Chapter 8
A Verification Logic for GOAL Agents

K.V. Hindriks

Abstract Although there has been a growing body of literature on verification of agents programs, it has been difficult to design a verification logic for agent programs that fully characterizes such programs and to connect agent programs to agent theory. The challenge is to define an agent programming language that defines a computational framework but also allows for a logical characterization useful for verification. The agent programming language GOAL has been originally designed to connect agent programming to agent theory and we present additional results here that GOAL agents can be fully represented by a logical theory. GOAL agents can thus be said to execute the corresponding logical theory.

K.V. Hindriks
Delft University of Technology, The Netherlands e-mail: k.v.hindriks@tudelft.nl

M. Dastani et al. (eds.), *Specification and Verification of Multi-agent Systems*,
DOI 10.1007/978-1-4419-6984-2_8,

8.1 Introduction

As technology for developing agent systems is becoming more mature, the availability of techniques for verifying such systems also becomes more important. Such techniques do not only complement tools for debugging agent systems but may also be used to supplement the techniques available for debugging agents. For example, model checking techniques may be used to find *counter examples* that show that an agent system does not satisfy a particular property. A counter example produces a run of a system that violates a property and as such indicates what is wrong with an agent. Program model checking discussed in [66] is an approach that supports this type of verification. It involves the construction of a semantic model M that correctly represents an agent's execution and that can be used to check satisfaction of a property φ, i.e. $M \models \varphi$. The key problem that needs to be solved to be able to use model checking for verification concerns the *efficient* construction of (part of) a model M from a given agent system that is sufficient for verifying this system.

Model checking is one approach to verifying agents. An alternative approach to verifying agents involves the use of *deduction*. This approach assumes that a *logical theory* of an agent is available. The task of verifying that an agent satisfies a particular property φ amounts to deducing φ from the given theory T, i.e. $T \vdash \varphi$. The key problem that needs to be solved to be able to use deduction as a verification tool concerns the construction of a corresponding *logical theory* T from a given agent system. It is the goal of this chapter to introduce such a theory for the Goal agent programming language [221].

Verification techniques based on deduction have been widespread in Computer Science and have been provided for a broad range of programming languages. The programming constructs and the structure of programs in a programming language often naturally give rise to an associated *programming logic*. This has been particularly true for imperative programming languages but also for concurrent programming languages [23, 296, 298].

The verification approach presented here for Goal consists of two parts. First, an operational semantics that provides a model for executing agent programs is defined. This provides a computational framework that specifies how Goal agents are to be executed. Second, a logic for verification is introduced and it is shown that the logical semantics corresponds with the operational semantics. It is our aim in this chapter to stay as close as possible to the actual implementation of the Goal language, although we do abstract away a number of features that are present in the interpreter for the language and focus on single agents. In particular, we have provided a semantics for logic programs as part of the operational semantics to model the Prolog engine that is used in the implementation. Similarly, we have aimed for a verification logic which semantics corresponds in a precise sense with the operational semantics and can be used to fully characterize agent programs. As we will show, basic Goal agent programs discussed here may be mapped into a corresponding logical theory by means of a straightforward translation scheme that

fully characterizes the runs of these programs. This result shows that GOAL agent programs may be perceived as executing the corresponding logical theory.

8.2 Related work

There is a growing body of literature on deductive verification of rational agents and agent programs that are based on the Belief-Desire-Intention metaphor. The work related most to our approach concerns logics for the languages 3APL [223] and its successor 2APL [122], and work on ConGolog [193] - a closely related language to 3APL (cf. [225]). The CASL framework [395], also discussed in this volume, that can be viewed as an extension of the situation calculus, also aims at defining a logical framework for specifying rational agents.

Early work on designing a verification logic for 3APL is reported in [216] and introduces a dynamic logic to reason about 3APL programs without self-modifying reasoning rules. The logic assumes that such programs terminate, which derives from the use of a dynamic logic [210], but also accounts for the use of free variables in the execution of 3APL programs. In [6], a logic for reasoning about agent programs in a simplified version of 3APL is introduced, called SimpleAPL. [6] presents a propositional logic to reason about the core features of 3APL, including beliefs and goals, which is proven sound and complete. Finally, [7] discusses a logic for reasoning about the deliberation cycle of an agent in 3APL. This paper addresses reasoning at another level, the execution strategy of the interpreter, rather than the execution of actions and action selection by the agent itself.

[288] presents a Hoare-style proof system for verifying Golog programs, a subset of the ConGolog language, which is proven sound and complete. The work is in many ways similar to that discussed in the previous paragraph. The logic and aims are similar in various respects, but agents in Golog do not have explicit beliefs and goals. The latter restriction has motivated the extension of the basic Golog framework with explicit knowledge and goal operators in CASL [395]. CASL extends the situation calculus with a semantics for such operators using situations in a way similar to how modal worlds are used in classical modal logic. The approach, however, is very expressive allowing for quantification over formulas (as terms) and it is less clear what the computational properties of the framework are.

In previous work on GOAL [60], a verification framework for GOAL agents has been introduced that consists of two parts: A Hoare-style logic for reasoning about actions and a temporal logic for reasoning about runs of such agents. This framework allows for the verification of GOAL agents and has been related to Intention Logic [217]. The work presented here differs in various ways from [60]. First, here we use a temporal logic for reasoning about actions and do not introduce Hoare-style axioms. Second, we show that the resulting logic can be used to fully characterize GOAL agents. Third, the logic allows for quantification and is a first-order verification logic.

8.3 The Agent Programming Language Goal

The agent programming language Goal is presented here by defining its *operational* semantics. The section is organized as follows. In 8.3.1 we very briefly informally introduce the key concepts that make up a Goal agent program. Goal agents derive their choice of action from their beliefs and goals and need a knowledge representation language to represent these which is introduced in 8.3.2. As Goal agents derive what to do next from their mental state, we then continue by introducing the semantics of mental states in 8.3.3. Using the mental state semantics, the meaning of a Goal agent program is specified using *structural operational semantics* [340]. The operational semantics determines which *transitions* from one mental state to another can be made. It makes precise how the mental state of a Goal agent changes when it performs an action.

8.3.1 Goal *Agent Programs*

A Goal agent program consists of the agent's *knowledge*, *beliefs*, its *goals*, a set of *action rules* and a set of *action specifications*. Other features present in the language for e.g. percept handling, modules and communication are not discussed here. For a more thorough and comprehensive introduction to the language see [221].[1]

The knowledge, the beliefs and the goals of an agent are specified *declaratively* by means of a knowledge representation language, which facilitates the design of agent programs at the *knowledge level* [321]. The knowledge of a Goal agent is assumed to be static and does not change over time. The knowledge, beliefs and goals of an agent define the agent's *mental state*. Goal does not commit to any particular knowledge representation technology but for purposes of illustration we will use a variant of PDDL here [192].[2] This has the additional benefit that PDDL action specficiations with conditional actions are supported, and this variant of Goal is able to support the full expressivity of ADL action specifications [336]. A Goal agent derives its choice of action from its beliefs and goals. It does so by means of action rules of the form **if** ψ **then** $a(\mathbf{t})$ where ψ is a condition on the mental state of the agent and $a(\mathbf{t})$ is an action the agent can perform. Whenever the mental state condition ψ holds the corresponding action $a(\mathbf{t})$ is said to be an *option*. At any time, there may be multiple options from which the agent will select one *nondeterministically*.

[1] The reader is referred to [215] for a semantics of modules and [220] for a semantics of communication.

[2] The first-order logic variant of PDDL that we will discuss includes so-called axioms for derived predicates. This variant has been implemented as one of the options for choosing a knowledge representation language in Goal. A programmer can also choose to use Prolog, for example. Although the first-order language presented is richer than that of Prolog, the fragment discussed here can be compiled into Prolog (cf. [289]).

8.3.2 Knowledge Representation Language

We will use a first-order language $\mathcal{L}_0$ for representing the knowledge, beliefs and goals of an agent.[3] $\mathcal{L}_0$ is built using a vocabulary that consists of a *finite* sets of predicates $\mathcal{P}$ with typical elements p, function symbols $\mathcal{F}$ with typical elements f, constant symbols C with typical elements a, b, c, and an infinite supply of variables $\mathcal{V}$ with typical elements x, y, z. We also assume that $\mathcal{L}_0$ includes equality $=$. The set of predicates $\mathcal{P}$ consists of two disjoint sets of so-called *basic* predicates $\mathcal{B}$ and *derived* predicates $\mathcal{D}$, such that $\mathcal{P} = \mathcal{B} \cup \mathcal{D}$ and $\mathcal{B} \cap \mathcal{D} = \emptyset$. The distinction is used to differentiate predicates that may be updated by actions from those that cannot be updated so. The idea is that basic predicates may be updated whereas derived predicates may only be used to define additional concepts that are defined in terms of the more basic predicates.

Definition 8.1. *(Syntax of $\mathcal{L}_0$)*

$$\begin{array}{lll} t \in \mathcal{T} & ::= & x \mid c \mid f(\mathbf{t}) \\ \phi \in \mathcal{L}_0 & ::= & t = t \mid p(\mathbf{t}) \mid \phi \wedge \phi \mid \phi \vee \phi \mid \neg\phi \mid \forall x(\phi) \end{array}$$

A *term* t is either a variable $x \in \mathcal{V}$, a constant $c \in C$, or a function symbol f applied to a vector $\mathbf{t}$ of terms of the appropriate arity. Vectors of variables and terms are denoted by bold face $\mathbf{x}$ respectively $\mathbf{t}$. Formulas $p(\mathbf{t})$ with $p \in \mathcal{P}$ are called *atoms*. Atoms $p(\mathbf{t})$ or their negations $\neg p(\mathbf{t})$ are also called *literals*. Literals l and $\neg l$ are said to be *complementary*. As usual, $\phi \rightarrow \phi'$ is an abbreviation for $\neg\phi \vee \phi'$, and $\exists x(\phi)$ abbreviates $\neg\forall x(\neg\phi)$. We write $\phi[\mathbf{x}]$ to indicate that all free variables of ϕ occur in the vector $\mathbf{x}$. A formula that does not contain free variables is said to be *closed*. Closed formulas without quantifiers, i.e. formulas without any variables, are also said to be *ground*. The set of all *ground* atoms of the form $p(\mathbf{t})$ is denoted by F. The subset $F_b \subseteq F$ consists of all atoms of the form $p(\mathbf{t})$ with $p \in \mathcal{B}$, and, similarly $F_d \subseteq F$ consists of all atoms $p(\mathbf{t})$ with $p \in \mathcal{D}$. Elements from F_b (F_d) are also called *basic (derived) facts*, and basic (derived) facts and their negations are called *basic (derived) literals*. Finally, we use $\forall(\phi)$ to denote the universal closure of ϕ.

The distinction between basic and derived predicates that is made here is used to distinguish basic facts about an environment from conceptual or domain knowledge that can be defined in terms of these basic facts. The use of such defined predicates facilitates programming and reduces the size of the program. We adopt the definition of *derived predicate axioms* and related definitions below from [413].

