Machine-checked Verification of Cognitive Agents

Alexander Birch Jensen[a]

DTU Compute - Department of Applied Mathematics and Computer Science, Technical University of Denmark, Richard Petersens Plads, Building 324, DK-2800 Kongens Lyngby, Denmark

Keywords: Cognitive Agent Systems, Formal Verification, Isabelle/HOL.

Abstract: The ability to demonstrate reliability is an important aspect in deployment of software systems. This applies to cognitive multi-agent systems in particular due to their inherent complexity. We are still pursuing better approaches to demonstrate their reliability. The use of proof assistants and theorem proving has proven itself successful in verifying traditional software programs. This paper explores how to apply theorem proving to verify agent programs. We present our most recent work on formalizing a verification framework for cognitive agents using the proof assistant Isabelle/HOL.

1 INTRODUCTION

In the deployment of software systems we are often required to provide assurance that the systems operate reliably. Cognitive multi-agent systems (CMAS) consist of agents which incorporate cognitive concepts such as beliefs and goals. Their dedicated programming languages enable a compact representation of complex decision-making mechanisms. For CMAS in particular the complexity of these systems often exceeds that of procedural programs (Winikoff and Cranefield, 2014). This calls for us to develop approaches that tackle the issue of demonstrating reliability for CMAS.

The current landscape reveals testing, debugging and formal verification such as model checking as the primary approaches to demonstrating reliability for CMAS (Calegari et al., 2021). This landscape has primarily been dominated by work on model checking techniques and exploration of better testing methods (Jongmans et al., 2010; Bordini et al., 2004). However, if we take a deeper look, we find promising work on theorem proving approaches (Alechina et al., 2010; Shapiro et al., 2002). Combining testing and formal verification has also been suggested as their individual strengths and weaknesses complement each other well (Winikoff, 2010).

A proof assistant is an interactive software tool which assists the user in developing formal proofs, often by providing powerful proof automation. State-of-the-art proof assistants have proven successful in verification of software and hardware systems (Ringer et al., 2019), but their potential use in the context of CMAS is yet to be uncovered. The major strength of a proof assistant is that every proof can be trusted as it is checked by the proof assistant's kernel. Exactly which proof assistant someone prefers is highly subjective. In this paper our work is carried out in the Isabelle/HOL proof assistant (Nipkow et al., 2002).

In the present paper, we formalize existing work on a verification framework for agents programmed in the language GOAL (de Boer et al., 2007; Hindriks and Dix, 2014; Hindriks et al., 2001; Hindriks, 2009). The work we present is an extension of previous work (Jensen, 2021a; Jensen et al., 2021; Jensen, 2021b; Jensen, 2021c). Section 2 elaborates on the contributions of this previous work.

The contribution of this paper is threefold:

- we present and describe how a temporal logic can be used to prove properties of agents specified in the agent logic,
- we demonstrate how this is achieved in practice by proving a correctness property for a simple example program, and finally
- we improve upon our previous work with a focus on concisely delivering the main results of the formalization in a format more friendly to those unfamiliar with proof assistants.

The paper is structured as follows. Section 2 places this work into context with related work. Section 3 provides an overview of the structure of the formalization. Section 4 describes the GOAL lan-

[a] https://orcid.org/0000-0002-7298-2133

Jensen, A.
Machine-checked Verification of Cognitive Agents.
DOI: 10.5220/0010838700003116
In *Proceedings of the 14th International Conference on Agents and Artificial Intelligence (ICAART 2022) - Volume 1*, pages 245-256
ISBN: 978-989-758-547-0; ISSN: 2184-433X

guage and its formalization. Section 5 describes how to specify agents in a Hoare logic and how this translates to the low-level semantics. Section 6 introduces a temporal logic on top of the agent logic and describes how this can be used to prove temporal properties of agents. Section 7 demonstrates the use of the verification framework by means of a simple example program. Finally, Section 8 discusses limitations and further work and concludes on the paper.

This paper contains a number of Isabelle/HOL listings from the formalization source files referenced below, all of which have been verified. In a few cases, their presentation in this paper has been altered in minor ways for the sake of readability. For absolute precision, please refer to the full source which is publicly available online:

https://people.compute.dtu.dk/aleje/#public

2 RELATED WORK

The ability to demonstrate reliability of agent systems and autonomous systems has emerged as an important topic, perhaps sparked by the reemergence of a broader interest in AI. The following quote from a recent seminar establishes this point (Dix et al., 2019):

> The aim of this seminar was to bring together researchers from various scientific disciplines, such as software engineering of autonomous systems, software verification, and relevant subareas of AI, such as ethics and machine learning, to discuss the emerging topic of the reliability of (multi-)agent systems and autonomous systems in particular. The ultimate aim of the seminar was to establish a new research agenda for engineering reliable autonomous systems.

The work presented in this paper embraces this perspective as we apply software verification techniques using the Isabelle/HOL proof assistant to verify agents programmed in the language GOAL.

The main approaches to agent verification can be categorized as testing, model checking and formal verification. Our approach falls into the latter category.

Testing is in general more approachable as it does not necessarily require advanced knowledge of verification systems. The main drawback is that it is non-exhaustive as tests explore a finite set of inputs and behaviors. In (Koeman et al., 2018), the authors present a testing framework that automatically detects failures in cognitive agent programs. Failures are detected at run-time and are enabled by a specification of test conditions. While the work shows good promise, the authors note that more work needs to be dedicated to localizing failures.

Model checking is a technique where a system description is analyzed with respect to a property. It is then checked that all possible executions of the system satisfy this property. Model checking is currently considered the state-of-the-art approach to an exhaustive demonstration of reliability for CMAS. In (Dennis and Fisher, 2009; Dennis et al., 2012), we are presented with a verification system for agent programming languages based on the belief-desire-intention model. In particular, an abstract layer maps the semantics of the programming languages to a model checking framework thus enabling their verification. The framework as a whole consists of two components: an Agent Infrastructure Layer (AIL) and an Agent Java Pathfinder (AJPF) which is a version of the Java Pathfinder (JPF) model checker (Java Pathfinder, 2021). The work contains a few example implementations. In particular, the semantics of the agent programming GOAL is mapped to AIL. While this remains purely speculative, it would be interesting to explore whether our formal verification approach could be generalized to an abstract agent infrastructure such as AIL.

Formal verification is a technique where properties are proved with respect to a formal specification using methods of formal mathematics. Because of the amount of work involved in conducting formal proofs and their susceptibility to human errors, such techniques are often applied with a theorem proving software tool such as a proof assistant. If the proof assistant is trusted, this in turn eliminates human errors in proofs while automation greatly reduces the amount of work required. The main drawback is the expert knowledge required to work with such software systems. In (Gomes et al., 2017), the authors verify correctness for a class of algorithms which provides consistency guarantees for shared data in distributed systems. (Ringer et al., 2019) provides a survey of the literature for engineering formally verified software. We have not been able to find related work that combines verification of agents with the use of proof assistants. We hope that the work presented in this paper can contribute to breaking the ice and stimulating interest in the potential of proof assistants for verification of agents.

We finally relate the contributions of this paper to our previous work. In (Jensen, 2021a), we outlined how to mechanically transform GOAL program code into the agent logic for GOAL formalized in this paper. In the present paper, we have not pursued the idea of formalizing this transformation in Isabelle/HOL.

The example we present in Section 7 is based on actual program code, but has been significantly simplified for the sake of readability to hide programming quirks that do not translate elegantly and because our focus is on the Isabelle/HOL formalization. In (Jensen et al., 2021), we argued for the use of theorem proving to verify CMAS and presented early work towards formalizing the framework. In (Jensen, 2021b), we go into the details of our early work on the formalization. In particular, there is more attention on the intricate details of formalizing the mental states of agents. Due to the sheer complexity and volume of the formal aspects we need to cover in the present paper, we cannot here dedicate attention to every detail. In (Jensen, 2021c), the emphasis is on formalizing actions and their effect on the mental state, and how to reason about such actions. The present paper is a formalization of the verification framework in its entirety and covers the most important aspects.

