Making Model Checking Feasible for GOAL

Yi Yang[0000−0001−9565−1559] and Tom Holvoet[0000−0003−1304−3467]

KU Leuven, Dept. of Computer Science, imec-DistriNet, B-3001 Leuven, Belgium
{yi.yang,tom.holvoet}@kuleuven.be

Abstract. Agent Programming Languages have been studied for over 20 years for programming complex decision-making for autonomous systems. The GOAL agent programming language is particularly interesting as it does not require any preprogrammed planning by developers, but instead relies on automated planning based on beliefs and goals to determine its behavior.
Model checking is a powerful verification technique to guarantee the safety of an autonomous system. Despite studies of model checking in other agent programming languages, GOAL lacks support for model checking of GOAL programs. The fundamental challenge is to make GOAL programs feasible for model checking in the first place.
In this paper, we tackle this fundamental issue. We devise an algorithm for transforming a (considerable) subset of GOAL programs to a transition system that is equivalent in terms of operational semantics, enabling model checking. We prove the correctness of this algorithm. We implement the transformation algorithm, and we discuss the scalability through Blocks World examples of increasing size. Moreover, we point out that we will extend the applicability of the transformation algorithm and its implementation to all stratified GOAL programs.

Keywords: GOAL agent programming language · Model checking · Decision-Making

1 Introduction

Autonomous systems, like robots in an Industry 4.0 setting [18], self-driving vehicles, and even autonomous software systems are exciting and promising areas, for which however quite some challenges remain. Making correct decisions in a complex world is obviously one of these challenges, and even more, being able to assure safe behavior is another. Providing solid evidence that safety in decision making is ensured will be a key element to have the public accept autonomous systems in their environments [7], [14].

In artificial intelligence, the concept of an agent refers to an entity that functions continuously and autonomously in an environment in which other processes take place and other agents exist [17]. Agent-oriented programming (AOP) provides programming languages for properly specifying autonomous decision-making. However, most agent programming languages rely on plan specifications.

GOAL distinguishes itself by automated plan generation, and as such is a prime candidate vehicle for rendering features of autonomous behavior [8].

The verification of agent programming languages is challenging, and there is no widely accepted solution at the current stage. Model checking is the most studied verification technique in agent programming languages [6] [3] [19]. The Model Checking Agent Programming Language (MCAPL) framework is a general verification framework for different agent programming languages. However, every agent programming language has to be implemented as a sub-class of the Agent Infrastructure Layer (AIL) agent class [6]. The faithfulness of the implementation is a crucial issue. GOAL agents were implemented as sub-classes of the AIL agent class [5]. However, the faithfulness of the implementation remains to be proved. Furthermore, the efficiency issue is a bottleneck of MCAPL. Unlike for other agent programming languages, there is little work to verify GOAL programs by model checking. The first verification framework of GOAL was proposed in [2]. This paper shows a direction on how to apply theorem proving on GOAL. This paper provides a complete theory of agent programming, thereby verifying the correctness of GOAL. However, first-order properties of GOAL cannot be expressed in the framework. In [9], the verification idea in [2] was implemented in Isabelle [13]. The verification framework has high confidence due to Isabelle's implementation, yet it provides no automated verification process, and it can not express first-order properties. [10] presented a model checker specifically for Goal, but there is no access to the model checker. Moreover, many efficient symbolic model checkers, such as [4] and [12], have been developed in recent years. Using the existing high-performance model checker is more reasonable than developing a model checker for GOAL from scratch.

In this paper, we explored how to make model checking feasible for GOAL programs. In summary, we make the following contributions.

- We propose an algorithm to transform a stratified GOAL program for a single agent and a single goal to its equivalent transition system in terms of operational semantics, and we justify the correctness of the algorithm.
- We implement the transformation algorithm, and we discuss the scalability with several Blocks World examples.

The paper is structured as follows. Section 2 gives the theoretical and technical background of the work of the paper. Section 3 describes an algorithm for transforming a substantial subset of GOAL programs to a transition system that is equivalent in terms of operational semantics and proves the correctness of this algorithm. Section 4 presents the scalability with several Blocks World examples. Section 5 draws conclusions and indicates future work.

2 Background

This section briefly explains how GOAL generates its decision-making, and it briefly introduces all necessary theoretical backgrounds of our work.

2.1 Decision-Making in GOAL

GOAL is a high-level programming language to program rational agents that derive their choice of actions from their beliefs and goals [8]. GOAL employs reasoning strategies to automatically generate its decisions. For more details of GOAL, we refer to [8] and [2].

We refer to those specifications in a GOAL program in relation to the automated generation of decision-making as logical specifications in this paper. The logical specifications of a GOAL program consist of six modules: a belief base, a goal base, a knowledge base, a set of constraints of action generation, a set of enabledness of actions, and a set of action specifications. GOAL automatically generates its decision-making on the basis of the first-order logic derivation. We explain how GOAL generates its decision-making by a Blocks World example.