Definition 8.2. *(Derived Predicate Axiom)*
A *derived predicate axiom* is a formula of the form $\forall\mathbf{x}(\phi[\mathbf{x}] \rightarrow d(\mathbf{x}))$ with $d \in \mathcal{D}$.[4]

[3] This language is referred to as a *knowledge representation language* traditionally, even though it is also used to represent the goals of an agent.

[4] We do not allow terms in the head of a derived predicate axiom here, mainly because it simplifies the presentation and definition of *completion* below.

In our setting, the antecedent ϕ of such an axiom may not contain occurrences of other derived predicates that "depend on" the definition of d.[5] Technically, the requirement is that a set of derived predicate axioms needs to be *stratified*. Before we define stratification it is useful to introduce the notion of a negated normal form. A formula ϕ is in *negated normal form* if all occurrences of negations occur directly in front of atoms. For example, $\forall x(\neg p(x) \vee (q(x) \wedge r(x)))$ is in negated normal form but $\neg\exists x(\neg(p(x) \wedge q(x)))$ is not because negation occurs in front of the existential quantifier and in front of a conjunction. We remark here that this definition assumes that all implications $\rightarrow$ have been expanded into their unabbreviated form. That is, occurrences of, for example, $(p(x) \wedge q(x)) \rightarrow r(x)$ are not allowed and must be replaced with the negated normal form of, in this case, $\neg(p(x) \wedge q(x)) \vee r(x)$, i.e. $\neg p(x) \vee \neg q(x) \vee r(x)$. It is clear that each formula $\phi \in \mathcal{L}_0$ can be transformed into an equivalent formula in negated normal form and we write $NNF(\phi)$ to denote this formula.

Definition 8.3. *(Stratified Derived Predicate Axiom Set)*
A set of derived predicate axioms is called *stratified* iff there exists a partition of the set of derived predicates $\mathcal{D}$ into (non-empty) subsets $\{\mathcal{D}_i, 1 \leq i \leq n\}$ such that for every $d_i \in \mathcal{D}_i$ and every axiom $\forall\mathbf{x}(\phi[\mathbf{x}] \rightarrow d_i(\mathbf{x}))$ we have that:

- if $d_j \in \mathcal{D}_j$ occurs positively in $NNF(\phi)$, then $j \leq i$.
- if $d_j \in \mathcal{D}_j$ occurs negated in $NNF(\phi)$, then $j < i$.

The semantics of $\mathcal{L}_0$ is defined relative to a state S of basic facts and a set of derived facts D. We first present this semantics and then show how the set D of derived facts can be obtained from a set of basic facts and a stratified axiom set. The *closed world assumption* applies, so any ground positive literal not in S is assumed to be false.

Definition 8.4. *(Truth conditions)* Let $S \subseteq F_b$ be a set of basic facts and $D \subseteq F_d$ be a set of derived facts. The truth conditions of *closed* formulas from $\mathcal{L}_0$ are defined by:

$$
\begin{array}{lll}
\langle S, D\rangle \models p(\mathbf{t}) & \text{iff} & p(\mathbf{t}) \in S \cup D \\
\langle S, D\rangle \models \neg\phi & \text{iff} & \langle S, D\rangle \not\models \phi \\
\langle S, D\rangle \models (\phi_1 \wedge \phi_2) & \text{iff} & \langle S, D\rangle \models \phi_1 \text{ and } \langle S, D\rangle \models \phi_2 \\
\langle S, D\rangle \models (\phi_1 \vee \phi_2) & \text{iff} & \langle S, D\rangle \models \phi_1 \text{ or } \langle S, D\rangle \models \phi_2 \\
\langle S, D\rangle \models \forall x(\phi) & \text{iff} & \langle S, D\rangle \models \phi[t/x] \text{ for all ground } t \in \mathcal{T} \\
\langle S, D\rangle \models \exists x(\phi) & \text{iff} & \langle S, D\rangle \models \phi[t/x] \text{ for some ground } t \in \mathcal{T}
\end{array}
$$

We have assumed that all terms refer to different objects, which is also known as the *unique names assumption*. In addition, it is assumed that all objects are named by some term. These are common assumptions in logic programming, and in PDDL. By making these assumptions a substitutional interpretation of quantifiers can be used as we have done in Definition 8.4.

[5] More precisely, recursion through negation is not allowed.

Formulas with free variables may be used to compute answers, i.e. substitutions. Substitutions are used to instantiate variables to values in the domain, i.e. bind variables to closed terms.

Definition 8.5. *(Substitution)*
A *substitution* is a mapping from variables $\mathcal{V}$ to *closed* terms in $\mathcal{T}$.

For details of what it means to apply a substitution θ to an expression e with result $e\theta$ see [289]. $e\theta$ is also called an *instance* of e. We use $\theta[t/x]$ to denote the substition σ such that $\sigma(x) = t$ and $\sigma(y) = \theta(y)$ for all $y \neq x$ in the range of θ.

The semantics of open formulas $\phi \in \mathcal{L}_0$ is defined by $\langle S, D\rangle \models \phi$ iff there is a substitution θ such that $\phi\theta$ is closed and $\langle S, D\rangle \models \phi\theta$.

In the definition of the semantics of $\mathcal{L}_0$ we have assumed that the set of derived facts D was given. Intuitively, we can derive $d(\mathbf{t})$ using axiom $a = \forall \mathbf{x}(\phi \rightarrow d(\mathbf{x}))$ if we have $\langle S, D\rangle \models \phi[\mathbf{t}]$, and add $d(\mathbf{t})$ to D if not already present; we write $[\![a]\!](S, D) = \{d(\mathbf{t}) \mid \langle S, D\rangle \models \phi[\mathbf{t}], \mathbf{t} \text{ is ground}\}$ to denote these consequences. $[\![a]\!](S, D)$ yields all immediate consequences of a stratified axiom set given that a pair $\langle S, D\rangle$ has been fixed.

Then the set of consequences of an axiom set A can be computed as follows, assuming that we have a stratification $\{A_i, 1 \leq i \leq n\}$ of A:

$$
\begin{aligned}
&[\![A]\!]_0(S) = \emptyset, \text{ and, for all } 1 \leq i \leq n\text{:} \\
&[\![A]\!]_i(S) = \bigcap \left\{ D \mid \bigcup_{a \in A_i} [\![a]\!](S, D) \cup [\![A]\!]_{i-1}(S) \subseteq D \right\}
\end{aligned}
$$

The set of all derived facts, written $[\![A]\!](S)$, that can be obtained from A then is defined as $[\![A]\!]_n(S)$. Using this set we can define the consequences of a (belief) state S relative to a set of derived predicate axioms A as follows.

Definition 8.6.

$$S \models_A \phi \text{ iff } \langle S, [\![A]\!](S)\rangle \models \phi$$

The semantics of axioms has been defined as a fixed point above. It is well-known, however, that this semantics corresponds with a logical semantics of the *completion* of an axiom set. See, for example, the discussion of logic programming for "unrestricted" programs in [289]. As this equivalence is useful for showing that the verification logic introduced below can be used to characterize Goal agents, we briefly discuss the key results that we need. Details can be found in the Appendix.

We first introduce the *completion comp*(A) of a *finite* stratified axiom set A. Intuitively, the completion *comp*(A) replaces implications with equivalences [17, 289]. We assume that with each derived predicate $d \in \mathcal{D}$ at least one axiom is associated. Then, in our setting, completion can be defined as follows.

Definition 8.7. *(Completion)*
Let A be a finite stratified axiom set. Then the *completion comp*(A) of that set is obtained by applying the following operations to this set:

1. For each derived predicate $d \in \mathcal{D}$, collect all associated axioms of the form $\forall \mathbf{x}(\phi_1 \rightarrow d(\mathbf{x})), \ldots, \forall \mathbf{x}(\phi_n \rightarrow d(\mathbf{x}))$. Replace these axioms by:

$$\forall \mathbf{x}((\phi_1 \vee \ldots \vee \phi_n) \rightarrow d(\mathbf{x}))$$

   Note that variables may need to be renamed to ensure that all axioms for d have $d(\mathbf{x})$ as their head with unique variables $\mathbf{x}$.
2. For each of the formulas obtained in the previous step, replace $\rightarrow$ with $\leftrightarrow$, i.e. replace each formula $\forall \mathbf{x}(\phi \rightarrow d(\mathbf{x}))$ by $\forall \mathbf{x}(\phi \leftrightarrow d(\mathbf{x}))$.

It is well-known that the completion of a stratified axiom set is consistent [289].

Definition 8.8. *(Answer, Correct Answer)*
Let S be a set of ground atoms and A be a stratified axiom set. A substitution θ is an *answer* for ϕ with respect to S and A if θ is a substitution for free variables in ϕ and $S \models_A \forall(\phi\theta)$. A substitution θ is a *correct* answer with respect to S and A if $comp(A) \cup S \models_c \forall(\phi\theta)$ and θ is an answer for ϕ. Here, $\models_c$ refers to the usual consequence relation for *classical* first-order logic.

The completion of a stratified axiom set defines the meaning of such a set in terms of classical first-order semantics. It shows that a *declarative reading* can be imposed on a stratified axiom set. It may moreover be used to verify that the semantics of Definition 8.4 is well-defined.

Theorem 8.1. (Correctness)
Let S be a set of basic facts, A be a stratified axiom set, and θ be an answer for ϕ with respect to S and A. Then θ is a correct answer. That is, we have:

$$S \models_A \forall(\phi\theta) \;\; \textit{iff} \;\; comp(A) \cup S \models_c \forall(\phi\theta)$$

Proof. See the Appendix.

8.3.3 Mental States

The knowledge representation language $\mathcal{L}_0$ is used by GOAL agents to represent their knowledge, beliefs and goals. We first discuss knowledge and beliefs. The difference between knowledge and beliefs is based on the distinction between derived and basic predicates discussed above. Knowledge is assumed to be *static* and concerns conceptual and domain knowledge which is defined using derived predicate axioms. Beliefs may change and represent the basic facts the agent believes to be true about the environment. Accordingly, the knowledge base maintained by a GOAL agent is a set of stratified derived predicate axioms as defined above, and the belief base is a set of ground atoms that only use basic predicates.

Although it is common in planning to allow complex goal descriptions, in contrast with typical planning problems [192] the goals maintained by a Goal agent may change over time. Moreover, a Goal agent needs to be able to inspect its goals and therefore it is important that the goal base can be efficiently queried. For these reasons, the goal base consists of conjunctions of ground atoms here. For example, $p(a) \wedge q(b)$ and $r(a) \wedge r(b) \wedge r(c)$ may be part of a goal base but $\neg p(c)$ and $\exists x(q(x) \rightarrow r(x))$ may not. It is clear that a conjunction of ground atoms can be identified with the set of corresponding atoms and we will abuse notation here and will also denote the corresponding set of ground atoms by means of a conjunction. This will allow us to write $p(a) \wedge q(b) \subseteq F$ to denote that $p(a)$ and $q(b)$ are in the set F of facts. We will make use of this below in the definition of the semantics of mental state conditions.