3 FORMALIZING GOAL IN ISABELLE/HOL

The framework formalization is around 2400 lines of code in total (178 KB) and loads in a few seconds. The BW4T example formalization is around 1100 lines of code in total (95 KB) and loads in about a minute. We want to emphasize that the example has not been optimized. With proper optimization, we expect the loading time for the example to be just a few seconds as well. The loading times have been tested on a modern laptop.

The *Gvf_Logic* theory sets up a framework for propositional classical logic, including a definition of semantics and a sequent calculus proof system. The *Gvf_Mental_States* theory introduces the definition of mental states and a logic for reasoning about mental states which is defined as a special case of propositional logic. The *Gvf_Actions* theory introduces concepts of agents, actions, transitions and traces. The *Gvf_Hoare_Logic* theory introduces a Hoare logic for reasoning about the (non-)effects of actions on mental states. The *Gvf_Agent_Specification* theory sets up a framework for specifying an agent using Hoare triples and a Hoare triple proof system. The *Gvf_Temporal_Logic* theory introduces a temporal logic to reason about temporal properties of specified agents. Finally, the *Gvf_Example_BW4T* theory proves the correctness of a simple example agent program.

Figure 1 illustrates the high-level components of the formalization and how they interact.

4 LOGIC FOR AGENTS

In this section, we describe a formalization of the GOAL language.

From the *Gvf_Logic* theory, we import a number of key definitions and proofs:

- $'a\ \Phi_P$: Datatype for propositional logic formulas with an arbitrary type $'a$ of atoms.
- $\models_P$: Semantic consequence for $'a\ \Phi_P$.
- $\vdash_P$: Sequent calculus proof system for $'a\ \Phi_P$.

The meaning of $\Gamma \models_P \Delta$ is that the truth of all formulas in the list of formulas Γ implies that at least one formula is true in the list of formulas Δ.

The standard form of propositional logic, using string symbols for atoms, is denoted by the derived type Φ_L. Here, the semantics are defined over a model which maps string symbols to truth values.

Theorem 4.1 (Soundness of $\vdash_P$)**.** *The proof system $\vdash_P$ is sound with respect to the semantics of propositional logic formulas $\models_P$:*

$\Gamma \vdash_P \Delta \Longrightarrow \Gamma \models_P \Delta$

Proof. By induction on the proof rules of $\vdash_P$. □

4.1 Mental States of Agents

We are now interested in formalizing a state-based semantics for GOAL. The state of an agent program is determined by the mental state of the agent. A mental state is modelled as two lists of formulas: a belief base and a goal base. The belief base formulas describe what the agents believe to be true while the goal base formulas describe what the agents want to achieve.

type-synonym *mental-state* $= (\Phi_L\ \mathit{list} \times \Phi_L\ \mathit{list})$

The original formulation describes these as sets, but we have chosen to work with lists. There are advantages and disadvantages to either approach.

Usually, we distinguish between the beliefs of the agent and the static knowledge about its world. In this setup, static knowledge is incorporated into the beliefs.

Due to limitations of the simple type system, we need a definition to formulate the special properties of a proper mental state:

definition *is-mental-state* :: *mental-state* $\Rightarrow$ *bool* (∇) **where**
$\nabla\ M \equiv \mathit{let}\ (\Sigma, \Gamma) = M\ \mathit{in}\ \neg\ \Sigma \models_P \bot\ \wedge$
$(\forall \gamma \in \mathit{set}\ \Gamma.\ \neg\ \Sigma \models_P \gamma \wedge \neg \models_P (\neg\ \gamma))$

The above states that the belief base should be consistent, that no goal should be entailed by the belief base and that goals should be achievable.

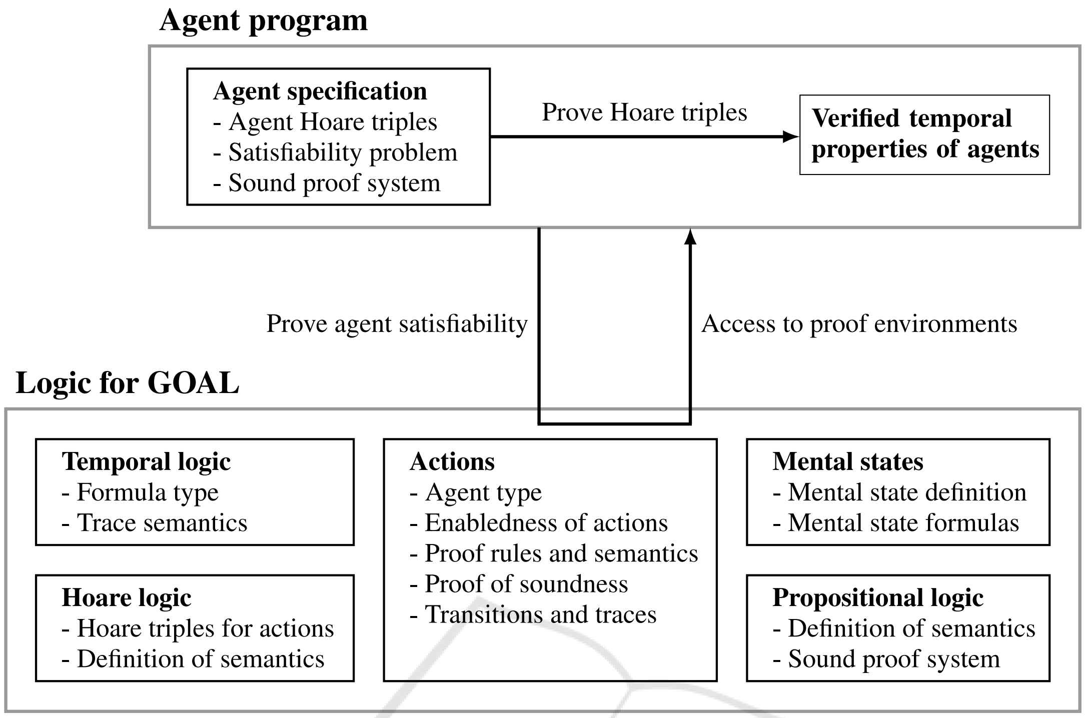

Figure 1: Visualization of the verification framework components.

The special belief and goal modalities, B and G respectively, enable the agent's introspective properties as defined by their semantics:

$(\Sigma, \text{-}) \models_M B\,\Phi = (\Sigma \models_P \Phi)$
$(\Sigma, \Gamma) \models_M G\,\Phi =$
$\quad (\neg\, \Sigma \models_P \Phi \wedge (\exists \gamma {\in} set\, \Gamma.\ \models_P (\gamma \longrightarrow \Phi)))$

The language of mental state formulas Φ_M can be perceived of as a special form of propositional logic where an atomic formula is either a belief or goal modality. This approach allows us to exploit the type variable $'a$ for atoms where the model is defined via the semantics of the belief and goal modalities.

We extend the proof system for propositional logic $\vdash_P$ with additional rules and axioms for the belief and goal modalities given in Table 1 to obtain a proof system for mental state formulas $\vdash_M$.

Table 1: Properties of beliefs and goals.

R_M-B	$\models_P \Phi \Longrightarrow \vdash_M B\,\Phi$
$A1_M$	$\vdash_M B\,(\Phi \longrightarrow \psi) \longrightarrow B\,\Phi \longrightarrow B\,\psi$
$A2_M$	$\vdash_M \neg\,(B\,\bot)$
$A3_M$	$\vdash_M \neg\,(G\,\bot)$
$A4_M$	$\vdash_M B\,\Phi \longrightarrow \neg\,(G\,\Phi)$
$A5_M$	$\models_P (\Phi \longrightarrow \psi) \Longrightarrow$
	$\vdash_M \neg\,(B\,\psi) \longrightarrow G\,\Phi \longrightarrow G\,\psi$

Theorem 4.2 (Soundness of $\vdash_M$). *The proof system $\vdash_M$ is sound with respect to the semantics of mental state formulas $\models_M$ for any proper mental state* (∇ M)*:*

$\nabla\, M \Longrightarrow \vdash_M \Phi \Longrightarrow M \models_M \Phi$

Proof. By induction on the proof rules and using Theorem 4.1. □

Notice that we assume $\nabla\, M$ which is necessary for the proof rules to hold semantically. Furthermore, in the soundness theorem for $\vdash_M$, there are no premises on the left-hand side. This is deliberate, as we want to match the formal notation for the semantics of mental states that takes on the left-hand side a mental state, i.e. $M \models_M \Phi$. We can solve this problem by incorporating premises in the target formula Φ by the use of implication, i.e. for premises $p_1,\ p_2,\ \ldots$ we have for the semantics $M \models_M p_1 \longrightarrow p_2 \longrightarrow \ldots \longrightarrow \Phi$.