The Blocks World example consists of seven blocks and a table. Figure 1 shows the initial state of the Blocks World, and figure 2 presents the goal state of the Blocks World. The task is to find a sequence of actions transforming the Blocks World from the initial state to the goal state. For the source code of the GOAL implementation of Blocks world, we refer to the GOAL example project *BlocksWorld* provided in the Eclipse plugin for GOAL.

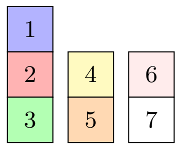

Fig. 1.
Initial State

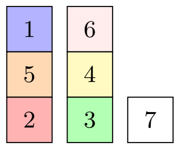

Fig. 2.
Goal State

Belief Base A belief base describes all beliefs that the agent currently needs to know. The belief base of the Blocks World example is encoded in GOAL as follows:

– on(b1,b2). on(b2,b3). on(b3,table). on(b4,b5). on(b5,table). on(b6,b7). on(b7,table).

The logical interpretation of the belief base is a set of atoms. In this example, the logical interpretation of the belief base is as follows:

– $\{on(b1, b2), on(b2, b3), on(b3, table), on(b4, b5), on(b5, table), on(b6, b7), on(b7, table)\}$

Goal Base A goal base describes all beliefs that the agent should achieve. The goal base of the Blocks World example is encoded in GOAL as follows:

– on(b1,b5), on(b2,table), on(b3,table), on(b4,b3), on(b5,b2), on(b6,b4), on(b7,table).

The logical interpretation of the goal base is a set of atoms. In this example, the logical interpretation of the goal base is as follows:

– $\{on(b1, b5), on(b2, table), on(b3, table), on(b4, b3), on(b5, b2), on(b6, b4), on(b7, table)\}$

Knowledge Base A knowledge base represents the collection of domain rules the agent needs to know. The knowledge base of the Blocks World example is encoded in GOAL as follows:

- $block(X) : -on(X, \_).$
- $clear(table)$
- $clear(X) : -block(X), not(on(\_, X)).$
- $tower([X]) : -on(X, table).$
- $tower([X, Y|T]) : -on(X, Y), tower([Y|T]).$

The logical interpretation of the knowledge base is a theory containing both ground atoms and closed formulas. In this example, the logical interpretation of the knowledge base is as follows:

- $\forall x.\exists y.on(x, y) \rightarrow block(x)$
- $clear(0)$
- $\forall x, y.block(x) \wedge \neg on(y, x) \rightarrow clear(x)$
- $\forall x.on(x, 0) \rightarrow tower([x])$
- $\forall x, y, t.on(x, y) \wedge tower([y|t]) \rightarrow tower([x, y|t]).$

In a GOAL program, all current beliefs are derived from its belief base and its knowledge base. Logically speaking, the current beliefs are the minimal model of the theory consisting of the belief base and the knowledge base. Similarly, all desired beliefs are derived from its goal base and its knowledge base. The current beliefs and the desired beliefs are used to evaluate which constraints of action generation will be feasible. In this example, the set of current beliefs and the set of desired beliefs are listed as follows:

- $current\_beliefs = \{on(1, 2), on(2, 3), on(3, 0), on(4, 5), on(5, 0), on(6, 7), on(7, 0), clear(0), block(1), block(2), block(3), block(4), block(5), block(6), block(7), tower([3]), tower([5]), tower([7]), clear(1), clear(4), clear(6), tower([2, 3]), tower([4, 5]), tower([6, 7]), tower([1, 2, 3])\}$
- $desired\_beliefs = \{on(1, 5), on(2, 0), on(3, 0), on(4, 3), on(5, 2), on(6, 4), on(7, 0), clear(0), block(1), block(2), block(3), block(4), block(5), block(6), block(7), tower([2]), tower([3]), tower([7]), clear(1), clear(6), clear(7), tower([4, 3]), tower([5, 2]), tower([1, 5, 2]), tower([6, 4, 3])\}$

Constraints of action generation A constraint of action generation specifies a collection of triggering conditions involving the current beliefs and desired beliefs. The set of constraints of action generation of the Blocks World example is encoded in GOAL as follows:

- $define\, constructiveMove(X, Y)\ as\ a{-}goal(tower([X, Y|T])), bel(tower([Y|T])).$
- $define\ misplaced(X)\ as\ a - goal(tower([X|T])).$

The logical interpretation of the constraints of action generation is a theory. In this example, the logical interpretation of the constraints of action generation is as follows:

- $\forall x, y. \exists t.(current\_beliefs \to \neg tower([x, y|t])) \to flag_1(x, y, t)$
- $\forall x, y. \exists t.(desired\_beliefs \to tower([x, y|t])) \to flag_2(x, y, t)$
- $\forall x, y. \exists t. flag_1(x, y, t) \wedge flag_2(x, y, t) \wedge tower([y|t]) \to constructiveMove(x, y)$
- $\forall x. \exists t.(current\_beliefs \to \neg tower([x|t])) \to flag_3(x, t)$
- $\forall x. \exists t.(desired\_beliefs \to tower([x|t])) \to flag_4(x, t)$
- $\forall x. \exists t. flag_3(x, t) \wedge flag_4(x, t) \to misplaced(x)$.