Definition 8.9. *(Mental State)*
A *mental state* is a triple $\langle K, \Sigma, \Gamma \rangle$ where K is a finite, stratified derived predicate axiom set, called a *knowledge base*, $\Sigma \subseteq F_b$ is a *belief base* that consists of a finite set of basic facts, and $\Gamma \subseteq 2^{F_b}$ is a *goal base* that consists of a finite set of finite subsets (or, conjunctions) of basic facts. Finally, the following *rationality constraint* is imposed on mental states:

$$\forall \gamma \in \Gamma : \Sigma \not\models_K \gamma$$

This constraint excludes mental states where a goal in the goal base is believed to be achieved. This constraint imposed on mental states is motivated by the principle that agents should not invest resources into achieving goals that have already been achieved. Goals thus are viewed as *achievement goals*, i.e. states that the agent wants to realize at some future moment.

It is usual to impose various *rationality constraints* on mental states [60]. These constraints typically include that (i) the knowledge base combined with the belief base is consistent, that (ii) individual goals are consistent with the knowledge base, and that (iii) no goal in the goal base is believed to be (completely) achieved. Constraint (iii) is part of the definition of a mental state but we do not need to impose the first two constraints explicitly as these follow by definition; both the belief base and goal base consist of basic facts only, and the knowledge base only consists of rules for derived predicates. Note that although goals cannot be logically inconsistent it is still possible to have conflicting goals, e.g. $on(a,b)$ and $on(b,a)$ in a Blocks World where one block cannot be simultaneously on top of and below another block.

In order to select actions an agent needs to be able to inspect its mental state. In Goal, an agent can do so by means of *mental state conditions*. Mental state conditions are conditions on the mental state of an agent, expressing that an agent believes something is the case, has a particular goal, or a combination of the two. Special operators to inspect the belief base of an agent, we use **bel**(φ) here, and to inspect the goal base of an agent, we use **goal**(φ) here, are introduced to do so. In addition, a special operator **o-goal**(ϕ) will be useful later and represents that ϕ is the "only goal" of an agent. This operator will allow us to introduce 'successor state axioms'

for goals below. We allow boolean combinations of these basic conditions but do not allow the nesting of operators. Basic conditions may be combined into a conjunction by means of $\wedge$ and negated by means of $\neg$. For example, $\mathbf{goal}(\varphi) \wedge \neg \mathbf{bel}(\varphi)$ with $\varphi \in \mathcal{L}_0$ is a mental state condition, but $\mathbf{bel}(\mathbf{goal}(\varphi))$ which has nested operators is not.

Definition 8.10. *(Syntax of Mental State Conditions)*
The language $\mathcal{L}_\Psi$ of mental state conditions, with typical elements ψ, is defined by:

$$\begin{array}{lll} \phi & ::= & \text{any element from } \mathcal{L}_0 \\ \psi \in \mathcal{L}_\Psi & ::= & \mathbf{bel}(\phi) \mid \mathbf{goal}(\phi) \mid \mathbf{o\text{-}goal}(\phi) \mid \psi \wedge \psi \mid \neg\psi \end{array}$$

Note that we allow variables in mental state conditions, i.e. a mental state condition ψ does not need to be closed. A mental state condition with free variables can be used in an agent program to retrieve particular bindings for these free variables. That is, mental state conditions can be used to compute a *substitution*.

The next step is to define the semantics of *mental state conditions*. The meaning of a mental state condition is derived from the mental state of the agent. A belief condition $\mathbf{bel}(\phi)$ is true whenever ϕ follows from the belief base combined with the knowledge stored in the agent's knowledge base. The meaning of a goal condition $\mathbf{goal}(\phi)$ is slightly different from that of a belief condition. Instead of simply defining $\mathbf{goal}(\phi)$ to be true whenever ϕ follows from *all* of the agent's goals (combined with the knowledge in the knowledge base), we will define $\mathbf{goal}(\phi)$ to be true whenever ϕ follows from *one* of the agent's goals (and the agent's knowledge). The intuition here is that each goal in the goal base has an implicit *temporal dimension* and two different goals need not be achieved at the same time. Goals are thus used to represent *achievement goals*. Finally, $\mathbf{o\text{-}goal}(\phi)$ is true iff all goals of the agent are logically equivalent with ϕ; that is, the only goal present is the goal ϕ.

Definition 8.11. *(Semantics of Mental State Conditions)*
Let $m = \langle K, \Sigma, \Gamma \rangle$ be a mental state. The semantics of *closed* mental state conditions ψ is defined by the following semantic clauses:

$$\begin{array}{lll} m \models_\Psi \mathbf{bel}(\phi) & \text{iff} & \Sigma \models_K \phi, \\ m \models_\Psi \mathbf{goal}(\phi) & \text{iff} & \exists \gamma \in \Gamma : \gamma \models_K \phi, \\ m \models_\Psi \mathbf{o\text{-}goal}(\phi) & \text{iff} & m \models_\Psi \mathbf{goal}(\phi) \text{ and } \forall \phi' (m \models_\Psi \mathbf{goal}(\phi') \Rightarrow \models_c \phi \leftrightarrow \phi'), \\ m \models_\Psi \psi_1 \wedge \psi_2 & \text{iff} & m \models_\Psi \psi_1 \text{ and } m \models_\Psi \psi_2, \\ m \models_\Psi \neg\psi & \text{iff} & m \not\models_\Psi \psi. \end{array}$$

As before, for open formulas $\psi \in \mathcal{L}_\Psi$ we define $m \models_\Psi \psi$ iff there is a substitution such that $\psi\theta$ is closed and $m \models_\Psi \psi\theta$.

Note that in the definition of the semantics of mental state conditions we have been careful to distinguish between the consequence relation that is defined, denoted by $\models_\Psi$, and the consequence relation $\models$ defined in Definition 8.4. The definition thus shows how the meaning of a mental state condition can be derived from the semantics of the underlying knowledge representation language.

Proposition 8.1. *Let $m = \langle K, \Sigma, \Gamma \rangle$ be a mental state. Then we have:*

$$m \models_\Psi \neg \boldsymbol{bel}(\phi) \;\; \textit{iff} \;\; \Sigma \models_K \neg\phi$$

Proof. We have: $m \models_\Psi \neg \boldsymbol{bel}(\phi)$ *iff* $m \not\models_\Psi \boldsymbol{bel}(\phi)$ *iff* $\Sigma \not\models_K \phi$ *iff* $\langle \Sigma, [\![K]\!](\Sigma)\rangle \not\models \phi$ *iff* $\langle \Sigma, [\![K]\!](\Sigma)\rangle \models \neg\phi$ *iff* $\Sigma \models_K \neg\phi$. □

Proposition 8.1 is a direct consequence of the closed world assumption. That is, when ϕ is not believed to be the case, by the closed world assumption it then follows that $\neg\phi$. In other words, we have that $\neg\mathbf{bel}(\phi)$ is equivalent with $\mathbf{bel}(\neg\phi)$.

P1	if ψ is an instantiation of a classical tautology, then $\models_\Psi \psi$.
P2	if $\models_c \phi$, then $\models_\Psi \mathbf{bel}(\phi)$.
P3	$\models_\Psi \mathbf{bel}(\phi \rightarrow \phi') \rightarrow (\mathbf{bel}(\phi) \rightarrow \mathbf{bel}(\phi'))$.
P4	$\models_\Psi \neg\mathbf{bel}(\bot)$.
P5	$\models_\Psi \neg\mathbf{bel}(\phi) \leftrightarrow \mathbf{bel}(\neg\phi)$.
P6	$\models_\Psi \forall x(\mathbf{bel}(\phi)) \leftrightarrow \mathbf{bel}(\forall x(\phi))$.
P7	$\not\models_\Psi \mathbf{goal}(\top)$.
P8	$\models_\Psi \neg\mathbf{goal}(\bot)$.
P9	if $\models \phi \rightarrow \phi'$, then $\models_\Psi \mathbf{goal}(\phi) \rightarrow \mathbf{goal}(\phi')$.
P10	$\models_\Psi \mathbf{goal}(\forall x(\phi)) \rightarrow \forall x(\mathbf{goal}(\phi))$.

Table 8.1 Properties of Beliefs and Goals

We briefly discuss some of the properties listed in Table 8.1. The first property (P1) states that mental state conditions that instantiate classical tautologies such as $\mathbf{bel}(\phi) \vee \neg\mathbf{bel}(\phi)$ and $\mathbf{goal}(\phi) \rightarrow (\mathbf{bel}(\phi') \rightarrow \mathbf{goal}(\phi))$ are valid with respect to $\models_\Psi$. Property (P2) corresponds with the usual necessitation rule of modal logic and states that an agent believes all validities of the base logic. (P3) expresses that the belief modality distributes over implication. This implies that the beliefs of an agent are closed under logical consequence. Property (P4) states that the beliefs of an agent are consistent. In essence, the belief operator thus satisfies the properties of the system KD (see e.g. [314]). Although in its current presentation, it is not allowed to nest belief or goal operators in mental state conditions in GOAL, from [314], section 1.7, we conclude that we may assume *as if* our agent has positive $\mathbf{bel}(\phi) \rightarrow \mathbf{bel}(\mathbf{bel}(\phi))$ and negative $\neg\mathbf{bel}(\phi) \rightarrow \mathbf{bel}(\neg\mathbf{bel}(\phi))$ introspective properties: every formula in the system KD45 (which is KD together with the two mentioned properties) is equivalent to a formula without nestings of operators. Property (P7) shows that $\neg\mathbf{goal}(\top)$ can be used to express that an agent has *no* goals. Property (P8) states that an agent also does not have inconsistent goals, that is, we have $\models_c \neg\mathbf{goal}(\bot)$. Property (P9) states that the goal operator is closed under implication in the base language. That is, whenever $\phi \rightarrow \phi'$ is valid in the base language then we also have that $\mathbf{goal}(\phi)$ implies $\mathbf{goal}(\phi')$. This is a difference with the presentation in [60] which is due to the more basic goal modality we have introduced here. It is important to note here that we do *not* that $\mathbf{bel}(\phi) \wedge \mathbf{goal}(\phi)$ is inconsistent. Finally, property (P10) is valid, but the implication cannot be reversed: $\forall x(\mathbf{goal}(\phi)) \rightarrow \mathbf{goal}(\forall x(\phi))$ is not valid.

It is clear from the properties discussed that the **goal** operator does not correspond with the more common sense notion of a goal but instead is an operator mainly introduced for technical reasons. The reason for using the label **goal** is to clearly differentiate it from the **bel** operator and make clear that this operator is related to the motivational attitudes of an agent. The **goal** operator is a *primitve* operator that does not match completely with the common sense notion of a goal that needs to be achieved in the future. The **goal** operator, however, may be used to define so-called *achievement goals* that usually require effort from an agent in order to realize the goal. The main characteristic which sets an achievement goal apart from "primitives" goals thus is that they are not believed to be achieved already. As noticed, it is possible to define the concept of an achievement goal and to introduce an achievement goal **a-goal** operator using the primitive **goal** operator and the belief **bel** operator. It is also useful to be able to express that a goal has been (partially) achieved. We therefore also introduce a "goal achieved" **goal-a** operator to be able to state that (part of) a goal is believed to be achieved. This operator can also be defined using the **goal** and **bel** operator.