4.2 Selecting and Executing Actions

In order to formalize a state-based semantics for GOAL, we need a notion of transitions between mental states of agents. We assume that agents themselves are the only ones capable of changing their mental state. As such, we can model transitions where an action executed in a given mental state has a predetermined outcome.

The low-level actions of GOAL are the basic actions. A basic action may be one of the two GOAL built-in actions for directly manipulating the goal base, or a user-defined action which is specific to the agent program:

datatype *cap* = *basic Bcap* | *adopt* Φ_L | *drop* Φ_L

The type *Bcap* is some type for identifying actions, in our case the type for strings.

An action transforms the mental state in which it is executed. In fact, for any action different from *adopt* and *drop* it is sufficient to consider only a transformation of the belief base as the effect on the goal base can be derived from the semantics of GOAL.

The transformation of mental states is partially defined by a function $\mathcal{T}$ of the type *Bcap* ⇒ *mental-state* ⇒ Φ_L *list option*: given a mental state and an action identifier, optionally a new belief base is returned. That $\mathcal{T}$ only optionally returns a new belief base is a result of the fact that a basic action may not be enabled in a given state. Notice that $\mathcal{T}$ is only concerned with the agent specific actions, namely those with an effect on the belief base.

The full effect on the mental state of executing a basic action is captured by a function $\mathcal{M}$:

fun *mental-state-transformer* :: *cap* ⇒ *mental-state* ⇒ *mental-state option* ($\mathcal{M}$) **where**
$\mathcal{M}$ (*basic n*) (Σ, Γ) = (*case* $\mathcal{T}$ *n* (Σ, Γ) *of*
  Some $\Sigma' \Rightarrow$ *Some* $(\Sigma', [\psi \leftarrow \Gamma.\ \neg\, \Sigma' \models_P \psi])$
  | - ⇒ *None*)
$\mathcal{M}$ (*drop* Φ) (Σ, Γ) = *Some* $(\Sigma, [\psi \leftarrow \Gamma.\ \neg\, [\psi] \models_P \Phi])$
$\mathcal{M}$ (*adopt* Φ) (Σ, Γ) = (*if* $\neg \models_P (\neg\, \Phi) \wedge \neg\, \Sigma \models_P \Phi$ *then*
  Some $(\Sigma$, *List.insert* $\Phi\ \Gamma)$ *else None*)

The case for agent specific actions captures the default commitment strategy where goals are only removed once achieved.

On top of basic actions we introduce the notion of a conditional action. A conditional action consists of a basic action and a condition φ, denoted $\varphi \triangleright do\ a$. This condition is specified as a mental state formula which is evaluated in a given mental state. The action can only be executed if the condition is met. As such, the enabledness of a conditional action depends on both $\mathcal{T}$ and φ. The set of conditional actions for an agent is denoted Π.

Combining the notion of conditional actions and a mental state transformer gives rise to the concept of a transition between two states due to the execution of an action, captured by the following definition:

definition *transition* :: *mental-state* ⇒ *cond-act* ⇒ *mental-state* ⇒ *bool* (- →- -) **where**
$M \rightarrow b\ M' \equiv b \in \Pi \wedge M \models_M (fst\ b) \wedge$
  $\mathcal{M}\ (snd\ b)\ M = Some\ M'$

In the formalization, a conditional action *b* is a tuple and *fst b* gives its condition while *snd b* gives its basic action. The definition states that a transition $M \rightarrow (\varphi \triangleright do\ a)\ M'$ exists if $\varphi \triangleright do\ a$ is in Π, the condition φ holds in *M* and *M′* is the resulting state.

A run of an agent program can be understood as a sequence of mental states interleaved with conditional actions $(M_0, b_0, M_1, b_1, \ldots, M_i, b_i, M_{i+1}, \ldots)$. We do not consider the use of a stop criterion and instead assume the program to continue indefinitely. It is not a criterion for scheduled actions (those in the trace) to necessarily be enabled and thus executed. In such cases, we have $M_i = M_{i+1}$. We call a possible run of the agent a trace. The *codatatype* command allows for a coinductive datatype:

codatatype *trace* = *Trace mental-state cond-act* × *trace*

We further define functions *st-nth* and *act-nth* which give the *i*-th state and conditional action of a trace, respectively.

Analogous to mental states, our definition of a trace includes all elements of the simple type, so we need a definition:

definition *is-trace* :: *trace* ⇒ *bool* **where**
 is-trace s ≡ ∀ *i*. (*act-nth s i*) ∈ Π ∧
  (((*st-nth s i*) →(*act-nth s i*) (*st-nth s* (*i*+*1*))) ∨
  ¬(∃*M*. (*st-nth s i*) →(*act-nth s i*) *M*) ∧
   (*st-nth s i*) = (*st-nth s* (*i*+*1*)))

The definition requires that for all M_i, b_i, M_{i+1} either a transition $M_i \rightarrow b_i\ M_{i+1}$ exists or the action is not enabled and $M_i = M_{i+1}$. The definition makes it explicit that if a transition exists from M_i then it is to M_{i+1}.

Figure 2 illustrates the set of all traces and highlights a single trace.

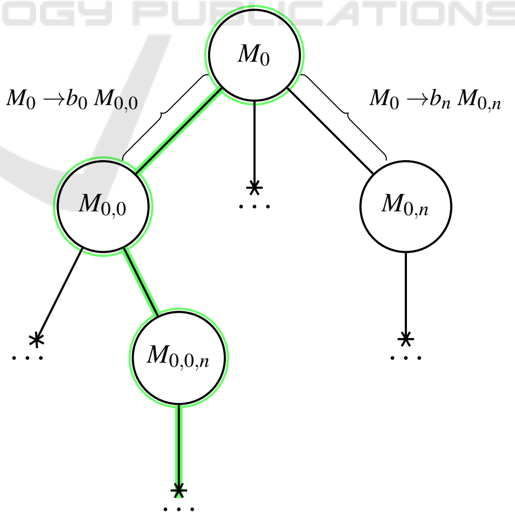
M_0
$M_0 \rightarrow b_0\ M_{0,0}$
$M_0 \rightarrow b_n\ M_{0,n}$
$M_{0,0}$
$M_{0,n}$
$M_{0,0,n}$
...
...
...
...

Figure 2: Visualization of the set of all traces, highlighting a single trace.

For any scheduling of actions by the agent, we assume *weak fairness*. Consequently, we have that the traces are *fair*:

definition *fair-trace* :: *trace* ⇒ *bool* **where**

$\textit{fair-trace}\ s \equiv \forall\ b \in \Pi.\ \forall i.\ \exists\ j > i.\ \textit{act-nth}\ s\ j = b$

At any point in a fair trace, for any action there always exists a future point where it is scheduled for execution. The meaning of an agent is defined as the set of all possible fair traces starting from a predetermined initial state M_0:

definition $\textit{Agent} :: \textit{trace set}$ **where**
$\textit{Agent} \equiv \{s\ .\ \textit{is-trace}\ s \land \textit{fair-trace}\ s \land \textit{st-nth}\ s\ 0 = M_0\}$

With the newly introduced terminology, it becomes rather straightforward to express the semantics of the new components in formulas regarding enabledness:

$M \models_E (\textit{enabled-basic}\ a) = (a \in \textit{Cap} \land \mathcal{M}\ a\ M \neq \textit{None})$
$M \models_E (\textit{enabledond}\ b) = (\exists M'.\ (M \rightarrow b\ M'))$

Here, *Cap* is the set of the agent's basic actions. For the second case, a similar check is built-in to the transition definition.