Enabledness of actions An enabledness of actions specifies which action is triggered under which collection of triggering conditions. The set of enabledness of actions of the Blocks World example is encoded in GOAL as follows:

- if constructiveMove(X,Y) then move(X, Y).
- if misplaced(X) then move(X, table).

Action specifications An action specification describes the preconditions and postconditions of an action. The action specifications of the Blocks World example is encoded in GOAL as follows:

- define move(X,Y) with
  pre clear(X), clear(Y), on(X,Z), not(on(X,Y))
  post not(on(X,Z)), on(X,Y)

An action can be triggered only when the constraints of action generation, its enabledness of actions, and its preconditions are satisfied. The logical interpretation of the generation of decision-making is a theory. In this example, the logical interpretation of the generation of decision-making is as follows:

- $\forall x, y. \exists z. constructiveMove(x, y) \wedge clear(x) \wedge clear(y) \wedge on(x, z) \wedge \neg on(x, y) \to move(x, y)$
- $\forall x. \exists z. misplaced(x) \wedge clear(x) \wedge clear(0) \wedge on(x, z) \wedge \neg on(x, 0) \to move(x, 0)$.

The set of current beliefs, the set of desired beliefs, the logical formulas of the constraints of action generation, the logical formulas of the enabledness of actions compose a theory. GOAL derives its decision-making from the theory. In this example, the first possible decision-making can be $move(1, 0)$, $move(4, 0)$, and $move(6, 0)$.

The agent transforms its state on the basis of the current belief base, the postcondition of the action, and the feasible action. The logical interpretation of the state transformer is a theory. In this example, the logical interpretation of the state transformer is as follows:

- $\forall x, y, z. move(x, y) \wedge on(x, z) \wedge \neg on(x, y) \to \neg on(x, z) \wedge on(x, y)$.

The agent generates decision-making by automated logical derivation until it reaches the desired state, or it tries all possibilities but cannot reach the desired state.

2.2 Theoretical Foundation

All lemmas, theorems, and definitions listed in this subsection will be used in the following sections.

Lemma 1. *(Universal Quantifier Elimination in Closed First-Order Formulas with a Finite Domain)*

$$\forall x.F(x) \rightarrow g \equiv \bigwedge_{i \in D} F(i) \rightarrow g,$$
$$\forall x.F(x) \rightarrow h(x) \equiv \bigwedge_{i \in D} (F(i) \rightarrow h(i)),$$

where $F(x)$ is a conjunction of atoms, g is an atom without x, h is an atom containing x, D is the finite domain of x.

Lemma 1 presents how to eliminate universal quantified variables in closed first-order formulas. The first logical equivalence shows how to eliminate the universal variable only occurring at the left side of the implication. The second logical equivalence presents how to eliminate the universal variable occurring at both sides of the implication. We distinguish universal variables in the implementation presented in Section 4.

Theorem 1. *(Minimal Model for a Stratified Logic Program) [16]*
If a normal logic program is stratified then it has a minimal model.

Theorem 1 justifies that a stratified logic program has a minimal model. We use this theorem in Section 3.2.

Theorem 2. *(Soundness and Completeness of first-order logic)*
Let Σ be a first-order theory, and ϕ be a well-defined first-order formula:

$$\Sigma \vdash_M \varphi \leftrightarrow \Sigma \models \varphi$$

For a first-order theory, if φ can be derived syntactically, φ can be derived semantically, vice versa. We use this theorem in Section 3.2.

Definition 1. *(Conditions of Stratified Logic Program) [15]*
The logic program P is stratified iff the dependency graph for P contains no cycles containing a negative edge.

Definition 1 defines a stratified logic program. We use this definition in Section 3.

Definition 2. *(Transition System) [1]*
A transition system TS is a tuple $(S, Act, \rightarrow, I, F, AP, L)$ where

- *S is a set of states,*
- *Act is a set of actions,*
- *$\rightarrow \subseteq S \times Act \times S$ is a transition relation,*

- *$I \subseteq S$ is a set of initial states,*
- *$F \subseteq S$ is a set of final states,*
- *AP is a set of atomic propositions, and*
- *$L : S \to 2^{AP}$ is a labeling function.*

Definition 3. *(Execution Fragment) [1]*
Let $(S, Act, \to, I, AP, L)$ be a transition system TS. A finite execution fragment ϱ of TS is an alternating sequence of states and actions ending with a state

$$\varrho = s_0\alpha_1 s_1\alpha_2...s_n\alpha_n \text{ such that } s_i \xrightarrow{\alpha_{i+1}} s_{i+1} \text{for all } 0 \leq i < n$$

where $n \geq 0$.

A transition system can be used as the input in the model checking. In this paper, we want to build a connection from a GOAL program to its equivalent transition system in terms of operational semantics. In this paper, we define a transition system following Definition 2, and we define an execution fragment following Definition 3.