Definition 8.12. *(Achievement Goal and Goal Achieved Operators)*
The *achievement goal* **a-goal**(ϕ) operator and the *goal achieved* **goal-a**(ϕ) operator are defined by:

$$\textbf{a-goal}(\phi) \stackrel{df}{=} \textbf{goal}(\phi) \wedge \neg \textbf{bel}(\phi),$$
$$\textbf{goal-a}(\phi) \stackrel{df}{=} \textbf{goal}(\phi) \wedge \textbf{bel}(\phi).$$

Both of these operators are useful when writing agent programs. The first is useful to derive whether a part of a goal has not yet been (believed to be) achieved whereas the second is useful to derive whether a part of a goal has already been (believed to be) achieved. It should be noted that an agent can only believe that part of one of its goals has been achieved but cannot believe that one of its goals has been *completely* achieved as such goals are removed automatically from the goal base. That is, whenever we have $\gamma \in \Gamma$ we must have **a-goal**(γ), or, equivalently, **goal**(γ) $\wedge$ $\neg$**bel**(γ) since it is not allowed by the third rationality constraint in Definition 8.9 that an agent believes γ in that case (see also (P21) and (P22) in Table 8.2).

Table 8.2 lists some properties of the **a-goal**, **goal-a**, and **o-goal** operators.[6] The main difference between the **a-goal** and **goal-a** operators concern Properties (P12) and (P17) and Properties (P15) and (P20), respectively. Property (P12) expresses that an achievement goal $\phi \wedge (\phi \rightarrow \phi')$ does not imply an achievement goal ϕ'. This property avoids the *side effect problem*. The **goal-a** operator, however, is closed under such effects as any side effect of a goal that has been achieved also is realized by implication. Properties (P15) and (P20) highlight the key difference between achievement goals and goals achieved: achievement goals are not believed to be achieved, whereas goals achieved are believed to be achieved.

[6] The properties of the **a-goal** operator are the same as those for the G operator listed in Lemma 2.4 in [60].

P11	$\not\models_\psi$ **a-goal**$(\phi \rightarrow \phi') \rightarrow$ (**a-goal**$(\phi) \rightarrow$ **a-goal**$(\phi'))$.
P12	$\not\models_\psi$ **a-goal**$(\phi \wedge (\phi \rightarrow \phi')) \rightarrow$ **a-goal**(ϕ').
P13	$\not\models_\psi$ (**a-goal**$(\phi) \wedge$ **a-goal**$(\phi')) \rightarrow$ **a-goal**$(\phi \wedge \phi')$.
P14	if $\models (\phi \leftrightarrow \phi')$, then $\models_\psi$ **a-goal**$(\phi) \leftrightarrow$ **a-goal**(ϕ').
P15	$\models_\psi$ **a-goal**$(\phi) \rightarrow \neg$**bel**(ϕ).
P16	$\not\models_c$ **goal-a**$(\phi \rightarrow \phi') \rightarrow$ (**goal-a**$(\phi) \rightarrow$ **goal-a**$(\phi'))$.
P17	$\models_\psi$ **goal-a**$(\phi \wedge (\phi \rightarrow \phi')) \rightarrow$ **goal-a**(ϕ').
P18	$\not\models_\psi$ (**goal-a**$(\phi) \wedge$ **goal-a**$(\phi')) \rightarrow$ **goal-a**$(\phi \wedge \phi')$.
P19	if $\models (\phi \leftrightarrow \phi')$, then $\models_\psi$ **goal-a**$(\phi) \leftrightarrow$ **goal-a**(ϕ').
P20	$\models_\psi$ **goal-a**$(\phi) \rightarrow$ **bel**(ϕ).
P21	$\models_\psi$ **o-goal**$(\phi) \rightarrow$ **goal**(ϕ).
P22	$\models_\psi$ **o-goal**$(\phi) \rightarrow \neg$**bel**(ϕ).
P23	$\not\models_\psi$ **o-goal**$(\phi \rightarrow \phi') \rightarrow$ (**o-goal**$(\phi) \rightarrow$ **o-goal**$(\phi'))$.

Table 8.2 Properties of Achievement Goals, Goals Achieved, and Only Goals

8.3.4 Actions and Action Selection

A GOAL agent derives its choice of action from its goals and beliefs (in combination with its knowledge). Action selection is implemented by means of so-called *action rules* that inspect the mental state of the agent. Actions are performed to change the environment. An agent keeps track of such changes by updating its beliefs. The updates asscociated with the execution of an action are provided by so-called *action specifications*. Actions may have *conditional effects* [336].[7] For example, the result of performing the action of switching the light button depends on the current state and may result in the light being either on or off. The effect of the light switching action thus depends on the state in which the action is executed.

We present a formal, operational semantics for GOAL with actions with conditional effects and use Plotkin-style transition semantics [340] to do so. This semantics is a computational semantics that provides a specification for executing a GOAL agent on a machine.

We distinguish between two types of actions: user-specified actions and built-in actions. Built-in actions are part of the GOAL language and include an action for *adding and deleting beliefs* and for *adopting* and *dropping goals*. The **insert**(ϕ) action, where ϕ should be a conjunction of basic literals, adds facts that occur positively in ϕ to the belief base and removes facts that occur negatively in ϕ from the belief base. This action can always be performed and has a precondition $\top$. The **adopt**(ϕ) action, where ϕ should be a conjunction of basic facts, adds a goal to the goal base. The **adopt**(ϕ) action can only be performed if ϕ is not believed to be the case; that is, the precondition of **adopt**(ϕ) is $\neg$**bel**(ϕ).[8] Finally, the **drop**(ϕ) ac-

[7] This is an extension of GOAL as presented in [60] introduced in [228].

[8] The condition that ϕ or a logically equivalent formula is not already present in the goal base may be added but is less important in this context but is relevant for efficiency reasons to avoid having to evaluate multiple times whether one and the same goal has been achieved.

tion, where ϕ should again be a conjunction of basic facts, removes any goal in the goal base that implies ϕ. The precondition of **drop**(ϕ) is $\top$ and thus can always be performed.

The semantics of these actions is formally defined by means of a *mental state transformer function* $\mathcal{M}$. This function maps an action $a(\mathbf{t})$ and a mental state m to a new mental state m' that is the result of performing the action. It is useful to introduce some notation here. We use $pos(\phi)$ and $neg(\phi)$ to denote the set facts that occur *positively* respectively *negatively* in a conjunction of literals ϕ.

Definition 8.13. *(Semantics of Built-in Actions)*
Let $m = \langle K, \Sigma, \Gamma \rangle$ be a mental state. The *mental state transformer function* $\mathcal{M}$ is defined as follows for the built-in actions **insert**(ϕ), **adopt**(ϕ), and **drop**(ϕ), where ϕ needs to be of the appropriate form: [9]

$$\mathcal{M}(\mathbf{insert}(\phi), m) = \langle K, (\Sigma \setminus neg(\phi)) \cup pos(\phi), \Gamma \rangle.$$

$$\mathcal{M}(\mathbf{adopt}(\phi), m) = \begin{cases} \langle K, \Sigma, \Gamma \cup \{\phi\} \rangle & \text{if } m \models_{\Psi} \neg \mathbf{bel}(\phi), \\ \text{undefined} & \text{otherwise.} \end{cases}$$

$$\mathcal{M}(\mathbf{drop}(\phi), m) = \langle K, \Sigma, \Gamma \setminus \{\phi' \in \Gamma \mid \phi' \models_c \phi\} \rangle.$$

To enable a programmer to add user-specified actions to an agent program we need a language for specifying when an action can be performed and what the effects of performing an action are. Actions are written as $a(\mathbf{t})$ where a is the *name* of the action and $\mathbf{t}$ are the *parameters* of the action. Preconditions specify when an action can be performed. These conditions can be specified using the knowledge representation language $\mathcal{L}_0$. The effects of an action may be conditional on the state in which the action is performed. To express such conditional effects we use statements of the form $\phi \Rightarrow \phi'$ where ϕ is called the *condition* and ϕ' the *effect*. Intuitively, an action with *conditional effect* $\phi \Rightarrow \phi'$ means that if the action is performed in a state where ϕ is true, then the effect of the action is ϕ'. When the condition ϕ is $\top$ we also simply write ϕ' to denote the effect. Free variables in a conditional effect may be bound universally and we write $\forall \mathbf{x}(\phi \Rightarrow \phi')$. Finally, multiple conditional effects may be associated with an action and in that case we write $\forall \mathbf{x}_1(\phi_1 \Rightarrow \phi'_1) \wedge \ldots \wedge \forall \mathbf{x}_n(\phi_n \Rightarrow \phi'_n)$. Finally, for specifying the preconditions and effects of an action $a(\mathbf{t})$ we use a Hoare-triple-style notation. That is, we use triples consisting of a precondition, an action and a conjunction of conditional effects to specify the precondition and effects of a particular action. Note, however, that although conditional effects are part of the postcondition of an action, the conditions of such effects need to be evaluated in the state where the action is executed and in this respect is similar to a precondition. The following definition summarizes the previous discussion.

Definition 8.14. *(Conditional Effect, Action Specification)*

[9] Note that since goals are conjunctions of basic facts, the formal semantics given here for the **drop** action that uses $\models_c$ is easily replaced by an efficient computational mechanism.

- A *conditional effect statement* is an expression of the form $\forall \mathbf{x}(\phi \Rightarrow \phi')$, where $\phi \in \mathcal{L}_0$ and ϕ' is a conjunction of basic literals. The quantor $\forall \mathbf{x}$ may be absent and whenever ϕ is $\top$ we also write $\forall \mathbf{x}(\phi')$.
- An *action specification* is a triple written as:

$$\{\ \phi\ \}\ a(\mathbf{t})\ \{\ \forall \mathbf{x}_1(\phi_1 \Rightarrow \phi'_1) \wedge \ldots \wedge \forall \mathbf{x}_n(\phi_n \Rightarrow \phi'_n)\ \}$$

  where ϕ is a formula from $\mathcal{L}_0$ called the *precondition*, $a(\mathbf{t})$ is an action with *name* a and *parameters* $\mathbf{t}$, and $\forall \mathbf{x}_1(\phi_1 \Rightarrow \phi'_1) \wedge \ldots \wedge \forall \mathbf{x}_n(\phi_n \Rightarrow \phi'_n)$ is a conjunction of conditional effect statements, which is also called *postcondition*. All free variables in the postcondition must occur free in either the precondition or the action parameters. Finally, the postcondition is required to be *consistent* (see Definition 8.15 below).

Note the condition on free variables in the definition of an action specification. The free variables that occur in the postcondition need to occur free in the precondition or action parameters in order to ensure they are instantiated. An action can only be performed when all free variables in the action parameters and the postcondition have been instantiated.

In GOAL, a precondition is evaluated on the belief base of the agent. This means that an agent believes it can perform an action if the agent believes the associated precondition of that action. An action may affect both the beliefs and goals of an agent. An agent's knowledge base is static and does not change since it is used to represent conceptual and domain knowledge that does not change. The postcondition of an action specifies how the belief base of an agent should be updated. Intuitively, for each conditional effect $\phi \Rightarrow \phi'$ if ϕ is believed then the facts that occur positively in ϕ' are added to the belief base and the facts that occur negatively in ϕ' are removed from the belief base.