We introduce the shorthand notation $\varphi[s\ i]$ for evaluating the mental state formula φ in the i-th state of a trace s.

An extended proof system $\vdash_E$ is obtained by including the rules of Table 2 which state syntactic properties of enabledness.

Table 2: Enabledness of actions.

$E1_E$	$(\varphi \triangleright do\ a) \in \Pi \Longrightarrow$ $\vdash_E \textit{enabled}\ (\varphi \triangleright do\ a) \longleftrightarrow (\varphi \land \textit{enabledb}\ a)$
$E2_E$	$\vdash_E \textit{enabledb}\ (\textit{drop}\ \Phi)$
$R3_E$	$\neg \models_P (\neg\, \Phi) \Longrightarrow$ $\vdash_E \neg\ (B_E\ \Phi) \longleftrightarrow \textit{enabledb}\ (\textit{adopt}\ \Phi)$
$R4_E$	$\models_P (\neg\, \Phi) \Longrightarrow \vdash_E \neg\ (\textit{enabledb}\ (\textit{adopt}\ \Phi))$
$R5_E$	$\forall M.\ \nabla M \longrightarrow \mathcal{T}\ a\ M \neq \textit{None} \Longrightarrow$ $\vdash_E \textit{enabledb}\ (\textit{basic}\ a)$

Theorem 4.3 (Soundness of $\vdash_E$)**.** *The proof system $\vdash_E$ is sound with respect to the semantics of mental state formulas including enabledness $\models_E$ for any proper mental state (∇ M):*

$\nabla M \Longrightarrow \vdash_E \varphi \Longrightarrow M \models_E \varphi$

Proof. By induction on the proof rules and using Theorem 4.2. □

4.3 Hoare Logic for GOAL

To reason about transition steps due to execution of actions, we set up a specially tailored Hoare logic. By means of Hoare triples, we can specify the effect of executing an action using mental state formulas as pre- and postconditions. We distinguish between Hoare triples for basic actions and conditional actions, but as becomes apparent later, there is a close relationship between the two. We introduce the usual Hoare triple notation: $\{\varphi\}\ a\ \{\psi\}$ for basic actions and $\{\varphi\}\ \upsilon \triangleright do\ b\ \{\psi\}$ for conditional actions. The notation masks two constructors for a simple datatype *hoare-triple* consisting of two formulas and a basic or conditional action.

We now give the semantics of Hoare triples. The pre- and postconditions can only be evaluated given a current mental state. However, for the definition of the semantics we quantify over all mental states. In other words, a Hoare triple is only true if it holds at any point in the agent program:

$\models_H \{\ \varphi\ \}\ a\ \{\ \psi\ \} = (\forall M.$
$\quad (M \models_E \varphi \land (\textit{enabledb}\ a) \longrightarrow \textit{the}\ (\mathcal{M}\ a\ M) \models_M \psi) \land$
$\quad (M \models_E \varphi \land \neg(\textit{enabledb}\ a) \longrightarrow M \models_M \psi))$
$\models_H \{\ \varphi\ \}\ (\upsilon \triangleright do\ b)\ \{\ \psi\ \} = (\forall\ s \in \textit{Agent}.\ \forall i.$
$\quad (\varphi[s\ i] \land (\upsilon \triangleright do\ b) = (\textit{act-nth}\ s\ i) \longrightarrow \psi[s\ (i{+}1)])$

The first case is for basic actions. In any state, if the precondition is true, the postcondition should hold in the state obtained by executing the action a. Otherwise, it should hold in the same state as the action is not enabled and thus not executed. The second case is for conditional actions. The case is analogous, but the enabledness of the action, and thus whether the successor state is unchanged, is implicit due to the definition of traces.

Lemma 4.4 (Relation between Hoare triples for basic and conditional actions)**.**

$\models_H \{\ \varphi \land \psi\ \}\ a\ \{\ \varphi'\ \} \Longrightarrow$
$\quad \forall\ s \in \textit{Agent}.\ \forall i.\ ((\varphi \land \neg\psi) \longrightarrow \varphi')[s\ i] \Longrightarrow$
$\quad \models_H \{\ \varphi\ \}\ (\psi \triangleright do\ a)\ \{\ \varphi'\ \}$

Proof. We need to show $\varphi'[s\ (i{+}1)]$. We can assume $(\varphi[s\ i] \land (\upsilon \triangleright do\ b) = (\textit{act-nth}\ s\ i)$. We then distinguish between the cases of whether $\psi[s\ i]$ holds. If it holds, the basic action Hoare triple can be used to infer $\varphi'[s\ (i{+}1)]$. If it does not hold, the result follows from the second assumption as the action a is not enabled and thus *st-nth s i = st-nth (i+1)*. □

Lemma 4.4 is used to prove Theorem 5.3.

5 SPECIFICATION OF AGENTS

In this section, we are concerned with the specification of agent programs using Hoare triples and finding a Hoare system to be able to derive Hoare triples for the specified agent program. Because the low-level semantics of GOAL is not based on Hoare logic, we need to show that the low-level semantics can be derived from an agent specification based on Hoare triples. More specifically, it suffices to show that it is possible to come up with a definition of $\mathcal{T}$ which satisfies all the specified Hoare triples.

We define an agent specification as a list:

type-synonym *ht-specification = ht-spec-elem list*

The elements of such a list are tuples containing an action identifier, a mental state formula which is the decision rule for the action and lastly a list of Hoare triples which specify the frame and effect axioms for the action:

type-synonym *ht-spec-elem* $= Bcap \times \Phi_M \times$ *hoare-triple list*

We have not touched upon this earlier, but frame axioms specify what does not change when executing an action and effect axioms specify what does change, i.e. what are the effects of executing the action. Another note is that we only allow a single decision rule in our setup. We could allow for multiple rules per action. Instead, we assume that if there are multiple rules they have been condensed to a single rule using disjunction.

5.1 Satisfiability Problem

Our task is now to show that a low-level semantics can be derived from a specification, i.e. that there exists some $\mathcal{T}$ which complies with the specification. This may also be seen as a model existence problem which has already been studied for other areas of logic. Another perspective on this problem is to consider it as a satisfiability problem, or equivalently that there are no contradictions.

The definition of satisfiability is split into two parts, quantifying over any proper mental state and every element of the specification:

definition *satisfiable* :: *ht-specification* $\Rightarrow$ *bool* **where**
satisfiable $S \equiv \forall M.\ \nabla M \longrightarrow$
$(\forall s \in set\ S.\ sat\text{-}l\ M\ s \wedge sat\text{-}r\ M\ s)$

The first part is concerned with those mental states where the action in question is enabled:

fun *sat-l* :: *mental-state* $\Rightarrow$ *ht-spec-elem* $\Rightarrow$ *bool* **where**
$sat\text{-}l\ M\ (a, \Phi, hts) = (M \models_M \Phi \longrightarrow (\exists \Sigma.\ sat\text{-}b\ M\ hts\ \Sigma))$

When the action is enabled, for all states there should exist a belief base which satisfies all Hoare triples for this action:

definition *sat-b* :: *mental-state* $\Rightarrow$ *hoare-triple list* $\Rightarrow \Phi_L$ *list* $\Rightarrow$ *bool* **where**
$sat\text{-}b\ M\ hts\ \Sigma \equiv$
$(\neg fst\ M \models_P \bot \longrightarrow \neg \Sigma \models_P \bot) \wedge$
$(\forall ht \in set\ hts.\ M \models_M pre\ ht \longrightarrow$
$(\Sigma, [\psi \leftarrow snd\ M.\ \neg \Sigma \models_P \psi]\ ) \models_M post\ ht)$

For a belief base to be satisfiable in this context, we require:

1. that the consistency is preserved, and
2. that the postcondition holds in the new mental state if the precondition holds in the current mental state (the new goal base is derived by removing achieved goals from the current goal base).