Definition 4. *The first-order theory underlying any GOAL programs consists of only ground atoms and closed formulas.*

$$ground\_atom = A_i(a_1, ..., a_n)$$
$$r = \forall x \in X.\exists y \in Y. \bigwedge_i F_i \wedge \bigwedge_j \neg A_j \to C$$
$$theory\_of\_Prolog = \{M_1, ..., M_n\}$$
$$a\text{-}goal\ F = (model_C \to \neg F) \wedge (model_G \to F)$$
$$c = \forall x \in X.\exists y \in Y.a\text{-}goal\ F \wedge \bigwedge_i F_i \wedge \bigwedge_j \neg A_j \to R$$
$$e = \forall x \in X.\exists y \in Y.R \wedge \bigwedge_i F_i \wedge \bigwedge_j \neg A_j \to Act$$
$$Theory_{GOAL\_State} = \{model_C, model_G, H_1, ..., H_n\}$$
$$post\_Act = \forall x \in X.Act \wedge \bigwedge_i B_i \wedge \bigwedge_j \neg C_j \to \bigwedge_m D_m \wedge \bigwedge_n \neg E_n$$

Here is a brief explanation of the above first-order theory. Syntactically, A_i denotes a predicate; a_i denotes a constant; F_i, A_j, C, F, R, Act, B_i, C_j, D_m, and E_n are atoms; c, e, r, M_i, H_i, and $post\_Act$ are first-order logical formulas; $model_C$ and $model_G$ denote a set of ground atoms. Semantically, r denotes a rule in the knowledge base; M_i is an item of the belief base, or an item of the goal base, or an item of the knowledge base; c denotes a constraint of action generation; and e denotes an enabledness of actions; $Theory_{GOAL\_State}$ denotes the first-order theory underlying the current state of the GOAL agent; $model_C$ represents all beliefs held in the current state; $model_G$ denotes all beliefs held in the goal state, H_i is either a constraint of action generation (c) or an enabledness of actions (e). We use this definition in Section 3.2.

Definition 5. *(State of a GOAL agent)*
A state of an GOAL agent with single goal is a pair $< \Sigma,\ \Gamma >$ *where* Σ *is the agent's current belief base and* Γ *is the agent's goal base.*

Given a knowledge base, a belief base is the minimal representation of a collection of beliefs. The set of a GOAL agent's all current beliefs is the minimal model of its belief base and knowledge base, and the set of a GOAL agent's all desired beliefs is the minimal model of its goal base and knowledge base.

Definition 6. *(Action Selection)*
Given a GOAL program, let S *be a state of the GOAL agent,* $Theory_{GOAL\_State}$ *is defined by the given GOAL program and the state, let* $Actions$ *be the set of all possible actions of the GOAL agent. We define* $enabled(act, S)$ *to evaluate if an action is enabled at the state as follows:*

$$\forall act \in Actions.enabled(act, S) = \begin{cases} True & \text{if } Theory_{GOAL\_State} \models act \\ False & otherwise \end{cases}$$

Definition 7. *(State Transformer M)*
Given a GOAL agent, let $< \Sigma,\ \Gamma >$ *be a state of the GOAL program, and act be a possible action of the GOAL agent, post_Act (Definition 4) is the postcondition of act, Sub is the unifying substitution of act,* Σ*, and post_Act. Then the state transformer M is defined by:*

$$M(act, < \Sigma,\ \Gamma >) = \begin{cases} < T(act, \Sigma), \Gamma \setminus T(act, \Sigma) > & \text{if } enabled(act, < \Sigma,\ \Gamma >) \\ undefined & otherwise \end{cases}$$

where

$$T(act, \Sigma) = \Sigma \cup \bigcup^{m} D_m[Sub] \setminus \bigcup^{n} E_n[Sub]$$

Definition 8. *(Feasible Trace)*
A feasible trace is a finite sequence $s_0, b_1, s_1, ..., b_n, s_n$ *such that* s_i *is a state of a GOAL agent,* b_i *is an enabled action,* s_0 *is the initial state,* s_n *is the state achieving the desired goal, and for every* i *we have:* $s_i \xrightarrow{b_i} s_{i+1}$*, where* $Theory_{s_i} \models b_{i+1}$.

3 Transformation Algorithm

This section presents an algorithm transforming a substantial subset of GOAL programs to an equivalent transition system in terms of operational semantics. We impose three restrictions on the GOAL programs: single agent, single goal, and stratified GOAL programs. It is worth mentioning here that the stratified GOAL program is not a severe limitation. In most real cases of GOAL programs, there is no negative recursion, which automatically leads to stratified GOAL programs according to Definition 1. On the basis of the intrinsic logic of GOAL, we propose an algorithm for transforming a GOAL program under certain restrictions to an equivalent transition system in terms of operational semantics. At the end of this section, we justify the correctness of the algorithm.

3.1 Algorithm

Algorithm 1 presents an algorithm for transforming an underlying first-order theory of any stratified GOAL program with a single agent and a single goal to an equivalent transition system in terms of operational semantics.

The inputs are the underlying first-order formulas of the logical specifications of a GOAL program: BB denotes the belief base; GB denotes the goal base; KB denotes the knowledge base; *constraints* denotes the constraints of action generation; *enabledness* denotes the enabledness of actions, *ActSpec* denotes the action specifications, and D denotes the domain of all variables occurring in the logical formulas. The output is a transition system following Definition 2.

