# Verifying Dribble Agents

Doan Thu Trang, Brian Logan, and Natasha Alechina

The University of Nottingham
School of Computer Science

**Abstract.** We describe a model-checking based approach to verification of programs written in the agent programming language Dribble. We define a logic (an extension of the branching time temporal logic CTL) which describes transition systems corresponding to a Dribble program, and show how to express properties of the agent program in the logic and how to encode transition systems as an input to a model-checker. We prove soundness and completeness of the logic and a correspondence between the operational semantics of Dribble and the models of the logic.

## 1 Introduction

BDI-based agent-oriented programming languages [5] facilitate the implementation of cognitive agents by providing programming constructs to implement concepts such as beliefs, goals, and (pre-defined) plans. In such languages, an agent selects a plan to achieve one or more goals based on its beliefs about the environment. However in anything other than toy environments, selecting an appropriate plan does not guarantee that it can be successfully executed. The beliefs used to select a particular plan for a given goal is only a heuristic, and cannot capture the preconditions of all the actions in the plan (some of which may be false when the plan is selected and will only be made true by actions in the plan). Moreover, in dynamic environments, an agent's beliefs (and hence the best way of achieving a goal) may change in unanticipated ways during plan execution, and a rational agent must be prepared to revise its plans at run time to take advantage of 'fortuitous' changes in the environment (e.g., which allow some steps in the plan to be skipped) or to recover from 'adverse' changes in the environment (e.g., when a precondition of an action is discovered not to hold).

Many BDI-based agent programming languages provide facilities to drop plans if the corresponding goal is 'unexpectedly' achieved or when execution of the plan fails [6, 16, 17]. More advanced languages, e.g., [18, 9] provide support for arbitrary modification of plans during their execution. However, while such meta-level capabilities simplify the development of rational agents, they make it more difficult to reason about the execution of agent programs, e.g., to verify their correctness. In addition to reasoning about the agent's beliefs, goals and plans, we need to model the current state of plan execution, and the evolution of the agent's program at run time in response to interactions between the effects the agent's actions in its environment and its plan revision capabilities.

In this paper we present an approach to verifying agent programs which admit arbitrary revisions at run time. We focus on the BDI agent programming language Dribble introduced in [18]. Dribble allows the implementation of agents with beliefs, (declarative) goals, actions, abstract actions (procedural goals), plans, and rules for selecting and revising plans. Although relatively simple and abstract, it is representative of a wider class of BDI agent programming languages which support plan revision, and presents significant challenges for verification. Our approach is based on model-checking. We define a logic (an extension of the branching time temporal logic CTL) which describes transition systems corresponding to a Dribble program, and show how to express properties of the program in the logic and how to encode transition systems as an input to a model-checker. We prove soundness and completeness of the logic and a correspondence between the operational semantics of Dribble and the models of the logic.

The rest of the paper is organised as follows. In the next section, we describe the syntax and operational semantics of Dribble. In section 3 we introduce a logic for expressing properties of Dribble programs, and give a complete axiomatisation of the set of models corresponding to the operational semantics of a Dribble agent program. We discuss the use of the logic for verification in section 4, where we describe model-checking of Dribble programs and give a simple example of a program and a property which can be model-checked. We give a brief survey of related work in section 5.

## 2 Dribble

In this section, we briefly review the syntax and operational semantics of Dribble.

### 2.1 Beliefs and goals

Let $Prop$ be a finite set of propositional variables and $\mathcal{L}$ the set of propositional formulas. In order to make $\mathcal{L}$ finite, we allow $\mathcal{L}$ to contain only formulas in Disjunctive Normal Form (DNF). A formula is said to be in DNF iff it is a disjunction of conjunctive clauses in which a conjunctive clause is a conjunction of literals. As usual, a literal is either $p$ or $\neg p$ for any $p \in Prop$. Moreover, formulas of $\mathcal{L}$ satisfy the following conditions:

1. formulas do not contain duplicates of conjunctive clauses;
2. conjunctive clauses of a formula do not contain duplicates of literals; and
3. literals in a conjunctive clause of a formula only occur in some fixed order.