The second part of the satisfiability definition is concerned with those mental states where the action in question is not enabled:

fun *sat-r* :: *mental-state* $\Rightarrow$ *ht-spec-elem* $\Rightarrow$ *bool* **where**
$sat\text{-}r\ M\ (\text{-}, \Phi, hts) =$
$(M \models_M \neg \Phi \longrightarrow$
$(\forall ht \in set\ hts.\ M \models_M pre\ ht \longrightarrow M \models_M post\ ht))$

In such a case the postcondition should hold in the same state if the precondition holds.

Because of the way we have set up the specification type, a definition *is-ht-specification* ensures that the specification is satisfiable and that each element reflects the specification of a distinct action.

We briefly mentioned that satisfiability entails the existence of a $\mathcal{T}$ which matches the agent specification. To show this, we first define what it means for a $\mathcal{T}$ to comply with an agent specification:

definition *complies* :: *ht-specification* $\Rightarrow$ *bel-upd-t* $\Rightarrow$ *bool* **where**
$complies\ S\ \mathcal{T} \equiv (\forall s \in set\ S.\ complies\text{-}hts\ s\ \mathcal{T}) \wedge$
$(\forall n.\ n \notin set\ (map\ fst\ S) \longrightarrow (\forall M.\ \mathcal{T}\ n\ M = None))$

Again we can define this for each element of specification individually. Furthermore, we require that $\mathcal{T}$ is only defined for action identifiers present in the specification.

definition *complies-hts* :: $(Bcap \times \Phi_M \times$ *hoare-triple list*$)$ $\Rightarrow$ *bel-upd-t* $\Rightarrow$ *bool* **where**
$complies\text{-}hts\ s\ \mathcal{T} \equiv \forall ht \in set\ (snd\ (snd\ s)).$
$is\text{-}htb\text{-}basic\ ht \wedge (\forall M.\ \nabla M \longrightarrow$
$complies\text{-}ht\ M\ \mathcal{T}\ (fst\ (snd\ s))\ (the\ (htb\text{-}unpack\ ht)))$

The following definition has a convoluted syntax. Essentially, we quantify over all proper mental states and assert compliance for each Hoare triple:

fun *complies-ht* :: *mental-state* $\Rightarrow$ *bel-upd-t* $\Rightarrow \Phi_M \Rightarrow$ $(\Phi_M \times Bcap \times \Phi_M) \Rightarrow$ *bool* **where**
$complies\text{-}ht\ M\ \mathcal{T}\ \Phi\ (\varphi, n, \psi) =$
$((M \models_M \Phi \longleftrightarrow \mathcal{T}\ n\ M \neq None) \wedge$
$(\neg\ (fst\ M) \models_P \bot \longrightarrow \mathcal{T}\ n\ M \neq None \longrightarrow$
$\neg the\ (\mathcal{T}\ n\ M) \models_P \bot) \wedge$
$(M \models_M \varphi \wedge M \models_M \Phi \longrightarrow$
$the\ (\mathcal{M}{*}\ \mathcal{T}\ (basic\ n)\ M) \models_M \psi) \wedge$
$(M \models_M \varphi \wedge M \models_M \neg \Phi \longrightarrow M \models_M \psi))$

Note that in the preceding definition, $\mathcal{M}{*}\ \mathcal{T}$ corresponds to the mental state transformer $\mathcal{M}$ where $\mathcal{T}$ is not fixed by the agent. Furthermore, the definition packs a number of important properties:

1. that the formula φ is the sole factor for enabledness of the action,
2. that consistency is preserved for belief bases, and lastly
3. that it matches the semantics of the Hoare triple.

Arguably, the definitions of satisfiability and compliance could be optimized for readability and less redundancy.

We now show that compliance is entailed by satisfiability. An interesting partial result is that because each element of specification is for a distinct action, the existence of a (partial) $\mathcal{T}$ can be shown for each action and used to show the existence of a $\mathcal{T}$ for the full specification:

Lemma 5.1 (Disjoint compliance)**.** *The existence of a $\mathcal{T}$ for each element of the specification can be used to show the existence of a $\mathcal{T}$ for the full specification.*

$\textit{is-ht-specification } S \Longrightarrow$
$\quad \forall s{\in}\textit{set } S.\ \exists \mathcal{T}.\ \textit{complies-hts } s\ \mathcal{T} \Longrightarrow$
$\quad \exists \mathcal{T}.\ \textit{complies } S\ \mathcal{T}$

Proof. Since there is no overlap between each partial $\mathcal{T}_x$ obtained from the assumptions, we combine the partial functions into a single function $\mathcal{T}$ which complies with the specification by construction. ☐

We then prove that for any specification which follows the definition (most importantly, it is thus satisfiable) there exists a compliant $\mathcal{T}$.

Lemma 5.2 (Compliance)**.** *There exists a belief update function $\mathcal{T}$ for any proper agent specification which is satisfiable.*

$\textit{is-ht-specification } S \Longrightarrow \exists \mathcal{T}.\ \textit{complies } S\ \mathcal{T}$

Proof. By construction of a $\mathcal{T}$ for each specification element using the definition of satisfiable bases and ultimately using Lemma 5.1. ☐

5.2 Derived Proof System

Our primary focus is on proving the truth of particular Hoare triples. Exactly how this plays into proving properties of agents becomes apparent later. To this end, we need a Hoare logic proof system. While we can state the general proof rules for Hoare triples, as well as axioms regarding pre- and postconditions involving the special belief and goal modalities, the most important properties of any agent program depend on the agent specification. The user needs to specify a number of effect and frame axioms. They state what is changed and what is not changed by executing the action. Outside of the general rules for proving Hoare triples, we include a special import rule which allows for any of the specified Hoare triples as axioms. The soundness of the import rule follows directly from the compliance of $\mathcal{T}$.

The proof system is defined inductively:

inductive $\textit{derive}_H :: \textit{hoare-triple} \Rightarrow \textit{bool}\ (\vdash_H)$ **where**

The *import* rule allows for a specified Hoare triple as an axiom:

$(n, \Phi, \textit{hts}) \in \textit{set } S \Longrightarrow \{\ \varphi\ \}\ (\textit{basic } n)\ \{\ \psi\ \} \in \textit{set hts} \Longrightarrow$
$\vdash_H \{\ \varphi\ \}\ (\textit{basic } n)\ \{\ \psi\ \}$

The *persist* rule states that a goal persists as either a belief or goal unless it is dropped:

$\neg\ \textit{is-drop } a \Longrightarrow \vdash_H \{\ G\ \Phi\ \}\ a\ \{\ B\ \Phi \vee G\ \Phi\ \}$

The *inf* rule states that nothing happens if the precondition implies that the action is not enabled:

$\models_E (\varphi \longrightarrow \neg(\textit{enabledb } a)) \Longrightarrow \vdash_H \{\ \varphi\ \}\ a\ \{\ \varphi\ \}$

The following rules state important properties of the *adopt* and *drop* actions:

$\vdash_H \{\ B\ \Phi\ \}\ (\textit{adopt } \psi)\ \{\ B\ \Phi\ \}$

$\vdash_H \{\ \neg\ (B\ \Phi)\ \}\ (\textit{adopt } \psi)\ \{\ \neg\ (B\ \Phi)\ \}$

$\vdash_H \{\ B\ \Phi\ \}\ (\textit{drop } \psi)\ \{\ B\ \Phi\ \}$

$\vdash_H \{\ \neg\ (B\ \Phi)\ \}\ (\textit{drop } \psi)\ \{\ \neg\ (B\ \Phi)\ \}$

$\neg \models_P (\neg\ \Phi) \Longrightarrow \vdash_H \{\ \neg\ (B\ \Phi)\ \}\ (\textit{adopt } \Phi)\ \{\ G\ \Phi\ \}$