Algorithm 1 builds a bijection between the state of the transition system and the states of the original GOAL program. Different states of an agent have different sets of beliefs. A belief base represents the necessary beliefs the agent should know, and the agent can derive all current beliefs on the basis of the belief base and knowledge base. Therefore, we encode a state as a belief base.

I denotes the initial state, and I is encoded with the initial belief base BB. F denotes the final state, and F is encoded with the desired belief base GB. *State* denotes all states of the transition system, which is initialized with a list containing I and F. *transitions* denotes all transitions of the transition system, which is initialized with an empty list. The state property of each state is a pair consisting of the set of all current beliefs and the set of all desired beliefs. The beliefs of each state are derived from the belief base, the knowledge base, and the domain of all variables. *atoms_F* denotes the beliefs of the goal state. L is a dictionary storing state properties in each state, which is initialized with a dictionary containing the state property of the goal state. *current_states* denotes all possible current states, which is initialized with a list containing I.

Line 8 -23 follows the same derivation mechanism of the decision-making in GOAL. *next_states* denotes all possible next states of the current state, which is initialized with an empty list. *end_state* denotes the final state where the loop ends, which is updated with the *current_states* at each iteration. Following the derivation mechanism, all possible actions and all possible states are generated. NA denotes all possible next actions of the current state, and NS denotes all possible next states of the current state. The loop will end when there is no feasible action of the current states. If the loop only stops at the final state, the algorithm generates the transition system; otherwise, the algorithm will return None.

Algorithm 1: Generate an equivalent transition system for any stratified GOAL program with a single agent and a single goal in terms of operational semantics

```
Input: First-order logical formulas of GOAL specifications:
       $BB$, $GB$, $KB$, $constraints$, $enabledness$, $ActSpec$, $D$
Output: A transition system-$TS : (S, Act, \rightarrow, I, F, AP, L)$
1  $I = BB$
2  $F = GB$
3  $S = [I, F]$
4  $transitions = []$
5  $atoms\_F$ is derived by $GB$, $KB$, and D.
6  $L = \{F : (atoms\_F, [])\}$
7  $current\_states = [I]$
8  while $current\_states ! = []$ do
9      $next\_states = []$
10     $end\_states = current\_states$
11     for $state$ in the $current\_states$ do
12         if $state$ not in the $S$ then
13             Derive $atoms\_state$ from $state$, $KB$, and $D$.
14             $L.update(state : (atoms\_state, atoms\_F))$
15             $S.append(state)$
16         Derive $NA$ based on $atoms\_state$, $constraints$, $enabledness$.
17         Derive $NS$ based on $next\_actions$ and $ActSpec$.
18         Add $NS$ to the $next\_states$.
19         while $i < len(NS)$ do
20             $transitions.append((state, NA(i), NS(i)))$
21             $i = i + 1$
22         $next\_states.extend(NS)$
23     $current\_states = next\_states$
24 if $\forall s \in end\_states.s == F$ then
25     Add all possible actions to $Act$
26     Add all ground atoms to $AP$
27     return $(S, Act, transitions, I, F, AP, L)$
28 else
29     return None
```

3.2 Algorithm Correctness

Theorem 3. *There is a minimal model for the first-order theory of a stratified GOAL program.*

Proof (sketch). We use Definition 4 to denote the underlying theory of a GOAL program. If a GOAL program is stratified, the GOAL program has no negative recursions by Definition 1. Therefore, The Prolog part of the GOAL program has no negative recursion, and the Prolog part of the GOAL program is stratified by Definition 1. According to Theorem 1, *theory_of_Prolog* has a minimal model, because the Prolog part is stratified.

In $Theory_{GOAL\_State}$, $model_C$ is the minimal model of the knowledge base and the belief base, and $model_G$ is the minimal model of the knowledge base and the goal base. Therefore, $model_C$ and $model_G$ are both uniquely determined. F denotes an *a-goal* predicate in $Theory_{GOAL\_State}$. We denote the minimal model of F as aP, which is constructed as follows:

$$aP = \begin{cases} True & if\ model_C \vdash \neg F \wedge model_G \vdash F \\ False & \text{otherwise} \end{cases}$$

aT denotes all *a-goal* predicates with $True$ interpretation. The minimal model of R is constructed as follows:

$$R = \begin{cases} True & if\ aT \vdash F \wedge model_C \vdash \bigwedge_i F_i \wedge \bigwedge_j \neg A_j \\ False & \text{otherwise} \end{cases}$$

RT denotes all c with $True$ interpretation. The minimal model of Act is constructed as follows:

$$Act = \begin{cases} True & if\ RT \vdash R \wedge model_C \vdash \bigwedge_i F_i \wedge \bigwedge_j \neg A_j \\ False & \text{otherwise} \end{cases}$$

Therefore, for any $Theory_{GOAL\_State}$ of a stratified GOAL program, there is a minimal model.

Theorem 4. *For any stratified GOAL program with a single agent and a single goal with at least one feasible trace (Definition 7), Algorithm 1 generates its equivalent transition system(TS) in terms of operational semantics.*

Proof (sketch).
First, we prove if a GOAL program has a feasible trace, the TS has an equivalent execution (Definition 3) in terms of operational semantics.

