates* a state $q_2 \in Q_2$, written $q_1 \approx q_2$, if there is a bisimulation B s.t. $\langle q_1, q_2 \rangle \in B$. Relation $\approx$ itself is a bisimulation, in fact, the largest one wrt set inclusion, and is also an equivalence relation. We say that T_1 *is bisimilar to* T_2, written $T_1 \approx T_2$, if $q_{01} \approx q_{02}$. Bisimulation invariance can be shown as in (De Giacomo, Lespérance, and Patrizi 2012; 2016):

Theorem 4 *Let T_1 and T_2 be two TS such that $T_1 \approx T_2$. Then $T_1 \models \Phi$ iff $T_2 \models \Phi$, for every $\mu\mathcal{L}$ formula Φ.*

Bisimulation invariance opens for the possibility for checking an infinite TS T against a $\mu\mathcal{L}$ formula Φ, using a finite TS T'. Indeed, if $T' \approx T$, then one can check $T' \models \Phi$ instead of $T \models \Phi$. The fact is that, by finiteness of T', the former check can be easily performed by recursive application of the extension function $(\cdot)^T_v$. Thus, if one can come up with a finite T' bisimilar to the (infinite) TS "generated" by a program, then the verification of the program is decidable.

Transition systems generated by ConGolog programs. Let $\mathcal{D}$ be a basic action theory, $\mathcal{C}$ the ConGolog axioms, and M a model of $\mathcal{D}$. Exploiting the fact that M can univocally be extended to interpret both $\mathcal{C}$ and $\mathcal{C}_{new}$, we slightly abuse notation and consider M as interpreting them too. Let $\mathcal{S}$ be the set of situations in M and Π be the set of programs in M. Let us define the TS $T^o_{\delta_0,M}$ that captures exactly the configurations, formed by a program and a situation, that are "reachable" from the initial one (δ_0, S_0) as per *Trans*'s and *Final*'s extensions in M. Formally, the *TS generated by a program δ_0 (starting in S_0) over M according to $\mathcal{C}$* is a tuple $T^o_{\delta_0,M} = \langle \Delta, Q^o, q^o_0, \rightarrow^o, L^o, Q^o_F \rangle$, where: (1) Δ is M's object sort; (2) $Q^o \subseteq \Pi \times \mathcal{S}$; (3) $q^o_0 = \langle \delta_0, S_0 \rangle$; (4) Q^o and $\rightarrow^o$ are defined by mutual induction: $q^o_0 \in Q^o$ and if $\langle \delta, s \rangle \in Q^o$, then $\langle \delta', s' \rangle \in Q^o$ and $\langle \delta, s \rangle \rightarrow^o \langle \delta', s' \rangle$, for all $\langle \delta', s' \rangle$ s.t. $M \models \mathit{Trans}(\delta, s, \delta', s')$; (5) for every fluent F of $\mathcal{D}$, state $\langle \delta, s \rangle \in Q^o$, and objects $\vec{o} \in \vec{\Delta}$: $L^o(\langle \delta, s \rangle) \models F(\vec{o})$ iff $M \models F(\vec{o}, s)$; and (6) $\langle \delta, s \rangle \in Q^o_F$ iff $M \models \mathit{Final}(\delta, s)$.

Next, using the ConGolog environment-based semantic characterization above, we define the *TS generated by a program δ_0 (starting in S_0) over M according to $\mathcal{C}_{new}$* as the TS $T_{\delta_0,M} = \langle \Delta, Q, q_0, \rightarrow, L, Q_F \rangle$ s.t.: (1) Δ is M's object sort; (2) $Q \subseteq \Gamma_{\delta_0} \times \Delta^n \times \mathcal{S}$, where n is the number of pick variables in δ_0; (3) $q_0 = \langle \delta_0, \vec{x_0}, S_0 \rangle$, where $\vec{x_0}$ is arbitrary; (4) Q and $\rightarrow$ are defined by mutual induction: $q_0 \in Q$; if $\langle \delta, \vec{x}, s \rangle \in Q$, then $\langle \delta', \vec{x}', s' \rangle \in Q$ and $\langle \delta, \vec{x}, s \rangle \rightarrow \langle \delta', \vec{x}', s' \rangle$, for all $\langle \delta', \vec{x}', s' \rangle$ s.t. $M \models \mathit{Trans}(\delta, \vec{x}, s, \delta', \vec{x}', s')$; (5) for every fluent F of $\mathcal{D}$, state $q = \langle \delta, \vec{x}, s \rangle \in Q$, and objects $\vec{o} \in \vec{\Delta}$, $L(q) \models F(\vec{o})$ iff $M \models F(\vec{o}, s)$; and (6) $\langle \delta, \vec{x}, s \rangle \in Q_F$ iff $M \models \mathit{Final}(\delta, \vec{x}, s)$. Note that we are not using the infinite program sort Π of M, but only the finitely many program symbols/counters in Γ_{δ_0} (which is independent from M). The following result relates the two TSs:

Theorem 5 *Let M be a model of $\mathcal{D}$. Then, for every ConGolog program δ_0, we have that $T^o_{\delta_0,M} \approx T_{\delta_0,M}$.*

PROOF (SKETCH). By exploiting Theorem 3, it can be shown that the following relation B is a bisimulation: $B = \{\langle \langle \delta, \vec{x}, s \rangle, \langle \delta[\vec{x}], s \rangle \rangle \mid \langle \delta, \vec{x}, s \rangle \in Q, \langle \delta[\vec{x}], s \rangle \in Q^o\}$ □

Checking programs over bounded action theories. A program ConGolog δ_0 over $\mathcal{D}$ *satisfies* a $\mu\mathcal{L}$ formula Φ, written $(\delta_0, \mathcal{D} \cup \mathcal{C}) \models \Phi$, if for all models M of $\mathcal{D}$ (which univocally extend to models of $\mathcal{D} \cup \mathcal{C}$), it is the case that $T^o_{\delta_0,M} \models \Phi$. In general, verifying $(\delta_0, \mathcal{D} \cup \mathcal{C}) \models \Phi$ is undecidable (even under complete information), as it can be easily shown by reduction from the halting problem. However,

we show next that the problem is decidable for ConGolog programs running over *bounded action theories* (De Giacomo, Lespérance, and Patrizi 2012). An action theory $\mathcal{D}$ is *bounded* by a natural number b if, at every executable situation (i.e., reachable through a finite sequence of executable actions), the number of distinct object tuples occurring in the extension of each fluent of $\mathcal{D}$ is bounded by b. Thus, the interpretation of a fluent at every situation does not use more than b distinct object tuples, though these change from situation to situation and are collectively infinitely many.

The crux of the decidability result consists in the possibility of abstracting the infinite TS $T_{\delta_0,M} = \langle \Delta, Q, q_0, \rightarrow, L, Q_F \rangle$ into a *TS with a finite number of states* $F_{\delta_0,M} = \langle \Delta, Q^f, q_0^f, \rightarrow^f, L^f, Q_F^f \rangle$ that is bisimilar to $T_{\delta_0,M}$, by using Procedure 1.

Procedure 1 Construction of $F_{\delta_0,M}$.

$Q^f := \{q_0\}$; $q_0^f := q_0$; $\rightarrow^f := \emptyset$; $L^f(q_0) := L(q_0)$; $Q_F^f := \emptyset$;
if $(q_0 \in Q_F)$ **then** $Q_F^f := \{q_0\}$;
repeat
  pick $q = (\delta, \vec{x}, s) \in Q^f$;
  for all $(q' = (\delta', \vec{x}', s') \mid q \rightarrow q'$ in $T_{\delta_0,M})$
    if $(\exists q'' = (\delta', \vec{x}'', s'') \in Q^f \mid L(q') \sim L^f(q''))$ **then**
      $\rightarrow^f := \rightarrow^f \cup \{\langle q, q'' \rangle\}$;
    else $\{Q^f := Q^f \uplus \{q'\}$; $L^f(q') := L(q')$;
      $\rightarrow^f := \rightarrow^f \cup \{\langle q, q' \rangle\}\}$
    if $(q' \in Q_F)$ **then** $Q_F^f := Q_F^f \uplus \{q'\}$
until (transition relation $\rightarrow^f$ does not change any more)

Intuitively, $F_{\delta_0,M}$ is obtained through a visit of $T_{\delta_0,M}$, from q_0, by redirecting current transition $q \rightarrow q'$ to $q \rightarrow q''$, whenever q'' has already been visited and its label is isomorphic (wrt fluent and constant interpretations) to that of q'.