$\vdash_H \{\ G\ \Phi\ \}\ (\textit{adopt } \psi)\ \{\ G\ \Phi\ \}$

$\neg \models_P (\psi \longrightarrow \Phi) \Longrightarrow$
$\quad \vdash_H \{\ \neg\ (G\ \Phi)\ \}\ (\textit{adopt } \psi)\ \{\ \neg\ (G\ \Phi)\ \}$

$\models_P (\Phi \longrightarrow \psi) \Longrightarrow \vdash_H \{\ G\ \Phi\ \}\ (\textit{drop } \psi)\ \{\ \neg\ (G\ \Phi)\ \}$

$\vdash_H \{\ \neg(G\ \Phi)\ \}\ (\textit{drop } \psi)\ \{\ \neg\ (G\ \Phi)\ \}$

$\vdash_H \{\ \neg(G\ (\Phi \wedge \psi)) \wedge (G\ \Phi)\ \}\ (\textit{drop } \psi)\ \{\ G\ \Phi\ \}$

Finally, we have the structural rules:

$\vdash_H \{\ \varphi \wedge \psi\ \}\ a\ \{\ \varphi'\ \} \Longrightarrow \models_M (\varphi \wedge \neg\psi \longrightarrow \varphi') \Longrightarrow$
$\quad \vdash_H \{\ \varphi\ \}\ (\psi \triangleright \textit{do } a)\ \{\ \varphi'\ \}$

$\models_M (\varphi' \longrightarrow \varphi) \Longrightarrow \vdash_H \{\ \varphi\ \}\ a\ \{\ \psi\ \} \Longrightarrow$
$\quad \models_M (\psi \longrightarrow \psi') \Longrightarrow \vdash_H \{\ \varphi'\ \}\ a\ \{\ \psi'\ \}$

$\vdash_H \{\ \varphi_1\ \}\ a\ \{\ \psi_1\ \} \Longrightarrow \vdash_H \{\ \varphi_2\ \}\ a\ \{\ \psi_2\ \} \Longrightarrow$
$\quad \vdash_H \{\ \varphi_1 \wedge \varphi_2\ \}\ a\ \{\ \psi_1 \wedge \psi_2\ \}$

$\vdash_H \{\ \varphi_1\ \}\ a\ \{\ \psi\ \} \Longrightarrow \vdash_H \{\ \varphi_2\ \}\ a\ \{\ \psi\ \} \Longrightarrow$
$\quad {}_H \{\ \varphi_1 \vee \varphi_2\ \}\ a\ \{\ \psi\ \}$

Due to the sheer number of proof rules, proving the soundness of the system becomes quite involved.

Theorem 5.3 (Soundness of $\vdash_H$)**.** *The Hoare system $\vdash_H$ is sound with respect to the semantics of Hoare triples $\models_H$.*

$\vdash_H H \Longrightarrow \models_H H$

Proof. By induction on the proof rules. The import rule follows directly from Lemma 5.2. The rule for conditional actions follows from Lemma 4.4. The remaining rules follow from the semantics of mental states $\models_E$ and the semantics of Hoare triples $\models_H$. ☐

6 PROVING CORRECTNESS

In this section, we describe how to prove temporal properties of agents by means of a temporal logic which is constructed on top of the logic for GOAL. Ultimately, we show that proofs of certain liveness and safety properties can be reduced to proofs of Hoare triples in the Hoare logic.

6.1 Temporal Logic

We start by setting up a datatype for temporal logic formulas with just two temporal operators and where the type variable $'a$ allows for any type of atoms:

datatype $'a\ \Phi_T =$
$F\ (\bot_T)\ |$
$Atom\ 'a\ |$
$Negation\ 'a\ \Phi_T\ (\neg_T)\ |$
$Implication\ 'a\ \Phi_T\ 'a\ \Phi_T$ (**infixr** $\longrightarrow_T$ *60*) |
$Disjunction\ 'a\ \Phi_T\ 'a\ \Phi_T$ (**infixl** $\vee_T$ *70*) |
$Conjunction\ 'a\ \Phi_T\ 'a\ \Phi_T$ (**infixl** $\wedge_T$ *80*) |
init |
$until\ 'a\ \Phi_T\ 'a\ \Phi_T$

The Boolean operators each have a subscript to avoid ambiguity, e.g. $\longrightarrow_T$ for implication.

Additional temporal operators can be defined by combining the existing ones. The *always* operator $\Box$ states that the operand remains true forever:

definition *always* :: $'a\ \Phi_T \Rightarrow 'a\ \Phi_T\ (\Box)$ **where**
$\Box\varphi \equiv \varphi\ until\ \bot_T$

The *eventuality* operator $\diamond$ states that the operand is true at some point:

definition *eventuality* :: $'a\ \Phi_T \Rightarrow 'a\ \Phi_T\ (\diamond)$ **where**
$\diamond\varphi \equiv \neg_T\ (\Box\ (\neg_T\varphi))$

The *unless* operator states for φ *unless* ψ that if φ becomes true then it remains true until ψ becomes true:

definition *unless* :: $'a\ \Phi_T \Rightarrow 'a\ \Phi_T \Rightarrow 'a\ \Phi_T$ **where**
$\varphi\ unless\ \psi \equiv \varphi \longrightarrow_T (\varphi\ until\ \psi)$

In the following, the type Φ_{TM} is for temporal logic with the belief and goal modalities as atoms. The semantics of temporal logic for agents is evaluated in terms of a trace s and a natural number i, indicating that the formula is to be evaluated in the i-th state of trace s:

$s, i \models_T \bot_T = False$
$s, i \models_T (Atom\ x) = ((Atom\ x)[s\ i]_M)$
$s, i \models_T (\neg_T\ p) = (\neg\ (s, i \models_T p))$
$s, i \models_T (p \longrightarrow_T q) = ((s, i \models_T p) \longrightarrow (s, i \models_T q))$
$s, i \models_T (p \vee_T q) = ((s, i \models_T p) \vee (s, i \models_T q))$
$s, i \models_T (p \wedge_T q) = ((s, i \models_T p) \wedge (s, i \models_T q))$
$s, i \models_T init = (i = 0)$
$s, i \models_T (\varphi\ until\ \psi) = ((\exists\ j \geq i.\ s, j \models_T \psi\ \wedge$
$(\forall k \geq i.\ j > k \longrightarrow s, k \models_T \varphi)) \vee (\forall k \geq i.\ s, k \models_T \varphi))$

The Boolean operators are defined using Isabelle's operators. Identical results can be achieved using the more traditional if-then-else constructs if one desires a more programming-like style. The case for atoms simply delegates the task to the semantics functions for mental state formulas. The case for *init* is true when *i = 0*, i.e. when we are at the very first state of the trace. The more complicated case for *until* warrants further explanation: either φ remains true until a future point j where ψ becomes true, or φ remains true forever.

6.2 Liveness and Safety Properties

There is a close relationship between proving Hoare triples and proving liveness and safety properties of an agent. Concerning safety, we can show that φ is a stable property by proving φ *unless F*. In case we also have *init* $\longrightarrow\varphi$, we say that φ is an invariant of the agent program. We now prove that, due to the *unless* operator, this safety property can be reduced to proving a Hoare triple for each conditional action.

Theorem 6.1. *After executing any action from* Π *either* φ *persists or* ψ *becomes true and we can conclude* φ unless ψ *and conversely.*

$\forall (\upsilon \triangleright do\ b) \in \Pi.\ \models_H \{\ \varphi \wedge \neg\ \psi\ \}\ (\upsilon \triangleright do\ b)\ \{\ \varphi \vee \psi\ \})$
$\longleftrightarrow\ Agent \models_T \varphi\ unless\ \psi$

Proof. The $\longrightarrow$ direction is shown by contraposition where the semantics of temporal logic leads us to a contradiction. The $\longleftarrow$ direction is shown by a case distinction of $\models_M \varphi$ in some arbitrarily chosen state. □

Liveness properties involve showing that a particular state is always reached from a given situation. A certain subclass of these properties is captured by the *ensures* operator:

definition *ensures* :: $'a\ \Phi_T \Rightarrow 'a\ \Phi_T \Rightarrow 'a\ \Phi_T$ **where**
$\varphi\ ensures\ \psi \equiv (\varphi\ unless\ \psi) \wedge_T (\varphi \longrightarrow_T \diamond\psi)$

Here, φ *ensures* ψ informally means that φ guarantees the realization of ψ.