We denote the feasible trace as

$$s_0 \xrightarrow{a_1} s_1 \xrightarrow{a_2} ... \xrightarrow{a_n} s_n$$

where $s_i =< \Sigma_i,\ \Gamma_i >$ denotes a state of an GOAL agent, the underlying theory of a state of the GOAL agent is denoted as $Theory_{s_i}$. Given the condition that the agent has a single goal, and it has a feasible trace, therefore, $\Gamma_0 = ... = \Gamma_{n-1},\ \Gamma_n = \emptyset,\ \Sigma_n = \Gamma_0$.

We prove there is an equivalent execution in TS of the feasible trace in terms of operational semantics by induction on n($n \geq 1$).

Induction step ($n = 1$)**:** By Definition 5, Σ_0 denotes the initial belief base, which is the same as the initial state of TS (line 1). Γ_0 denotes the goal base, which is the same as the final state of TS (line 2). Given a knowledge base and a current belief base, all current beliefs held by the agents are uniquely determined. Precisely, Σ_0 and $model_I$ are bijective; Γ_0 and $model_F$ are bijective. Therefore, $< \Sigma_0, \Gamma_0 >$ and $< model_I, model_F >$ are bijective, and we have $L(I) =< model_I, model_F >$. Therefore, $< \Sigma_0, \Gamma_0 >$ is bijective to $L(I)$.

By Definition 8, we have:

$$Theory_{s_0} \models a_1$$

By Theorem 2, we have:

$$Theory_{s_0} \vdash a_1$$

Therefore, a_1 is a element in NA in Algorithm 1 (line 16). Moreover, the current belief base, knowledge base, the selected enabled action, and action specification will uniquely determine the next state of the transition system according to Definition 6 and Definition 7.

If $< \Sigma_1, \Gamma_1 >=< \Gamma_0, \emptyset >$, TS will reach its final state F. According to line 6, we have: $L(F) =< model_F, \emptyset >$. Therefore, $< \Sigma_1, \Gamma_1 >$ and $L(F)$ are bijective.

Otherwise, TS will reach a state S_1, such that $< \Sigma_1, \Gamma_1 >$ and $L(S_1)$ are bijective, where $L(S_1) =< model_{S_1}, model_F >$.

Thus, the claim holds at the induction step.

Induction Hypothesis ($n = i$): For the first-i steps of a feasible trace in a GOAL program:

$$s_0 \xrightarrow{a_1} s_1 \xrightarrow{a_2} ... \xrightarrow{a_i} s_i.$$

In terms of operational semantics, there is an equivalent execution fragment of TS:

$$I \xrightarrow{a_1} S_1 \xrightarrow{a_2} ... \xrightarrow{a_i} S_i, \text{ where } < \Sigma_i,\ \Gamma_i > \text{ and } L(S_i) \text{ are bijective.}$$

Induction Step ($n = i + 1$): For the step, $s_i \xrightarrow{a_{i+1}} s_{i+1}$.

Based on the induction hypothesis, $< \Sigma_i, \Gamma_i >$ and $L(S_i)$ are bijective.

By Definition 8, we have:

$$Theory_{s_i} \models a_{i+1}.$$

By Theorem 2, we have:

$$Theory_{s_i} \vdash a_{i+1}$$

The current belief base, knowledge base, the selected enabled action, and action specification will uniquely determine the next state of the transition system. If $< \Sigma_{i+1}, \Gamma_{i+1} >$ is the desired state of the GOAL program, $< \Sigma_{i+1}, \Gamma_{i+1} >$ and $L(F)$ are bijective; otherwise, $< \Sigma_{i+1}, \Gamma_{i+1} >$ and $L(S_{i+1})$ are bijective.

Therefore, the claim holds for all finite feasible traces of a GOAL program.

Second, we prove if the TS has an execution, the GOAL program has an equivalent feasible trace in terms of operational semantics.

We denote the execution as

$$I \xrightarrow{a_1} S_1 \xrightarrow{a_2} ... \xrightarrow{a_n} F.$$

The state of the transition system and its state properties are bijective. A state property of TS and a state of GOAL agent are also bijective. Therefore, we have a feasible trace as follows:

$$s_I \xrightarrow{a_1} s_1 \xrightarrow{a_2} ... \xrightarrow{a_n} s_F.$$

Remark: If there is no feasible trace of a GOAL program, no transition system will be generated by Algorithm 1. Algorithm 1 (line 24) evaluates if all states stored in *end_states* are the goal states, if one of the ending states is not the goal state, no transition system will be generated.

4 Implementation

We implemented a framework in Python, which generates the equivalent transition of a GOAL program that is stratified, single-agent, and single-goal in terms of the operational semantics. The framework contains two main parts: the first part is the minimal model generation for the first-order theory of a GOAL program; the second part is the implementation of Algorithm 1.