Definition 8.15. *(Positive and Negative Effects of an Action, Consistency)*
Let $\{\ \phi\ \}\ a(\mathbf{t})\ \{\ \phi'\ \}$ be an instantiation of an action specification such that ϕ and $a(\mathbf{t})$ are ground, which implies that $\phi' = \forall \mathbf{x}_1(\phi_1 \Rightarrow \phi'_1) \wedge \ldots \wedge \forall \mathbf{x}_n(\phi_n \Rightarrow \phi'_n)$ is also ground. Then the *positive effects* respectively the *negative effects* in a mental state m are defined by:

$$Eff^{+}(\phi', m) = \bigcup_i \{\ at \in pos(\phi'_i) \mid m \models_{\Psi} \mathbf{bel}(\phi_i)\}$$
$$Eff^{-}(\phi', m) = \bigcup_i \{\ at \in neg(\phi'_i) \mid m \models_{\Psi} \mathbf{bel}(\phi_i)\}$$

A postcondition ϕ' is said to be *consistent* if for all mental states m and all instantiations of an action specification the set $Eff^{+}(\phi', m) \cup \neg Eff^{-}(\phi', m)$ is consistent, i.e. if this set does not contain complementary literals.[10]

[10] Where $\neg T$ denotes the set $\{\neg\phi \mid \phi \in T\}$, with $T \subseteq \mathcal{L}_0$.

After updating the beliefs, an agent also needs to check whether any of its goals have been realized and can be removed from its goal base. A GOAL agent only removes goals that have been completely achieved. A goal such as $on(a,b) \wedge on(b,table)$ is not removed or replaced by $on(b,table)$ since the goal has been achieved only when at the same time block a is on block b and b is on the table. The semantics of user-specified actions is again defined by means of the mental state transformer function $\mathcal{M}$.

Definition 8.16. *(Semantics of User-Specified Actions)*
Let $m = \langle K, \Sigma, \Gamma \rangle$ be a mental state, and $\{ \phi \}\, a(\mathbf{t})\, \{ \phi' \}$ be an instantiation of an action specification such that ϕ and $a(\mathbf{t})$ are ground. Then the *mental state transformer function* $\mathcal{M}$ is defined as follows for action $a(\mathbf{t})$:

$$\mathcal{M}(a(\mathbf{t}), m) = \begin{cases} \langle K, \Sigma', \Gamma' \rangle & \text{if } m \models_{\Psi} \mathbf{bel}(\phi) \\ \text{undefined} & \text{otherwise} \end{cases}$$

where:

- $\Sigma' = (\Sigma \setminus \mathit{Eff}^{-}(\phi')) \cup \mathit{Eff}^{+}(\phi')$.
- $\Gamma' = \Gamma \setminus \{\phi \in \Gamma \mid \Sigma \models_K \phi\}$.

Note that the formula ϕ' in Definition 8.16 is closed since any free variables in a postcondition ϕ' need to be free in either the precondition ϕ or action parameters $a(\mathbf{t})$.

GOAL agents derive their choice of action from their beliefs and goals. They do so by means of *action rules* of the form **if** ψ **then** α. Here ψ is a mental state condition and α an action, either built-in or user-specified. An action rule specifies that action α may be selected for execution if the mental state condition ψ and the precondition of action α hold. In that case, we say that action α is an *option*. At runtime, a GOAL agent non-deterministically selects an action from the set of options to perform. This is expressed in the following transition rule, describing how an agent's mental state changes from one to another.[11]

Definition 8.17. *(Action Semantics)*
Let m be a mental state, and **if** ψ **then** α be an action rule, and θ a substitution. The transition relation $\xrightarrow{\alpha\theta}$ is the smallest relation induced by the following transition rule.

$$\frac{m \models_{\Psi} \psi\theta \quad \mathcal{M}(\alpha\theta, m) \text{ is defined}}{m \xrightarrow{\alpha\theta} \mathcal{M}(\alpha\theta, m)}$$

A GOAL agent (program) consists of the knowledge, initial beliefs and goals, action rules and action specifications.

[11] A transition rule is an inference rule for deriving transitions or computation steps. The statements above the line are called the premises of the rule and the transition below the line is called the conclusion. See also [340].

Definition 8.18. *(Goal Agent Program)*
A Goal *agent program* is a tuple $\langle K, \Sigma, \Gamma, \Pi, A \rangle$ with:

- $\langle K, \Sigma, \Gamma \rangle$ a mental state,
- Π a set of action rules, and
- A a set of action specifications.

Below we will assume that there is exactly one action specification associated with each action name a. Although it is possible to use multiple action specifications in a Goal agent, this assumption is introduced to somewhat simplify the presentation below.

The execution of a Goal agent results in a *run* or computation. We define a computation as a sequence of mental states and actions, such that each mental state can be obtained from the previous by applying the transition rule of Definition 8.17. As Goal agents are non-deterministic, the semantics of a Goal agent is defined as the *set* of possible computations of the Goal agent, where all computations start in the initial mental state of the agent.

Definition 8.19. *(Run)*
A *run* or *computation* of an agent $Agt = \langle K, \Sigma, \Gamma, \Pi, A \rangle$, typically denoted by r, is an infinite sequence of mental states and actions $m_0, \alpha_0, m_1, \alpha_1, m_2, \alpha_2, \ldots$ such that:

- $m_0 = \langle K, \Sigma, \Gamma \rangle$, and
- for each i we have either that:
  - $m_i \xrightarrow{\alpha_i} m_{i+1}$ can be derived using the transition rule of Definition 8.17, or
  - for all $j > i$, $m_j = m_i$ and $m_i \not\xrightarrow{\alpha} m'$ for any α and m', and $\alpha_i = \textbf{skip}$.[12]

We also write r_i^m to denote the ith mental state and r_i^a to denote the ith action. The meaning $\mathcal{R}_{Agt}$ of a Goal agent Agt is the set of all possible runs of that agent.

Observe that a computation is infinite by definition, even if the agent is not able to perform any actions anymore from some point in time on. In the latter case, the agent is assumed to perform no action at all, which is represented by **skip**. Also note that the concept of a computation is a general notion in program semantics that is not particular to Goal. The notion of a computation can be defined for any (agent) programming language that is provided with a well-defined operational semantics.

[12] Implicitly, this also defines the semantics of **skip**. The label **skip** denotes the action that does not change the mental state of an agent.

8.4 Verifying Goal Agent Programs

The verification logic for Goal that we present here is an extension of linear temporal logic (LTL) with an action, belief and goal operator. It is similar to the verification framework presented in [60]. However, the verification framework presented here does not consist of two different components, a Hoare logic component and a linear temporal logic component, as in [60]. Instead, the Hoare logic for reasoning about actions in [60] is replaced by an action theory represented in the temporal verification logic.

A second difference with [60] is that the verification framework presented here is less abstract. In particular, the verification logic presented here incorporates Reiter's solution to the frame problem [364].

The differences with the very generic approach presented in [60] have important implications. The approach presented here introduces a deductive approach to the verification of agents that is obtained by a direct translation of an agent program into the logic. We will discuss in more detail what this means in section 8.4.2.

8.4.1 Verification Logic

To obtain a verification logic for Goal agents temporal operators are added on top of mental state conditions to be able to express temporal properties over runs and an action operator **done**(α) is introduced.

Definition 8.20. *(Temporal Language: Syntax)*
The *temporal language* $\mathcal{L}_G$, with typical elements χ, χ', is defined by:

$$\chi \in \mathcal{L}_G ::= \psi \in \mathcal{L}_\Psi \mid \mathbf{done}(\alpha) \mid \neg\chi \mid \chi \wedge \chi \mid \forall x(\chi) \mid \bigcirc\chi \mid \chi \,\mathbf{until}\, \chi$$

Using the **until** operator, other temporal operators such as the "sometime in the future operator" $\Diamond$ and the "always in the future operator" $\Box$ can be introduced as abbreviations for $\Diamond\psi ::= \top \,\mathbf{until}\, \psi$ and $\Box\psi ::= \neg\Diamond\neg\psi$.

Although the language $\mathcal{L}_G$ is intended for verification of runs of Goal agents, the semantics of $\mathcal{L}_G$ is defined more generally relative to a trace t. Each *time point* (t,i) denotes a state of the agent and is labeled with the action performed at that state by the agent. Instead of databases to model the mental state of an agent, moreover, we use sets of Herbrand interpretations as models of the agents' beliefs and goals. A *Herbrand interpretation* is a set of ground atoms [22].

Definition 8.21. *(Trace)*
A *trace* is a mapping from the natural numbers $\mathbb{N}$ (including 0) to triples $\langle B, G, \alpha\rangle$, where B consists of a set of Herbrand interpretations, G is a set of sets of Herbrand

models, and α is an action (including possibly **skip**). A pair (t,i) is called a *time point*. We use t_i^b to denote B, t_i^g to denote G, and t_i^a to denote α in $t(i) = \langle B, G, \alpha \rangle$. We use $\bigcup t_i^g$ to denote the union of all sets in t_i^g (which yields a set of Herbrand interpretations).

The trace semantics introduced above is an approximation of a more general modal semantics, and only includes those elements strictly needed in our setting. The B and G components of a time point correspond respectively to the belief and goal base of an agent, modeled as (sets of) sets of Herbrand models. This setup allows us to use a classical first-order semantics (where models have admittedly been restricted to Herbrand models). Also note that the set of traces does not need to correspond with the set of runs of a particular agent, but - as we will show - we do have that any set of runs generated by an agent may be viewed as a subset of the set of all traces.

Definition 8.22. *(Temporal Language: Semantics)*
The truth conditions of sentences from $\mathcal{L}_G$ are provided relative to a time point (t,i) and are inductively defined by:

$$
\begin{array}{ll}
t,i \models_G \textbf{bel}(\phi) & \text{iff } \forall M \in t_i^b : M \models_c \phi, \\
t,i \models_G \textbf{goal}(\phi) & \text{iff } \exists g \in t_i^g : \forall M \in g : M \models_c \phi, \\
t,i \models_G \textbf{o-goal}(\phi) & \text{iff } \forall M : M \in \bigcup t_i^g \Leftrightarrow M \models_c \phi, \\
t,i \models_G \textbf{done}(\alpha) & \text{iff } t_i^a = \alpha, \\
t,i \models_G \neg\chi & \text{iff } t,i \not\models_G \chi, \\
t,i \models_G \chi \wedge \chi' & \text{iff } t,i \models_G \chi \text{ and } t,i \models_G \chi', \\
t,i \models_G \forall x(\chi) & \text{iff } t,i \models_G \chi[t/x] \text{ for all } t \in \mathcal{T}, \\
t,i \models_G \bigcirc\chi & \text{iff } t,i+1 \models_G \chi, \\
t,i \models_G \chi \textbf{ until } \chi' & \text{iff } \exists j \geq i : t,j \models_G \chi' \text{ and } \forall i \leq k < j : t,k \models_G \chi
\end{array}
$$

We write $t \models \chi$ for $t,0 \models \chi$.

8.4.2 Logical Characterization of Agent Programs

An issue in the verification of agents is whether the behavior of an agent program can be *fully* characterized in a verification logic. A verification logic should not only enable proving properties of *some* of the possible runs of an agent but should also enable to conclude that certain properties hold on *all* possible runs of the agent. This is important because a verification logic that does not allow to fully characterize the behavior of an agent cannot be used to *guarantee* that certain properties are never violated, for example. We explore this issue in more detail in this section.