Also for the *ensures* operator we can show that it can be reduced to proofs of Hoare triples.

Theorem 6.2. *The proof of an* ensures *property can be reduced to the proof of a set of Hoare triples.*

$\forall b \in \Pi.\ \models_H \{\ \varphi \wedge \neg\ \psi\ \}\ b\ \{\ \varphi \vee \psi\ \} \Longrightarrow$
$\exists b \in \Pi.\ \models_H \{\ \varphi \wedge \neg\ \psi\ \}\ b\ \{\ \psi\ \} \Longrightarrow$
$Agent \models \varphi\ ensures\ \psi$

Proof. From the fact that any trace in *Agent* is a fair trace, we obtain a contradiction using the semantics of temporal logic $\models_T$ and of Hoare triples $\models_H$. □

Finally, we introduce the temporal operator $\mapsto$ ("leads to"). The property $\varphi \mapsto \psi$ is similar to *ensures* except that it does not require φ to remain true until ψ is realized. It is defined from the ensures operator inductively:

inductive *leads-to* :: $\Phi_{TM} \Rightarrow \Phi_{TM} \Rightarrow bool$ (**infix** $\mapsto$ *55*) **where**
base: $\forall s \in Agent.\ \forall i.\ s, i \models_T (\varphi\ ensures\ \psi) \Longrightarrow \varphi \mapsto \psi\ |$
imp: $\varphi \mapsto \chi \Longrightarrow \chi \mapsto \psi \Longrightarrow \varphi \mapsto \psi\ |$
disj: $\forall \varphi \in set\ \varphi_L.\ \varphi \mapsto \psi \Longrightarrow disL\ \varphi_L \mapsto \psi$

The rule *imp* states a transitive property and *disj* states a disjunctive property. The function *disL* forms a disjunction from a list of formulas:

fun $disL :: {}'a\ \Phi_T\ list \Rightarrow {}'a\ \Phi_T$ **where**
$disL\ [] = \bot_T \mid$
$disL\ [\varphi] = \varphi \mid$
$disL\ (\varphi \# \varphi_L) = \varphi \vee_T disL\ \varphi_L$

We can prove that the temporal operator $\mapsto$ can be used to state a correctness property for agents.

Lemma 6.3. *The proof of a certain class of temporal properties can be reduced to a proof of the "leads to" operator* $\mapsto$.

$\varphi \mapsto \psi \Longrightarrow \forall s \in Agent.\ \forall i.\ s, i \models_T (\varphi \longrightarrow_T \diamond\psi)$

Proof. By induction on the rules and using the semantics of temporal logic $\models_T$. □

Lemma 6.3 is used to prove that temporal logic statements of the form $P \mapsto Q$ in fact state temporal properties of agents.

7 AN EXAMPLE AGENT

In this section, we showcase how to use the verification framework in Isabelle/HOL to prove the correctness of a simple agent which can solve a simple task to collect a block. Informally, the agent initially is located in a special dropzone location where a block is to be delivered. The agent must go to a room containing such a block. Before the agent can pick up a block, it must move right next to it. Finally, the agent must return to the dropzone and deliver the block.

The belief base of the initial mental state is:

$[\ in\text{-}dropzone, \neg\ in\text{-}room, \neg\ holding, \neg\ at\text{-}block\ ]$

The goal base of the initial mental state is:

$[\ collect\ ]$

The initial belief and goal bases capture the initial configuration:

- The agent is located in the dropzone.
- Naturally, the agent is therefore not in a room containing a block, and consequently also not next to a block.
- The agent is not holding a block.
- The goal of the agent is to collect a block.

Furthermore, our example agent has a number of available actions:

- *go-dropzone*: The agent moves to the dropzone.
- *go-room*: The agent moves to a room containing a block.
- *go-block*: When in a room containing a block, the agent moves right next to the block.
- *pick-up*: If the agent is right next to a block, the agent will pick it up.
- *put-down*: If the agent is carrying a block, the agent will put it down.

Each action has a condition which states when the action can be performed. This is specified by the following formulas for enabledness:

$enabled(go\text{-}dropzone) \equiv B\ in\text{-}room \wedge B\ holding$

$enabled(go\text{-}room) \equiv B\ in\text{-}dropzone \wedge \neg\ (B\ holding)$

$enabled(go\text{-}block) \equiv B\ in\text{-}room \wedge \neg\ (B\ at\text{-}block) \wedge \neg\ (B\ holding)$

$enabled(pick\text{-}up) \equiv B\ at\text{-}block \wedge \neg\ (B\ holding)$

$enabled(put\text{-}down) \equiv B\ holding \wedge B\ in\text{-}dropzone$

Because of the simplicity of our example, only one of these conditions is true in any given state. In other words, there is only ever one meaningful action.

Furthermore, we must specify effect axioms for the actions. In our case, each action has exactly one effect axiom:

$\{\ B\ in\text{-}room \wedge B\ holding\ \}\ go\text{-}dropzone\ \{\ B\ in\text{-}dropzone\ \}$

$\{\ B\ in\text{-}dropzone \wedge \neg\ (B\ holding)\ \}\ go\text{-}room\ \{\ B\ in\text{-}room\ \}$

$\{\ B\ in\text{-}room \wedge \neg\ (B\ at\text{-}block) \wedge \neg\ (B\ holding)\ \}\ go\text{-}block\ \{\ B\ at\text{-}block\ \}$

$\{\ B\ at\text{-}block \wedge \neg\ (B\ holding)\ \}\ pick\text{-}up\ \{\ B\ holding\ \}$

$\{\ B\ in\text{-}dropzone \wedge B\ holding\ \}\ put\text{-}down\ \{\ B\ collect\ \}$

Since we do not have the space to go through the details, the specification of frame axioms and proofs of invariants has been left out of the paper.

We define the belief update function as some $\mathcal{T}$ which complies with the specification. We are allowed to do this if we can prove that such a specification exists.

definition $\mathcal{T}_x \equiv SOME\ \mathcal{T}.\ complies\ S_x\ \mathcal{T}$

We need to prove that our different example components actually compose a single agent program:

interpretation *bw4t*:
$single\text{-}agent\text{-}program\ \mathcal{T}_x\ (set\ \Pi_x)\ M_{0x}\ S_x$

The requirements of a single agent program has not been described earlier. It requires showing the existence of some $\mathcal{T}$ which complies with the specification. This is due to the definition of $\mathcal{T}_x$ using *SOME* which is based on Hilbert's epsilon operator i.e. the axiom of choice. The main issue is thus to show the satisfiability of the specification due to Lemma 5.2. The proof is rather lengthy and is not shown here.

The main point of interest is the proof of the statement involving the "leads to" operator:

lemma $B\ in\text{-}dropzone \wedge \neg\ (B\ at\text{-}block) \wedge \neg\ (B\ holding) \wedge G\ collect \mapsto B\ collect$

This captures that the desired state, where the agent believes to have collected the block, is reached from the initial configuration of the agent. The inductive definition of the operator makes it possible to split the proof into subproofs for each step:

have *in-dropzone* $\wedge \neg$ (*B at-block*) $\wedge \neg$ (*B holding*) $\wedge$ *G collect* $\mapsto$ *B in-room* $\wedge \neg$ (*B at-block*) $\wedge \neg$ (*B holding*) $\wedge$ *G collect*
...
moreover have *B in-room* $\wedge \neg$ (*B at-block*) $\wedge \neg$ (*B holding*) $\wedge$ *G collect* $\mapsto$ *B in-room* $\wedge$ *B at-block* $\wedge \neg$ (*B holding*) $\wedge$ *G collect*
...
moreover have *B in-room* $\wedge$ *B at-block* $\wedge \neg$ (*B holding*) $\wedge$ *G collect* $\mapsto$ *B in-room* $\wedge$ *B holding* $\wedge$ *G collect*
...
moreover have *B in-room* $\wedge$ *B holding* $\wedge$ *G collect* $\mapsto$ *B in-dropzone* $\wedge$ *B holding* $\wedge$ *G collect*
...
moreover have *B in-dropzone* $\wedge$ *B holding* $\wedge$ *G collect* $\mapsto$ *B collect*
...
ultimately show *?thesis* **using** *imp* **by** *blast*

Every step in the proof above is achieved by a single action; again this is due to the simplicity of the example. We can easily conceive of programs which have multiple paths to the desired state in which case all paths need to be considered.