Figure 3 presents the workflow of the generation of the transition system of the framework. We restricted the GOAL programs to stratified GOAL programs. Therefore, we concern the first-order theory underlying stratified GOAL programs. The logical derivation part builds a connection between first-order logical syntax and first-order logical semantics. Especially, it allows the automated generation of the minimal model of a first-order theory. Moreover, it allows the usage of a classical recursive data type - *list*. It is worth mentioning that the implementation for other recursive data types shares the same implementation pattern of *list*. On the basis of the automated logical derivation, we are able to implement Algorithm 1.

Furthermore, we also use an efficient algorithm to calculate the minimal model of a first-order theory. We briefly explain why our algorithm is efficient here. For a first-order theory underlying stratified GOAL programs, it has three key features: finite ground atoms, finite closed formulas, and a finite domain of any variable. The intuitive way of generating the minimal model is to try all possibilities of the combination of all ground atoms. According to Lemma 1, our algorithm distinguishes the universal variables occurring at both sides of an implication (specified with uni_exp_2) and the universal variables occurring at the left side of an implication (specified with uni_exp_1). Instead of the instantiation of all variables, our algorithm only instantiates the variables specified with uni_exp_1. Our algorithm generates the derived ground atoms based on existing atoms and the closed formulas step by step until the minimal model is found. Assume a first-order theory consists of n_1 formulas and n_2 ground atoms. For each formula, we assume it contains at most u variables specified with uni_exp_1, and each variable has at most d interpretations in the domain. The time complexity of the minimal model generation process of our algorithm is $O(n_1 \times d^u \times n_2)$. It is worth mentioning that the number of the variables specified with uni_exp_1 is usually very small, and in many cases, it is 0. The algorithm has low time complexity in most cases. For more details, we refer to the source code [20].

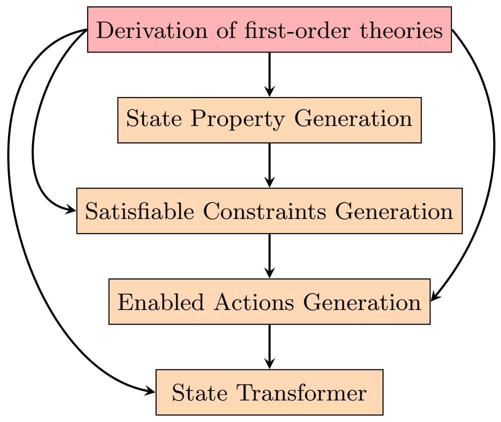
Derivation of first-order theories
State Property Generation
Satisfiable Constraints Generation
Enabled Actions Generation
State Transformer

Fig. 3. Generation of the Transition System

5 Result and Discussion

The major limitation of model checking is the state-space explosion. We use several Blocks World examples to illustrate the state space of the transition system from a GOAL program can be properly controlled.

We chose a series of Blocks World with an increasing number from 2 to 20. Generally speaking, the more blocks are not in the goal state, the more states will be generated. To make the results more convincing, we only chose examples where all blocks are not in the goal state. All Blocks World examples share almost the same GOAL programs except the belief base and the goal base of each example. Table 1 presents the results of the generated transition system of all chosen Blocks World examples. The number of misplaced blocks records the number of blocks that are not in their goal state. The source code of all examples is available at [20].

As we explained in Section 3.1, we encode a state as a belief base. In the Blocks World example, the belief base is the set of all locations of each block. Precisely, for an n-Blocks World example, each state is an n-tuple, where each item can have n possibilities. Mathematically speaking, the state space can up to n^n. However, we hypothesize that the state space can be controlled to not grow exponentially. From the result shown in table 1, the state space is not exponential to the number of blocks. We briefly explain why the state space of the transition system generated from a GOAL program could be controlled

Table 1. The Transition System of Blocks World Example

Number of Blocks	Number of Misplaced Blocks	Number of States	Number of Transitions
2	2	3	2
3	3	5	4
4	4	9	12
5	5	15	23
6	6	22	35
7	7	28	52
8	8	42	82
9	9	58	116
10	10	75	154
11	11	94	196
12	12	114	240
13	13	138	296
14	14	240	622
15	15	366	1010
16	16	498	1415
17	17	636	1837
18	18	1272	4310
19	19	3798	15429
20	20	6330	27825

not to exponentially grow. We can restrict the actions by the enabledness of actions. Precisely, we only allow the action towards the goal state instead of all possibilities. In the Blocks World, the constraints of action generation are specified as follows:

- if constructiveMove(X,Y) then move(X, Y).
- if misplaced(X) then move(X, table).

We only allow a block is moved to its goal state or the table. Therefore, each block moves at most twice with the constraints. The result of these examples shows that we are able to generate the equivalent transition system for a GOAL program with a solvable state space if we impose proper action constraints.

6 Conclusion

The starting point of our work is the automated verification of autonomous decision-making. We use GOAL as the tool to program agents, because it can automatically generate decision-making. We hope to utilize the existing advanced symbolic model checkers to achieve automated verification of GOAL programs. This paper focuses on the key issue of making the model checking feasible for GOAL: the transformation from a GOAL program to its equivalent transition system.