In order to show that a Goal agent program can be fully characterized logically we transform a program into a set of corresponding axioms. To prove formally that this set of axioms fully characterizes the Goal agent we need to show that the traces

that are models of these axioms correspond with the runs of the GOAL agent. A basic GOAL agent program $\langle K, \Sigma, \Gamma, \Pi, A \rangle$ as discussed above consists of the knowledge contained in K, the initial beliefs Σ and goals Γ, a set of action rules Π, and action specifications A. Each of these components will be transformed in a set of corresponding axioms of the language $\mathcal{L}_G$. It turns out that providing axioms that fully characterize the agent is possible by using the **o-goal** operator introduced in section 8.3.3 and imposing a restriction on the goal base of an agent.

Providing the appropriate axioms that fully characterize the knowledge and beliefs in our setting is relatively straightforward. The knowledge base represents what the agent knows about derived predicates and captures the agent's conceptual and domain knowledge. Since knowledge does not change, intuitively, we must have that $\Box\mathbf{bel}(K)$ is true on all runs of the agent program. This does not yet fully characterize the agent's knowledge with respect to derived predicates, however, as it does not exclude that the agent believes more than $\mathbf{bel}(K)$. A full characterization of the agent's knowledge can be provided using the completion $comp(K)$ as defined in Definition 8.7. The knowledge of the agent about derived predicates is characterized by the following axiom.

$$\Box\mathbf{bel}(comp(K)) \tag{8.1}$$

Similarly, we have that $\mathbf{bel}(\Sigma)$ is true in the initial (mental) state as the belief base Σ contains the initial beliefs of the agent. Again this is not sufficient to fully characterize the agent's beliefs about basic predicates and to exclude that the agent believes basic facts that are not included in the initial state. We have assumed that the base language $\mathcal{L}_0$ contains a finite number of basic predicates and the belief base is finite as well. In addition, a closed world assumption was made. It is therefore possible to finitely characterize an agent's beliefs about basic facts by axioms of the following form, where we need one axiom for each basic predicate $b \in \mathcal{B}$.

$$\forall \mathbf{x}(\mathbf{bel}(b(\mathbf{x})) \leftrightarrow (\mathbf{x} = \mathbf{t}_1 \vee \ldots \vee \mathbf{x} = \mathbf{t}_n)) \tag{8.2}$$

The particular form that the expression $(\mathbf{x} = \mathbf{t}_1 \vee \ldots \vee \mathbf{x} = \mathbf{t}_n)$ can be determined by inspecting all occurrences of the b predicate in the initial belief base of the agent program. The number of disjuncts needed corresponds with the number of ground basic facts of the form $b(\mathbf{t})$ in the initial belief base. If the predicate b does not occur in the belief base, then instead of the axiom above the axiom $\forall \mathbf{x}(\neg\mathbf{bel}(b(\mathbf{x}))$ should be used.

A set of axioms to *fully* characterize the initial goal base of the agent cannot be constructed similarly to those for the initial beliefs. Although it is clearly true that $\mathbf{goal}(\phi_1) \wedge \ldots \wedge \mathbf{goal}(\phi_n)$ holds for any agent with an initial goal base that consists of the goals $\phi_1, \ldots, \phi_n$, it is not so easy to provide an axiom that excludes the possibility that the agent may have any other goals. Although the axiom above may be sufficient for proving that the behavior of the agent will satisfy some specific properties, it is not sufficient to exclude behavior that leads to the violation of certain desired properties. In particular, it is not sufficient to prove that an agent does *not* have

certain goals. Conditions of the form $\neg\mathbf{goal}(\phi)$, which are often used in practice in Goal programs, cannot be derived from a specification $\mathbf{goal}(\phi_1) \wedge \ldots \wedge \mathbf{goal}(\phi_n)$ of goals that are pursued by the agent. We will postpone the discussion of this issue and propose a solution below.

Action rules of the form **if** ψ **then** $a(\mathbf{t})$ provide an agent with the capability to select actions. An action $a(\mathbf{t})$ can only be performed when its precondition $pre(a(\mathbf{t}))$ holds and when the mental state condition ψ of one of the action rules for $a(\mathbf{x})$ holds. We introduce the notion of *enabledness* to express that an action can be performed.

Definition 8.23. *(Enabled)*
Suppose that **if** ψ_1 **then** $a(\mathbf{t}_1)$, ..., **if** ψ_n **then** $a(\mathbf{t}_n)$ are all action rules for a user-specified action $a(\mathbf{x})$ in program $\langle K, \Sigma, \Gamma, \Pi, A\rangle$ and the variables in $\mathbf{x}$ do not occur in any of the conditions ψ_i. The definition of action $a(\mathbf{x})$ being *enabled*, written $enabled(a(\mathbf{x}))$, is the following:

$$enabled(a(\mathbf{x})) \stackrel{df}{=} pre(a(\mathbf{x})) \wedge ((\mathbf{x} = \mathbf{t}_1 \wedge \psi_1) \vee \ldots \vee (\mathbf{x} = \mathbf{t}_n \wedge \psi_n))$$

For the built-in actions **insert**, **adopt** and **drop**, the following axioms are provided, where we suppose again that **if** ψ_1 **then** α, ..., **if** ψ_n **then** α are all action rules for the built-in action α:

$$\begin{aligned} enabled(\mathbf{insert}(\phi)) &\stackrel{df}{=} \psi_1 \vee \ldots \vee \psi_n \\ enabled(\mathbf{adopt}(\phi)) &\stackrel{df}{=} \neg\mathbf{bel}(\phi) \wedge (\psi_1 \vee \ldots \vee \psi_n) \\ enabled(\mathbf{drop}(\phi)) &\stackrel{df}{=} \psi_1 \vee \ldots \vee \psi_n \end{aligned}$$

These axioms express that the action **insert** and **drop** can always be performed when the mental state conditions of the action rules in which they occur hold. For the action **adopt**, additionally, the formula ϕ to be adopted as a goal may not be believed to be the case.

We can use the notion of enabledness to introduce an axiom that characterizes action selection in Goal. The following axiom represents that $a(\mathbf{x})$ can only be performed if one of the mental state conditions ψ_i holds (where variables have been appropriately substituted to obtain a ground action):

$$\forall\mathbf{x}\Box(\bigcirc\mathbf{done}(a(\mathbf{x})) \rightarrow enabled(a(\mathbf{x}))) \tag{8.3}$$

This axiom is sufficient to express that an action is only performed when the right conditions are true. The semantics of actions in Definition 8.17 guarantees that a single action may be executed at any time (it is an interleaving model of action execution). This property is also built into the trace semantics for linear temporal logic above and we do not need an axiom to ensure this. In other words, the following axiom, which excludes that any two actions happen simultaneously, is valid on traces (by Definition 8.21).

$$\forall \mathbf{x} \Box((\mathbf{done}(a(\mathbf{t})) \wedge \mathbf{done}(a'(\mathbf{t}'))) \rightarrow a(\mathbf{t}) = a'(\mathbf{t}'))$$

We need to specify that **skip** is performed in case no other action can be performed. Using the *enabled* predicate introduced above, we obtain the following axiom:

$$\forall \mathbf{x}_1, \ldots, \mathbf{x}_n \Box(\bigcirc \mathbf{done}(\mathbf{skip}) \leftrightarrow (\neg \mathit{enabled}(a_1(\mathbf{x}_1)) \wedge \ldots \wedge \neg \mathit{enabled}(a_n(\mathbf{x}_n)))) \quad (8.4)$$

Finally, we need to state that an agent always performs some action, or otherwise performs the **skip** action.

$$\forall \mathbf{x}_1, \ldots, \mathbf{x}_n \Box \bigcirc (\mathbf{done}(a_1(\mathbf{x}_1)) \vee \ldots \vee \mathbf{done}(a_n(\mathbf{x}_n)) \vee \mathbf{done}(\mathbf{skip})) \quad (8.5)$$

The last component of a GOAL agent program that we need to translate consists of action specifications of the form $\{\phi\}\ a(\mathbf{t})\ \{\phi_1 \Rightarrow \phi'_1 \wedge \ldots \wedge \phi_n \Rightarrow \phi'_n\}$. Note that the action preconditions have already been captured in the axioms above that translate the action rules of a program. What remains is to represent the *action effects*. The effects on the beliefs of an agent are represented here by a temporal logic encoding of Reiter's successor state axioms [305, 364]. The basic idea of Reiter's solution to the frame problem is that a propositional atom $p(\mathbf{t})$ may change its truth value only if an action is performed that affects this truth value. Two cases are distinguished: (i) actions that have $p(\mathbf{t})$ as effect and (ii) actions that have $\neg p(\mathbf{t})$ as effect. For each atom $p(\mathbf{x})$ the first set of actions $a_1(\mathbf{t}_1), \ldots, a_m(\mathbf{t}_m)$ is collected and a disjunction is formed of the form $(\bigcirc \mathbf{done}(a_1(\mathbf{t}_1)1) \wedge \phi_1) \vee \ldots \vee (\bigcirc \mathbf{done}(a_m(\mathbf{t}_m)) \wedge \phi_m)$ denoted by A_p^+, where the ϕ_i are the corresponding conditions of the conditional effects that need to hold to establish $p(\mathbf{t})$, and, similarly, the second set of actions $a_{m+1}(\mathbf{t}_{m+1}), \ldots, a_n(\mathbf{t}_n)$ is collected and a disjunction is formed of the form $(\bigcirc \mathbf{done}(a_{m+1}(\mathbf{t}_{m+1})) \wedge \phi_{m+1}) \vee \ldots \vee (\bigcirc \mathbf{done}(a_n(\mathbf{t}_n)) \wedge \phi_n)$ denoted by A_p^-, where ϕ_i denote the conditions associated with the effects.[13]

Then, for each proposition $p(\mathbf{x})$ a successor state axiom of the form

$$\forall \mathbf{x} \Box(\bigcirc \mathbf{bel}(p(\mathbf{x})) \leftrightarrow (A_p^+ \vee (\mathbf{bel}(p(\mathbf{x})) \wedge \neg A_p^-))) \quad (8.6)$$

is introduced. Intuitively, such formulas express that at any time, in the next state $\mathbf{bel}(p(\mathbf{x}))$ holds iff an action is performed that has $p(\mathbf{x})$ as effect (i.e. A_p^+ holds) or $p(\mathbf{x})$ is true in the current state and no action that has $\neg p(\mathbf{x})$ as effect is performed (i.e. $\neg A_p^-$ holds). Note that this axiom is consistent if the postconditions are consistent in the sense of Definition 8.15.

The more difficult part is again to represent the effects of an action on an agent's goals. Informally, whenever an agent comes to believe that one of its goals has been *completely* achieved, it will drop this goal. The difficult part here is to represent the fact that a goal has been *completely* achieved and not just part of it. For this reason an axiom such as $(\mathbf{goal}(\phi) \wedge \bigcirc \mathbf{bel}(\phi)) \rightarrow \bigcirc \neg \mathbf{goal}(\phi)$ is not valid; it may be that

[13] Note that we assume that $\bigcirc$ binds stronger than $\wedge$ and $\bigcirc(\phi) \wedge \psi$ is equivalent to $(\bigcirc(\phi)) \wedge \psi$.