Due to Lemma 6.3, the proof of *B in-dropzone* $\wedge \neg$ (*B at-block*) $\wedge \neg$ (*B holding*) $\wedge$ *G collect* $\mapsto$ *B collect* shows a correctness property of our simple example agent.

8 CONCLUSIONS

We have presented a formalization of a verification framework for GOAL agents in its entirety. Furthermore, we have demonstrated how it can be applied to an agent specified in the agent logic by means of a simple example.

There are still a number of issues for our attention in the future. As was clear in our earlier work, the original formulation of the verification framework has some limitations. First, the framework is limited to single agent programs. Long term, the aim is to verify programs with multiple communicating agents. To this end, we want to explore how existing work on extending the framework can be integrated into our Isabelle/HOL formalization (Bulling and Hindriks, 2009). Second, the agent logic is limited to propositional logic which not only causes an inconvenience in specifying complex agent programs, but also means that certain things cannot be modelled.

Outside of extending the framework, there are also a number of usability concerns to address. This has little theoretical significance, but it would be interesting to experiment with providing more means of automation for conducting proofs of agent properties. While we did not have the space to show the proofs of our example agent in full details, we note that it can become quite tedious to conduct these proofs manually. Due to the complexity of the structures involved in the proofs, the automation of Isabelle/HOL is not geared towards such proofs out-of-the-box. Further work is required to enable more automation.

Nevertheless, the present paper demonstrates that a theorem proving approach to verifying agents is feasible. Furthermore, because we have formalized the framework in a proof assistant, we are able to provide a high level of assurance as everything is checked by Isabelle/HOL. In conclusion, we are excited to see the future potential of the framework enabled by the capabilities of Isabelle/HOL.

ACKNOWLEDGEMENTS

Koen V. Hindriks has provided valuable insights into the details of GOAL. Jørgen Villadsen, Frederik Krogsdal Jacobsen and Asta Halkjær From have commented on drafts.

REFERENCES

Alechina, N., Dastani, M., Khan, A. 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*, pages 1–33. Springer.

Bordini, R., Fisher, M., Wooldridge, M., and Visser, W. (2004). Model Checking Rational Agents. *Intelligent Systems, IEEE*, 19:46–52.

Bulling, N. and Hindriks, K. V. (2009). Towards a Verification Framework for Communicating Rational Agents. In *Multiagent System Technologies, 7th German Conference, MATES 2009, Hamburg, Germany, September 9-11, 2009. Proceedings*, Lecture Notes in Computer Science, pages 177–182. Springer.

Calegari, R., Ciatto, G., Mascardi, V., and Omicini, A. (2021). Logic-Based Technologies for Multi-Agent Systems: Summary of a Systematic Literature Review. In *Proceedings of the 20th International Conference on Autonomous Agents and Multiagent Systems*, AAMAS '21, pages 1721—1723. International Foundation for Autonomous Agents and Multiagent Systems.

de Boer, F. S., Hindriks, K. V., van der Hoek, W., and Meyer, J.-J. (2007). A verification framework for agent programming with declarative goals. *Journal of Applied Logic*, 5:277–302.

Dennis, L., Fisher, M., Webster, M. P., and Bordini, R. H. (2012). Model checking agent programming languages. *Automated Software Engineering*, 19:5–63.

Dennis, L. A. and Fisher, M. (2009). Programming Verifiable Heterogeneous Agent Systems. In *Programming Multi-Agent Systems*, pages 40–55. Springer.

Dix, J., Logan, B., and Winikoff, M. (2019). Engineering Reliable Multiagent Systems (Dagstuhl Seminar 19112). *Dagstuhl Reports*, 9(3):52–63.

Gomes, V. B. F., Kleppmann, M., Mulligan, D. P., and Beresford, A. R. (2017). Verifying strong eventual consistency in distributed systems. *Proc. ACM Program. Lang.*, 1(OOPSLA).

Hindriks, K. V. (2009). Programming Rational Agents in GOAL. In *Multi-Agent Programming: Languages, Tools and Applications*, pages 119–157. Springer.

Hindriks, K. V., de Boer, F. S., van der Hoek, W., and Meyer, J.-J. (2001). Agent Programming with Declarative Goals. In *Intelligent Agents VII Agent Theories Architectures and Languages*, pages 228–243. Springer.

Hindriks, K. V. and Dix, J. (2014). GOAL: A Multi-agent Programming Language Applied to an Exploration Game. In *Agent-oriented software engineering*, pages 235–258. Springer.

Java Pathfinder (2021). https://github.com/javapathfinder/jpf-core/wiki. Accessed: 2021-11-22.

Jensen, A. (2021a). Towards Verifying a Blocks World for Teams GOAL Agent. In Rocha, A., Steels, L., and van den Herik, J., editors, *Proceedings of the 13th International Conference on Agents and Artificial Intelligence*, volume 1, pages 337–344. Science and Technology Publishing.

Jensen, A. (2021b). Towards Verifying GOAL Agents in Isabelle/HOL. In Rocha, A., Steels, L., and van den Herik, J., editors, *Proceedings of the 13th International Conference on Agents and Artificial Intelligence*, volume 1, pages 345–352. Science and Technology Publishing.

Jensen, A., Hindriks, K., and Villadsen, J. (2021). On Using Theorem Proving for Cognitive Agent-Oriented Programming. In Rocha, A., Steels, L., and van den Herik, J., editors, *Proceedings of the 13th International Conference on Agents and Artificial Intelligence*, volume 1, pages 446–453. Science and Technology Publishing.

Jensen, A. B. (2021c). A theorem proving approach to formal verification of a cognitive agent. In Matsui, K., Omatu, S., Yigitcanlar, T., and Rodriguez-González, S., editors, *Distributed Computing and Artificial Intelligence, Volume 1: 18th International Conference, DCAI 2021, Salamanca, Spain, 6-8 October 2021*, volume 327 of *Lecture Notes in Networks and Systems*, pages 1–11. Springer.

Jongmans, S.-S., Hindriks, K., and Riemsdijk, M. (2010). Model Checking Agent Programs by Using the Program Interpreter. In *Computational Logic in Multi-Agent Systems*, pages 219–237.

Koeman, V., Hindriks, K., and Jonker, C. (2018). Automating failure detection in cognitive agent programs. *International Journal of Agent-Oriented Software Engineering*, 6:275–308.

Nipkow, T., Paulson, L., and Wenzel, M. (2002). *Isabelle/HOL — A Proof Assistant for Higher-Order Logic*. Springer.

Ringer, T., Palmskog, K., Sergey, I., Gligoric, M., and Tatlock, Z. (2019). QED at Large: A Survey of Engineering of Formally Verified Software. *Foundations and Trends in Programming Languages*, 5(2-3):102–281.

Shapiro, S., Lespérance, Y., and Levesque, H. J. (2002). The Cognitive Agents Specification Language and Verification Environment for Multiagent Systems. In *Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems: Part 1*, AAMAS '02, pages 19–26. Association for Computing Machinery, New York, NY, USA.

Winikoff, M. (2010). Assurance of Agent Systems: What Role Should Formal Verification Play? In Dastani, M., Hindriks, K. V., and Meyer, J.-J. C., editors, *Specification and Verification of Multi-agent Systems*, pages 353–383. Springer US, Boston, MA.

Winikoff, M. and Cranefield, S. (2014). On the Testability of BDI Agent Systems. *Journal of Artificial Intelligence Research*, 51:71–131.