Note that, as a result of the construction above, $F_{\delta_0,M}$ cannot contain distinct states with isomorphic labels. Thus, since with a bounded number of distinct objects there exists only finitely many equivalence classes of isomorphic interpretations (called *isomorphism types*), it follows that $F_{\delta_0,M}$ contains only finitely many states. The boundedness of $\mathcal{D}$ also implies that checking whether two interpretations are isomorphic is decidable. Moreover, when checking whether $q \rightarrow q'$ or $q \in Q_F$, we do not need to construct $T_{\delta_0,M}$ explicitly. Indeed, for the (known) interpretation $L(q)$ of the state q currently visited, we have that: *(i)* the truth values of fluents after an action (as defined by successors state axioms) are fully determined by the truth values in the current state, i.e., by $L(q)$; and *(ii)* action types are finitely many. Thus, we can compute *Trans* and *Final* directly from $L(q)$ and the successor state axioms for each possible action type. Finally, evaluating FO formulas against $L(q)$ is decidable. Indeed, it can be reduced to evaluating a formula on the interpretation of fluents given by $L(q)$ (Libkin 2007), which contains finitely many (in fact, bounded) elements, completely disregarding the remaining (infinite) object domain. Hence, *each step of the procedure is computable*. For termination, we observe that the **for all** loop need not iterate over all (the infinitely many) states. Indeed, as discussed above, since isomorphic states behave in the same way, it is sufficient to consider only one representative per isomorphism type. As there are finitely many of them, the loop can be completed in a finite number of steps. Hence $F_{\delta_0,M}$ can be effectively (symbolically) constructed. It can be shown that $F_{\delta_0,M}$ is indeed bisimilar to $T_{\delta_0,M}$ which in turn is bisimilar to $T^o_{\delta_0,M}$. Using these results we prove that:

Theorem 6 *For every ConGolog program over a situation calculus bounded action theory $\mathcal{D}$ and every $\mu\mathcal{L}$ formula, checking $(\delta_0, \mathcal{D} \cup \mathcal{C}) \models \Phi$ is decidable.*

PROOF (SKETCH). Under complete information on S_0, all models of $\mathcal{D}$ are isomorphic wrt the interpretation of constants and fluents, and differ only on the number of objects (which needs to be at least countably infinite) in the object domain. All such models M generate TSs $T^o_{\delta_0,M}$ that are bisimilar. Thus, we can elect one of them, M', and exploit the fact that $T^o_{\delta_0,M'} \approx T_{\delta_0,M'} \approx F_{\delta_0,M'}$, together with bisimulation invariance (Theorem 4), to check Φ against $F_{\delta_0,M'}$, which is finite-state and can be checked with standard model checking techniques. Under incomplete information, since $\mathcal{D}_0$ is bounded, there are only finitely many isomorphically distinct types for models of $\mathcal{D}_0$ wrt the interpretation of constants and fluents. Hence, we can take a representative for each isomorphism type, and proceed as in the complete information case. □

Compiling Programs into Action Theories

Next we focus on *situation-determined* ConGolog *programs* (De Giacomo, Lespérance, and Muise 2012) for which the remaining program is *determined* by the resulting situation: $\mathit{SituationDetermined}(\delta, s) \doteq \forall s', \delta', \delta''.\mathit{Trans}^*(\delta, s, \delta', s') \wedge \mathit{Trans}^*(\delta, s, \delta'', s') \supset \delta' = \delta''$, where *Trans** denotes the reflexive transitive closure of *Trans*. For example, assuming all actions are executable, program $(a; b) \mid (a; c)$ is not situation-determined in situation S_0, as it is not possible to determine the remaining program in $do(a, S_0)$, whereas program $a; (b \mid c)$ is.

Compiling situation-determined programs. Let $\mathcal{D}$ be a situation calculus basic action theory and δ_0 a situation determined ConGolog program. We show how to compile δ_0 into a variant theory $\mathcal{D}_{\delta_0}$ of $\mathcal{D}$ that includes a fluent to store the program counter and environment, stepping through the finite set of possible remaining programs of δ_0 (which must belong to the syntactic closure Γ_{δ_0} of δ_0).