$\phi = \phi_1$ is part of a larger goal $\phi_1 \wedge \phi_2$ and ϕ_2 has not been achieved yet. In that case it would be irrational to drop the goal. Note that even when ϕ in the axiom would be restricted to conjunctions of basic facts the axiom would still not be valid. The axiom $(\mathbf{goal}(\phi) \wedge \bigcirc\neg\mathbf{bel}(\phi)) \rightarrow \bigcirc\mathbf{goal}(\phi)$ is valid, but does not capture the removal of a goal when it has been achieved.[14]

We have encountered two problems with the characterization of goals. The first concerns the characterization of the *initial goals* of the agent. The second concerns the *dynamics* of goals. One solution to resolve these issues is to use the **o-goal** operator to characterize goals. The **o-goal** operator can be used to fully characterize the goal base of an agent. This comes at a cost: the class of agents needs to be restricted to those agents that only have a single goal. As less restrictive solutions are not apparent in this setting, however, we use the **o-goal** operator.[15] From now on, it is therefore assumed that agents have a single goal to start with and that the adopt action has an additional precondition that ensures that an agent never adopts a second goal if it already has one.

Using the **o-goal** operator and the assumption that the initial goal base consists of a single goal ϕ, the initial goal base of an agent can be characterized by the following axiom.

$$\mathbf{o\text{-}goal}(\phi) \tag{8.7}$$

Using the **o-goal** operator it is possible to represent the dynamics of goals and to provide an axiom that captures the removal of goals correctly. Here we use the fact that $\neg\mathbf{goal}(\top)$ can only be true if the goal base is empty. Several axioms to characterize goal dynamics are introduced which each deal with one out of a number of cases.

$$\forall \mathbf{x}\Box(\mathbf{o\text{-}goal}(\phi) \wedge \bigcirc\mathbf{bel}(\phi)) \rightarrow \bigcirc\neg\mathbf{goal}(\top) \tag{8.8}$$

This axiom covers the cases where either a user-specified action or the built-in action **insert** is performed. Notice that ϕ cannot have been believed by the agent in the state in which the action was performed as we must have $\mathbf{o\text{-}goal}(\phi)$ for this axiom to apply. Since the beliefs of an agent are not changed by an **adopt** or **drop** action, we therefore can be certain neither of these actions has been performed.

Goals persist when an agent does not believe the goal to be achieved, *and* the goal has not been explicitly dropped by a **drop** action. We thus have:

$$\forall \mathbf{x}\Box(\mathbf{o\text{-}goal}(\phi) \wedge \bigcirc(\neg\mathbf{bel}(\phi) \wedge \mathit{done}(\alpha)) \rightarrow \bigcirc\mathbf{o\text{-}goal}(\phi) \tag{8.9}$$

[14] It is easy to show that (i) $(\mathbf{goal}(\phi) \wedge \bigcirc\mathbf{bel}(\phi)) \rightarrow \bigcirc\neg\mathbf{goal}(\phi)$ is inconsistent with (ii) $(\mathbf{goal}(\phi) \wedge \bigcirc\neg\mathbf{bel}(\phi)) \rightarrow \bigcirc\mathbf{goal}(\phi)$. Suppose $\mathbf{goal}(p \wedge q)$, $\bigcirc\mathbf{bel}(p)$ and $\bigcirc\neg\mathbf{bel}(q)$ are true at a time point (t,i). It follows that we have $\bigcirc\neg\mathbf{bel}(p \wedge q)$, and, as a consequence of (ii), we then must have that $\bigcirc\mathbf{goal}(p \wedge q)$ is true. Note that this implies that both $\bigcirc\mathbf{goal}(p)$ and $\bigcirc\mathbf{goal}(q)$ are true. Using (i) and $\bigcirc\mathbf{bel}(p)$ we then should also have $\bigcirc\neg\mathbf{goal}(p)$, which yields a contradiction.

[15] One way to go is to allow temporal formulas in the goal base and define the semantics of goals in terms of time points (see e.g. [227]). It is outside the scope of this chapter to discuss this solution, and additional research is needed to address this issue. A similar approach is used in [396].

where α is either a user-specified action or an **insert** or **adopt** action.[16]

For the case where a **drop** action is performed, two additional cases need to be distinguished: (i) the case where the goal of the agent is dropped, and (ii) the case where the goal is not dropped. A goal ϕ is dropped by performing **drop**(ϕ') only if $\phi \models_c \phi'$. As we cannot express in our verification logic that ϕ' is a logical consequence of ϕ, we introduce two inference rules to represent cases (i) and (ii).

$$\frac{\phi \models_c \phi'}{\Box(\textbf{o-goal}(\phi) \wedge \bigcirc \mathit{done}(\textbf{drop}(\phi'))) \rightarrow \bigcirc \neg \textbf{goal}(\top)} \tag{8.10}$$

$$\frac{\phi \not\models_c \phi'}{\Box(\textbf{o-goal}(\phi) \wedge \bigcirc \mathit{done}(\textbf{drop}(\phi'))) \rightarrow \bigcirc \textbf{o-goal}(\phi)} \tag{8.11}$$

Finally, we need to treat the case where the agent did not have any goals. That is, we need axioms that characterize the persistence of the absence of goals. Only performing an **adopt** action can add a goal, and the following axioms characterize respectively goal adoption and the persistence of the absence of any goals.

$$\forall \mathbf{x} \Box(\neg \textbf{goal}(\top) \wedge \bigcirc \mathit{done}(\textbf{adopt}(\phi))) \rightarrow \bigcirc \textbf{o-goal}(\phi) \tag{8.12}$$

$$\forall \mathbf{x} \Box(\neg \textbf{goal}(\top) \wedge \bigcirc \mathit{done}(\alpha)) \rightarrow \bigcirc \neg \textbf{goal}(\top) \tag{8.13}$$

where α is either a user-specified action or one of the built-in actions **insert** or **drop**.

Definition 8.24. *(Logical Representation of an Agent)*
The *logical representation* of a GOAL agent $Agt = \langle K, \Sigma, \Gamma, \Pi, A \rangle$, denoted by $Rep(Agt)$, is the set of axioms listed in Table 8.3.

In order to make precise what we mean by a run that corresponds to a trace, we introduce the following definitions.

Definition 8.25. *(Run Corresponds with Trace)*
Let $Agt = \langle K, \Sigma, \Gamma, \Pi, A \rangle$ be a GOAL agent. A run r *corresponds* with a trace t, written $r \approx t$, if for each $i \in \mathbb{N}$, with $r_i^m = \langle K, \Sigma, \Gamma \rangle$ and $t_i^b = B$ and $t_i^g = G$:

- $[\![K]\!](\Sigma) = \bigcap B$,
- $\forall \gamma \in \Gamma : \exists g \in t_i^g : \gamma = \bigcap g$, and $\forall g \in t_i^g : \exists \gamma \in \Gamma : \gamma = \bigcap g$, and
- $r_i^a = t_i^a$.

We also write $\mathcal{R} \approx T$ for a set of runs $\mathcal{R}$ and traces T if for all runs there is a trace that corresponds with it, and vice versa.

[16] In line with our assumption that an agent only has a single goal, an **adopt** action cannot be performed when the agent already has a goal.

For any Goal agent $\langle K,\Sigma,\Gamma,\Pi,A\rangle$ with a goal base Γ with exactly one goal, the following set of axioms is a logical representation of this agent, in combination with inference rules (10) and (11). (See also Theorem 8.2.)

knowledge base	$\Box\mathbf{bel}(comp(K))$
belief base	$\forall\mathbf{x}(\mathbf{bel}(b(\mathbf{x})) \leftrightarrow (\mathbf{x}=\mathbf{t}_1 \vee \ldots \vee \mathbf{x}=\mathbf{t}_n))$
goal base	$\mathbf{o\text{-}goal}(\phi)$
action rules	$\forall\mathbf{x}\Box(\bigcirc\mathbf{done}(a(\mathbf{x})) \rightarrow enabled(a(\mathbf{x}))$
	$\forall\mathbf{x}_1,\ldots,\mathbf{x}_n\Box(\bigcirc\mathbf{done}(\mathbf{skip}) \leftrightarrow (\neg enabled(a_1(\mathbf{x}_1)) \wedge \ldots \wedge \neg enabled(a_n(\mathbf{x}_n))))$
	$\forall\mathbf{x}_1,\ldots,\mathbf{x}_n\Box\bigcirc(\mathbf{done}(a_1(\mathbf{x}_1)) \vee \ldots \vee \mathbf{done}(a_n(\mathbf{x}_n)) \vee \mathbf{done}(\mathbf{skip}))$
action specs	$\forall\mathbf{x}\Box(\bigcirc\mathbf{bel}(p(\mathbf{x})) \leftrightarrow (A_p^+ \vee (\mathbf{bel}(p(\mathbf{x})) \wedge \neg A_p^-)))$
goal dynamics	$\forall\mathbf{x}\Box(\mathbf{o\text{-}goal}(\phi) \wedge \bigcirc\mathbf{bel}(\phi)) \rightarrow \bigcirc\neg\mathbf{goal}(\top)$
	$\forall\mathbf{x}\Box(\mathbf{o\text{-}goal}(\phi) \wedge \bigcirc(\neg\mathbf{bel}(\phi) \wedge done(\alpha)) \rightarrow \bigcirc\mathbf{o\text{-}goal}(\phi)$
	where α is either a user-specified action or an **insert** or **adopt** action
	$\forall\mathbf{x}\Box(\neg\mathbf{goal}(\top) \wedge \bigcirc done(\mathbf{adopt}(\phi))) \rightarrow \bigcirc\mathbf{o\text{-}goal}(\phi)$
	$\forall\mathbf{x}\Box(\neg\mathbf{goal}(\top) \wedge \bigcirc done(\alpha)) \rightarrow \bigcirc\neg\mathbf{goal}(\top)$
	where α is either a user-specified action or an **insert** or **drop** action.

Table 8.3 Logical Representation of a Goal Agent

The following theorem states that the set of axioms obtained by means of the "translation" procedure discussed above characterizes a Goal agent completely, in the sense that runs of the agent correspond to traces that are models of the logical representation.

Theorem 8.2. (Logical Representation Characterizes Agent)
Let $Agt = \langle K,\Sigma,\Gamma,\Pi,A\rangle$ be a Goal *agent and $\mathcal{R}$ denote the meaning of this agent. Let $Rep(Agt)$ be the corresponding logical representation of the agent. Then we have:*

$$\mathcal{R} \approx \{t \mid t \models_G Rep(Agt)\}$$

Proof. The proof proceeds in three steps and is based on induction. First, we show that a state representation $r(i)$ in a run corresponds with a time point (t,i) in a trace. Second, we show that if there is a run in which action α is executed at i then there is also a corresponding trace that executes α at i. Third, we show that the resulting state representation at the next point $r(i+1)$ corresponds with the time point $(t,i+1)$.