We proposed an algorithm on how to transform a stratified GOAL program with a single agent and a single goal into an equivalent transition system in terms of operational semantics, and we proved the correctness of the algorithm. Therefore, the algorithm makes the model checking approach feasible for a subset of GOAL programs. We also implemented a framework integrating the proposed algorithm, where an equivalent transition system is generated from a stratified GOAL program. Moreover, we illustrate the scalability of the framework through a series of Blocks World examples. From the result, we hypothesize that the state space of a complicated GOAL program can be properly controlled under proper action constraints. Therefore, it is possible to achieve automated formal verification of GOAL if we use an efficient symbolic model checker to conduct the automated verification against the transition system.

Extended research on the automated verification of autonomous decision-making is being conducted. Both the proposed algorithm and the implementation are suitable for a stratified GOAL program with a single agent and a single goal. In future work, we will extend the semantics of GOAL presented in section 2, and we will investigate how to make the approach suitable for all stratified GOAL programs. Then, we will facilitate the existing framework with an advanced symbolic model checker. We will also extend the framework to probabilistic actions, which requires an important change in GOAL: we need to replace embedded Prolog in GOAL with Problog [11]. Next, we will connect the verification framework to the Robot Operating System (ROS). We believe the connection between the verification framework and the ROS will be a crucial step toward verifying real-time autonomous decision-making. Finally, we will explore the possibility of the integration of reinforcement learning and automated verification.

Acknowledgements

This research is partially funded by the Research Fund KU Leuven.

References

1. Baier, C., Katoen, J.P.: Principles of model checking. MIT press (2008)
2. de Boer, F.S., Hindriks, K.V., van der Hoek, W., Meyer, J.J.C.: A verification framework for agent programming with declarative goals. Journal of Applied Logic **5**(2), 277–302 (2007)
3. Bordini, R.H., Fisher, M., Pardavila, C., Wooldridge, M.: Model checking agentspeak. In: Proceedings of the second international joint conference on Autonomous agents and multiagent systems. pp. 409–416 (2003)
4. Dehnert, C., Junges, S., Katoen, J.P., Volk, M.: A storm is coming: A modern probabilistic model checker. In: International Conference on Computer Aided Verification. pp. 592–600. Springer (2017)
5. Dennis, L.A., Fisher, M.: Programming verifiable heterogeneous agent systems. In: International Workshop on Programming Multi-Agent Systems. pp. 40–55. Springer (2008)

6. Dennis, L.A., Fisher, M., Webster, M.P., Bordini, R.H.: Model checking agent programming languages. Automated software engineering **19**(1), 5–63 (2012)
7. Guiochet, J., Machin, M., Waeselynck, H.: Safety-critical advanced robots: A survey. Robotics and Autonomous Systems **94**, 43–52 (2017)
8. Hindriks, K.V.: Programming rational agents in goal. In: Multi-agent programming, pp. 119–157. Springer (2009)
9. Jensen, A.B., Hindriks, K.V., Villadsen, J.: On using theorem proving for cognitive agent-oriented programming. In: 13th International Conference on Agents and Artificial Intelligence. pp. 446–453. Science and Technology Publishing (2021)
10. Jongmans, S.: Model checking goal agents. `https://repository.tudelft.nl/islandora/object/uuid:25d88441-7ecb-4327-9271-19fc76fb0a61?collection=education` (2010)
11. Kimmig, A., Demoen, B., De Raedt, L., Costa, V.S., Rocha, R.: On the implementation of the probabilistic logic programming language problog. Theory and Practice of Logic Programming **11**(2-3), 235–262 (2011)
12. Kwiatkowska, M., Norman, G., Parker, D.: Prism: Probabilistic symbolic model checker. In: International Conference on Modelling Techniques and Tools for Computer Performance Evaluation. pp. 200–204. Springer (2002)
13. Nipkow, T., Paulson, L.C., Wenzel, M.: Isabelle/HOL: a proof assistant for higher-order logic, vol. 2283. Springer Science & Business Media (2002)
14. Parida, S., Franz, M., Abanteriba, S., Mallavarapu, S.: Autonomous driving cars: future prospects, obstacles, user acceptance and public opinion. In: International Conference on Applied Human Factors and Ergonomics. pp. 318–328. Springer (2018)
15. Sergot, M.: Stratified logic programs, lecture notes in 491 knowledge representation (January 2005)
16. Sergot, M.: Stable models ('answer sets'), lecture notes in 491 knowledge representation (November 2016)
17. Shoham, Y.: Agent-oriented programming. Artificial intelligence **60**(1), 51–92 (1993)
18. Vaidya, S., Ambad, P., Bhosle, S.: Industry 4.0–a glimpse. Procedia manufacturing **20**, 233–238 (2018)
19. Wooldridge, M., Fisher, M., Huget, M.P., Parsons, S.: Model checking multi-agent systems with mable. In: Proceedings of the first international joint conference on Autonomous agents and multiagent systems: part 2. pp. 952–959 (2002)
20. Yang., Y.: Transition ststem generation for goal. `https://gitlab.kuleuven.be/u0144511/transition-system-generation-for-goal` (2022)