We introduce a new fluent $\mathit{PCEnv}(\delta, \vec{x}, s)$, where δ is the current program (counter value), $\vec{x}$ is a tuple of objects assigned to the pick variables, and s is the current situation. Its successor state axiom is as follows (see below regarding quantification on programs):

$$\mathit{PCEnv}(\delta', \vec{x}', do(a, s)) \equiv \exists \delta, \vec{x}.\mathit{PCEnv}(\delta, \vec{x}, s) \wedge \Phi_{Trans}(\delta, \vec{x}, s, \delta', \vec{x}', do(a, s)),$$

where $\Phi_{Trans}(\delta, \vec{x}, s, \delta', \vec{x}', do(a, s))$ is the uniform situation calculus formula equivalent to $\mathit{Trans}(\delta, \vec{x}, s, \delta', \vec{x}', do(a, s))$ discussed before. We maintain the same successor state axioms for all the other fluents appearing in the original theory. As for the precondition axioms, we replace them with the following one:

$$\mathit{Poss}(a, s) \equiv \exists \delta, \delta', \vec{x}, \vec{x}'.\mathit{PCEnv}(\delta, \vec{x}, s) \wedge \Phi_{Trans}(\delta, \vec{x}, s, \delta', \vec{x}', do(a, s)),$$

which states that an action is possible *if the current program can do it* (in the *Trans* abbreviation we still use the right-hand side of the original precondition axioms).

The initial situation description is as before with the addition of fact sentence $\mathit{PCEnv}(\delta_0, \vec{x_0}, S_0)$, for some arbitrary tuple of object values $\vec{x_0}$. Finally, we introduce an abbreviation to denote whether a situation s is final, i.e., the situation-determined program δ_0 can be considered terminated in s:

$$\mathit{Final}(s) \doteq \exists \delta, \vec{x}.\mathit{PCEnv}(\delta, \vec{x}, s) \wedge \Phi_{Final}(\delta, \vec{x}, s)$$

where $\Phi_{Final}(\delta, \vec{x}, s)$ is the uniform situation calculus formula equivalent to $\mathit{Final}(\delta, \vec{x}, s)$. Note that when we quantify over programs, e.g., in the above formulas, we are actually quantifying over the finite domain Γ_{δ_0}, hence we can actually replace such quantifications with finite disjunctions (for $\exists$) and conjunctions (for $\forall$).

We can prove that the new basic action theory $\mathcal{D}_{\delta_0}$—the compilation of situation-determined ConGolog program δ_0 into $\mathcal{D}$—generates exactly the same configurations as those generated by the program δ_0 running over $\mathcal{D}$. In fact, using the results in (De Giacomo, Lespérance, and Patrizi 2012), we can verify a $\mu\mathcal{L}$ property Φ of program δ_0 running over the theory $\mathcal{D}$ by verifying Φ directly over the compiled action theory $\mathcal{D}_{\delta_0}$ (interpreting *Final* as abbreviation above).

Theorem 7 *Let $\mathcal{D}$ be a basic action theory, δ_0 a situation-determined ConGolog program, and $\mathcal{D}_{\delta_0}$ as above. Then, for every $\mu\mathcal{L}$ formula Φ: $(\delta_0, \mathcal{D} \cup \mathcal{C}) \models \Phi$ iff $\mathcal{D}_{\delta_0} \models \Phi$.*

In (De Giacomo, Lespérance, and Patrizi 2012), $\mu\mathcal{L}$ formulas are interpreted over the tree of executable situations (e.g., $\langle - \rangle \Phi$ holds in a situation if there exists an executable action in it such that Φ holds afterwards). But, if the program is situation-determined, then there is a unique program configuration for each executable situation in a model of the compiled theory, so a $\mu\mathcal{L}$ formula holds in a situation iff it holds in the associated configuration.