For the base case, we need to show that the initial mental state $r_0^m = \langle K,\Sigma_0,\Gamma_0\rangle$ corresponds with time points $(t,0) = \langle B_0,G_0,\alpha_0\rangle$ for arbitrary traces t that are models of $Rep(Agt)$. We need to show that:

- $[\![K]\!](\Sigma_0) = \bigcap B_0$. *Use axiom (2) above to show that basic facts match, and use axiom (1) and the proof of Theorem 8.1 to demonstrate that derived facts also match.*

- $\Gamma = \bigcap G_0$.[17] *For this case, use axiom (7).*

Below, we assume $r_i^m \approx (t,i)$ *for all* $0 \leq i \leq n$ *as induction hypothesis (IH).*

Next we show that whenever α *is executed in r at point i, then it is also executed at time point* (t,i) *for some trace t that corresponds for all time points with* $i \leq n$*, and vice versa. So, suppose that* α *is executed in the run at point i. This means that there is an action rule* ***if*** ψ ***then*** α *such that* $r_i^m \models_\Psi \psi \wedge \textbf{\textit{bel}}(pre(\alpha))$*. It follows that we also have* $(t,i) \models_G \psi \wedge \textbf{\textit{bel}}(pre(\alpha))$ *by the IH. We have that axiom (3) is also satisfied if* $t_i^a = \alpha$ *and t is a model of* $Rep(Agt)$*. The other direction is similar, using axiom (3) once more in combination with axiom (5). In case* $\alpha =$ ***skip****, we need axiom (4) instead.*

Finally, we need to show that the effects of executing an action produce corresponding states in the run and trace. As knowledge does not change, it is sufficient to show that basic facts are updated correspondingly. It follows using axiom (6) that the beliefs of an agent in a run correspond with that in the trace. For the single goal of the agent, use axioms and inference rules (7-13) to show that updates correspond. □

Concluding, we have established that Goal agents can be said to *execute* particular logical specifications, i.e. their logical representations. These logical representations fully characterize a Goal agent. As a result, any properties that logically follow from its logical representation are properties of that Goal agent. A logical representation can be used and provided as input to a model checker to verify a property χ or to a theorem prover to deduce χ.

8.5 Conclusion

We have discussed a logic for reasoning about agents with *declarative* goals. Our aim has been to provide a logic that facilitates the verification of computational agents that derive their choice of action from their beliefs and goals. To this end we have presented a verification logic for Goal agents based on linear temporal logic. The agent programming language Goal enables programming computational agents that have a declarative beliefs and goals.

It has turned out to be a difficult task to fully characterize a Goal agent by means of logic. The main reason is that the logic of goals is hard to capture formally. Although on the one hand goals seem to have some obvious logical properties, on the other hand a goal seems to be not much more than a resource that is being used by an agent. The main issue here is to express that an agent does *not* have certain goals, and how to provide a frame axiom for goals. It may be possible to do so by characterizing goals by temporal formula (see e.g. [227]). The presentation of our

[17] Recall we have assumed there is a single goal only, which allows us to simplify somewhat here.

results is intended to further stimulate research in this area which is so important for agents with motivational attitudes.

Even though we were able to fully characterize GOAL agents logically, provided that such agents only have a single goal, we did not prove completeness of the verification logic $\mathcal{L}_G$. Providing a complete axiomatization remains for future work. A complete axiomatization is important since without it we are still not sure that we can prove all properties of an agent by means of deduction.

One of the aims of this chapter has been to show that additional research is needed in order to provide the tools to reason about goals in agent programming. This is still true for agents that only have achievement goals, the type of goals that we discussed here. But this is even more true when other types of goals are concerned, such as maintenance goals [129, 152, 227].

Finally, although there is work that deals with verifying multi-agent systems, as far as we know there is no work that combines multi-agent with cognitive agents. A considerable challenge thus remains to provide a logic to verify computational multi-agent systems, and extend work on single-agent logics to multi-agent logics.

Acknowledgements The work presented would not have been possible without many useful discussions and comments from, in particular, Frank de Boer, Wiebe van der Hoek, and John-Jules Ch. Meyer. I'd like to thank them here for the interesting exchanges on the topic of verification.

Appendix

This Appendix contains proofs for some additional Propositions and Theorem 8.1 in the main text. We first provide an alternative semantics for stratified axiom sets. This semantics is more convenient as it allows for the use of induction in the proofs below.

Alternative Fix Point Semantics

It will be convenient to write $F \models \phi$ for a set of facts $F = S \cup D$ instead of $\langle S, D\rangle \models \phi$, where S is a set of basic facts and D a set of derived facts; $F \models \phi$ can be viewed simply as a notational convenience, as shorthand for $\langle S, D\rangle \models \phi$.

Definition 8.26. Let A be a stratified axiom set, and $\{A_i, 1 \leq i \leq n\}$ be a partition that stratifies A. Let S be a set of basic facts and D a set of derived facts. Then for $0 \leq i \leq n$ define:

$$\begin{array}{ll} T_i(S,D) & = \{d_i(\mathbf{t}) \mid \forall \mathbf{x}(\phi \rightarrow d(\mathbf{x})) \in A_i, \langle S, D\rangle \models \phi[\mathbf{x} \leftarrow \mathbf{t}], \mathbf{t} \text{ are ground } \} \cup D \\ T_0^{\infty}(S) & = \emptyset \\ T_i^0(S) & = T_{i-1}^{\infty}(S) \text{ for } i > 0 \\ T_i^j(S) & = T_i(S, T_i^{j-1}(S)) \text{ for } i, j > 0 \end{array}$$

where $T_i^{\infty}(S) = \bigcup_{j \in \mathbb{N}} T_i^j(S)$ for $i > 0$.

Proposition 8.2. *Let S be a set of basic facts. For $1 \leq i \leq n$, we have:*

$$T_i^{\infty}(S) = [\![A]\!]_i(S)$$

Proof. By induction on i. For the base case $i = 1$, we need to show that $T_1^{\infty}(S) = [\![A]\!]_1(S)$. We first show that:

$$\bigcup_{a \in A_1} [\![a]\!](S, T_1^{\infty}(S)) \subseteq T_1^{\infty}(S)$$

Or, equivalenty, that $\{d(\mathbf{t}) \mid \langle S, T_1^{\infty}(S)\rangle \models \phi[\mathbf{t}], \mathbf{t}$ is ground$\} \subseteq T_1^{\infty}(S)$ for all axioms $\forall \mathbf{x}(\phi \rightarrow d(\mathbf{t})) \in A_1$. If $\langle S, T_1^{\infty}(S)\rangle \models \phi[\mathbf{t}]$, we must have that $\langle S, T_1^j(S)\rangle \models \phi[\mathbf{t}]$ for some $j \in \mathbb{N}$. But then $d(\mathbf{t}) \in T_1(S, T_1^j(S)) = T_1^{j+1}(S) \subseteq T_1^{\infty}(S)$, and we are done.

What remains is to show that for any D such that $\bigcup_{a \in A_1} [\![a]\!](S, D) \subseteq D$, we have $T_1^{\infty}(S) \subseteq D$. Clearly, we have $T_1^0(S) = \emptyset \subseteq D$. So, to arrive at a contraduction, suppose that $T_1^{\infty}(S) \not\subseteq D$. Then there must be a largest $j \in \mathbb{N}$ such that $T_1^j(S) \subseteq D$ but $T_1^{j+1}(S) \not\subseteq D$. But this means that $T_i(S, T_1^j(S))$ is non-empty, which implies that for

some axiom a we have that $[\![a]\!](S,D)$ is non-empty and $[\![a]\!](S,D) \subseteq T_i(S, T_1^j(S))$. A contradiction.

The case for $i > 1$ is similar. □

Proofs

Lemma 8.1. *Let A be a stratified axiom set and M be a model for $una \cup dca \cup S \cup comp(A)$. Then:*

1. *M is a Herbrand model (up to isomorphism).*
2. *M satisfies $S \cup [\![A]\!](S)$.*

Proof.

1. *As $M = \langle D, I\rangle$ satisfies $una \wedge dca$, we must have that D is isomorph with $\mathcal{T}$.*
2. *By induction on the number of partitions k of A that stratify A. The base case $k = 0$ is trivial as in that case $[\![A]\!](S) = \emptyset$. For the inductive case, suppose we have $M \models S \cup [\![A]\!]_k(S)$. We need to show that $M \models [\![A]\!]_{k+1}(S)$, or, equivalently (by Proposition 8.26) $M \models T_{k+1}^{\infty}(S)$. We show $M \models T_{k+1}^{j}(S)$ for all $j \in \mathbb{N}$ by induction on j. The base case, $j = 0$, is trivial as in that case $T_{k+1}^{0}(S) = T_k^{\infty}(S)$. For $j > 0$, we assume that $M \models T_{k+1}^{j-1}(S)$ as induction hypothesis. By Definition 8.26 of T_{k+1} it follows immediately that $M \models T_{k+1}^{j}(S) = T_{k+1}(S, T_{k+1}^{j-1}(S))$, and we are done.*

□

Lemma 8.2. *$S \cup [\![A]\!](S)$ is a Herbrand model for $una \cup dca \cup S \cup comp(A)$.*

Proof. Clearly, $S \cup [\![A]\!](S)$ is a Herbrand model for $una \cup dca \cup S$. □

Theorem 8.3. *Let $S \subseteq F$ be a set of basic facts and A be a stratified axiom set. Let ϕ be a closed formula. Then we have that if $una \cup dca \cup S \cup comp(A) \cup \phi$ is satisfiable, then $S \cup [\![A]\!](S)$ is a Herbrand model for $una \cup dca \cup S \cup comp(A) \cup \phi$.*

Proof. The proof proceeds by induction on the maximum level k of the stratification of A.

For the base case $k = 0$ we have that A must be a definite

□

Lemma 8.3. *Let S be a set of ground atoms and A be a stratified axiom set. Then we have that $S \cup [\![A]\!](S)$ is a (minimal) Herbrand model of $comp(A)$.*

Proof. Note that $S \cup [\![A]\!](S)$ *can be interpreted as a Herbrand model H. Simply define the domain as the set of closed terms, and define an interpretation that on the set of terms is the identity function and maps ground atoms to true if and only if they are an element of* $S \cup [\![A]\!](S)$*. See [289], p. 111.* □

Finally, we can provide the proof for Theorem 8.1. Let S be a set of ground atoms, A be a stratified axiom set, and θ be an answer for ϕ with respect to S and A. Then θ is a correct answer.

Proof. We have to show that:

$$S \models_A \forall(\phi\theta) \text{ iff } S \cup una \cup dca \cup comp(A) \models \forall(\phi\theta)$$

Let B denote the Herbrand base for $\mathcal{L}_0$. We use $\neg F$ to denote the set of all negated formulas from a set of formulas F, i.e. $\{\neg\phi \mid \phi \in F\}$. Let $T = S \cup [\![A]\!](S)$. For the left to right direction, it is sufficient to show that $S \cup una \cup dca \cup comp(A)$ entails $T \cup \neg(B \setminus T)$.

For the right to left direction, as above, we use $S \cup [\![A]\!](S)$ to construct a corresponding Herbrand model H and show that H is a model of $una \cup dca \cup comp(A) \cup S$. By definition, we have $H \models una$ and $H \models dca$, as the domain of H is the set of terms $\mathcal{T}$. It is also obvious that $H \models S$. Finally, using Lemma 8.3 it follows that H is also a model of $comp(A)$.

□