Non-situation-determined programs. The above compilation technique actually works to a certain extent also for non-situation-determined programs. For those programs, Theorem 7 fails, since one cannot reconstruct the actual configuration that the program is in at each step—only the "possible" ones can be obtained. For example, consider the program $\delta_0 \doteq (a; a) \mid (a; b)$, where primitive action a causes fluent P to become true while action b makes it false. Consider the $\mu\mathcal{L}$ property $\Phi \doteq \langle - \rangle [-] P \wedge \langle - \rangle [-] \neg P$ (i.e., P can be forced true and can be forced false). This property does hold for δ_0, i.e., $(\delta_0, \mathcal{D} \cup \mathcal{C}) \models \Phi$. To see this, observe that if we perform a step using the left branch of δ_0, the remaining program is a, and after performing one more step (another a), P must hold. If instead we perform a step using the right branch, the remaining program is b, and after doing one more step, $\neg P$ must hold. However for the compiled action theory, we have $\mathcal{D}_{\delta_0} \not\models \Phi$. This is because both $\mathit{PCEnv}(a, \vec{x_0}, do(a, S_0))$ and $\mathit{PCEnv}(b, \vec{x_0}, do(a, S_0))$ are true in $\mathcal{D}_{\delta_0}$, so neither $[-]P$ nor $[-]\neg P$ hold in $do(a, S_0)$. In fact, this example shows that we would need nondeterministic effects to specify the remaining program in a given execution, whereas Reiter's situation calculus requires actions to be deterministic.

Nevertheless, the compiled theory $\mathcal{D}_{\delta_0}$ can still be used to check properties such as whether a given action sequence/situation *can* be produced in an execution of the program. In particular, consider $\mathit{Do}(\delta_0, S_0, s) \doteq \exists \delta.\mathit{Trans}^*(\delta_0, S_0, \delta, s) \wedge \mathit{Final}(\delta, s)$ as defined in (De Giacomo, Lespérance, and Levesque 2000). Then we have:

Theorem 8 *Let $\mathcal{D}$ by a basic action theory, δ_0 a ConGolog program, $\mathcal{C}$ the original axioms for Trans and Final, and $\mathcal{D}_{\delta_0}$ the compiled theory as above (with Poss renamed in the original theory to avoid clashing). Then, the following are logically implied by $\mathcal{D} \cup \mathcal{C} \cup \mathcal{D}_{\delta_0}$:*

$$\mathit{Do}(\delta_0, S_0, s) \equiv \mathit{Executable}(s) \wedge \mathit{Final}(s);$$

$$\mathit{Trans}^*(\delta_0, S_0, \hat{\delta}, s) \equiv$$
$$\exists \delta, \vec{x}.\mathit{Executable}(s) \wedge \mathit{PCEnv}(\delta, \vec{x}, s) \wedge \hat{\delta} = \delta[\vec{x}];$$

where $\mathit{Executable}(s)$ states that s is an executable situation (Reiter 2001). The first equivalence gives a characterization of ConGolog's standard offline semantics, while the second can be used to check whether a sequence of actions amounts to a partial execution of a program, as often needed in plan conformance checking (Goultiaeva and Lespérance 2007).

Discussion and Conclusion

There has been significant interest in reasoning about and verifying agent programs, such as Shapiro, Lespérance, and Levesque (2002; 2010)'s CASLve verification environment for multi-agent ConGolog programs, Alechina et al. (2010)'s PDL-like logic for SimpleAPL programs, Bordini et al. (2003)'s and Yadav and Sardina (2012)'s model-checking frameworks for BDI programs, and the work in (Claßen and Lakemeyer 2008; Claßen et al. 2014; De Giacomo, Lespérance, and Pearce 2010; Sardina and De Giacomo 2009) based on ConGolog programs "characteristic graphs". [4] However, these approaches generally impose important restrictions, such as propositional agents and/or simple programs (e.g., with pick variables ranging over finite domains), or resort to verification via theorem proving and fixpoint approximation with no decidability guarantees.

Unlike the original ConGolog semantics, our new semantics avoids the use of a complex encoding of programs as terms. This allows us to compile programs away into standard basic action theories when these are situation-determined. This is similar to Fritz, Baier, and McIlraith (2008), who showed how to compile arbitrary programs as Petri-nets (plus unbounded stacks for recursion), and encode these into a basic action theory. Also it is related to Lin (2014), who recently showed that programs can be compiled into action theories using an extra situation parameter. Both proposals yield a correctness result for *Do*, like our Theorem 8, but not for temporal verification, i.e., our Theorem 7, though the notion of situation-determined programs could be used to get one. Nor do they generate action theories that are bounded, so one cannot use (De Giacomo, Lespérance, and Patrizi 2012) to get decidability results.

[4] Interestingly, these graphs include an uninstantiated version of pick operators, which are then instantiated in the labeling model-checking-like verification algorithm.

References

Alechina, N.; Dastani, M.; Khan, F.; Logan, B.; and Meyer, J.-J. 2010. Using theorem proving to verify properties of agent programs. In *Specification and Verification of Multi-agent Systems*. Springer US. 1–33.

Baier, C., and Katoen, J.-P. 2008. *Principles of model checking*. MIT Press.

Bordini, R. H.; Fisher, M.; Pardavila, C.; and Wooldridge, M. 2003. Model checking AgentSpeak. In *Proc. of AAMAS*, 409–416. ACM Press.

Claßen, J., and Lakemeyer, G. 2008. A logic for non-terminating Golog programs. In *Proc. of KR*, 589–599.

Claßen, J.; Liebenberg, M.; Lakemeyer, G.; and Zarrieß, B. 2014. Exploring the boundaries of decidable verification of non-terminating Golog programs. In *Proc. of AAAI*, 1012–1019.

De Giacomo, G.; Lespérance, Y.; and Levesque, H. J. 2000. ConGolog, a concurrent programming language based on the situation calculus. *Artificial Intelligence* 121(1–2):109–169.

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

De Giacomo, G.; Lespérance, Y.; and Patrizi, F. 2012. Bounded situation calculus action theories and decidable verification. In *Proc. of KR*.

De Giacomo, G.; Lesperance, Y.; and Patrizi, F. 2016. Bounded situation calculus action theories. *Artificial Intelligence*. To appear. Preliminary version available at `http://arxiv.org/abs/1509.02012`.

De Giacomo, G.; Lespérance, Y.; and Pearce, A. R. 2010. Situation calculus based programs for representing and reasoning about game structures. In *Proc. of KR*.

De Giacomo, G.; Ternovskaia, E.; and Reiter, R. 1997. Non-terminating processes in the situation calculus. In *Proc. of the AAAI'97 Workshop on Robots, Softbots, Immobots: Theories of Action, Planning and Control*.

Emerson, E. A. 1996. Model checking and the mu-calculus. In *Descriptive Complexity and Finite Models*, 185–214.

Fritz, C.; Baier, J. A.; and McIlraith, S. A. 2008. ConGolog, Sin Trans: Compiling ConGolog into basic action theories for planning and beyond. In *Proc. of KR*, 600–610.

Goultiaeva, A., and Lespérance, Y. 2007. Incremental plan recognition in an agent programming framework. In *Proc. of PAIR*.

Levesque, H. J.; Reiter, R.; Lesperance, Y.; Lin, F.; and Scherl, R. 1997. GOLOG: A logic programming language for dynamic domains. *J. of Logic Programming* 31:59–84.

Libkin, L. 2007. Embedded finite models and constraint databases. In *Finite Model Theory and Its Applications*. Springer.

Lin, F. 2014. A first-order semantics for Golog and ConGolog under a second-order induction axiom for situations. In *Proc. of KR*, 39–46.

Lomuscio, A.; Qu, H.; and Raimondi, F. 2009. MCMAS: A model checker for the verification of multi-agent systems. In *Proc. CAV*, 682–688.

McCarthy, J., and Hayes, P. J. 1969. Some philosophical problems from the standpoint of Artificial Intelligence. *Machine Intelligence* 4:463–502.

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

Sardina, S., and De Giacomo, G. 2009. Composition of ConGolog programs. In *Proc. of IJCAI*, 904–910.

Shapiro, S.; Lespérance, Y.; and Levesque, H. J. 2002. The cognitive agents specification language and verification environment for multiagent systems. In *Proc. of AAMAS*, 19–26.

Shapiro, S.; Lespérance, Y.; and Levesque, H. 2010. The cognitive agents specification language and verification environment. In *Specification and Verification of Multi-agent Systems*. Springer US. 289–315.

Stirling, C. 2001. *Modal and Temporal Properties of Processes*. Springer.

Yadav, N., and Sardina, S. 2012. Reasoning about BDI agent programs using ATL-like logics. In *Proc. of JELIA*, volume 7519 of *LNCS*, 437–449. Springer.