A Dribble agent has both a belief base and a goal base which are finite subsets of $\mathcal{L}$. The agent's beliefs and goals are expressed in a language $\mathcal{L}_{BG}$. The syntax of $\mathcal{L}_{BG}$ is defined as follows:

$$\beta \;\leftarrow\; \mathbf{B}\alpha \mid \mathbf{G}\alpha \mid \neg\beta \mid \beta_1 \wedge \beta_2 \text{ where } \alpha \in \mathcal{L}.$$

The meaning of $\mathbf{B}\alpha$ is that $\alpha$ can be propositionally derived from the belief base of an agent, and $\mathbf{G}\alpha$ means that $\alpha$ is the consequence of some single goal in the goal base of an agent. For convenience, a formula $\beta$ is of $\mathcal{L}_B$ ($\mathcal{L}_G$) iff it does not contain any subformula of the form $\mathbf{G}\alpha$ ($\mathbf{B}\alpha$, respectively).

A formula of $\mathcal{L}_{BG}$ is interpreted by a pair of a belief base and a goal base $\langle\delta, \gamma\rangle$, in which both $\delta$ and $\gamma$ are finite subsets of formulas of $\mathcal{L}$. The truth of a formula $\beta$ is defined inductively as follows.

- $\langle \delta, \gamma \rangle \models_{BG} \mathbf{B}\alpha \Leftrightarrow \delta \models_{Prop} \alpha$
- $\langle \delta, \gamma \rangle \models_{BG} \mathbf{G}\alpha \Leftrightarrow \exists g \in \gamma : g \models_{Prop} \alpha$
- $\langle \delta, \gamma \rangle \models_{BG} \neg\varphi \Leftrightarrow \langle \delta, \gamma \rangle \not\models_{BG} \varphi$
- $\langle \delta, \gamma \rangle \models_{BG} \beta \wedge \beta' \Leftrightarrow \langle \delta, \gamma \rangle \models_{BG} \beta$ and $\langle \delta, \gamma \rangle \models_{BG} \beta'$

### 2.2 Plans

A Dribble plan consists of basic actions and abstract plans composed by sequence and conditional choice operators. The sequence operator, ';', takes two plans, $\pi_1, \pi_2$, as arguments and states that $\pi_1$ should be performed before $\pi_2$. The conditional choice operator allows branching and generates plans of the form '`if` $\phi$ `then` $\pi_1$ `else` $\pi_2$'.The syntax of plans is defined as follows:

$$\pi \ \leftarrow \ a \mid b \mid \textit{if } \beta \textit{ then } \pi'_1 \textit{ else } \pi'_2 \mid \pi'_1; \pi'_2$$

where $a$ is an abstract plan, $b$ is a basic action and $\beta \in \mathcal{L}_B$.

We depart from [18] in that we do not have an empty plan (denoted by $E$ in [18]) as a special kind of plan which can occur as part of other plans. Below, we will use $E$ as a marker for an empty plan base, but not as a plan expression, to avoid introducing rewriting rules for $E; E$ to $E$ and $\pi_1; E; \pi_2$ to $\pi_1; \pi_2$, etc.

We define length of a plan $\pi$, $len(\pi)$, inductively as follows:

$$\begin{aligned} len(a) &= 1 \\ len(b) &= 1 \\ len(\textit{if } \beta \textit{ then } \pi'_1 \textit{ else } \pi'_2) &= len(\pi'_1) + len(\pi'_2) + 4 \\ len(\pi'; \pi) &= len(\pi') + len(\pi) \end{aligned}$$

Notice that in the case of the if-then-else statement, the length is the sum of lengths of the plans $\pi'_1$ and $\pi'_2$ together with the number of extra symbols of the statement, i.e. $if$, $then$, $else$ and $\beta$.

Since in reality, agents can hold a plan up to some fixed length, we make an assumption that all plans have length smaller than a certain preset number. Restricting the length of plans also makes the set of plans finite. This is necessary for the axiomatisation of the logic later in the paper.

In the rest of this paper, we denote by $Plans$ the set of all plans whose lengths are smaller than $len_{MAX}$, where $len_{MAX}$ is a natural number.

$$Plans = \{\pi \mid len(\pi) \leq len_{MAX}\}$$

### 2.3 Dribble agents

Writing a Dribble agent means writing a number of goal rules and practical reasoning rules. The syntax of goal rules (PG) and practical reasoning (PR) rules is given below.

- PG rules: $\beta \rightarrow \pi$ where $\beta \in \mathcal{L}_{BG}$ and $\pi \in Plans$
- PR rules: $\pi_1 \mid \beta \rightarrow \pi_2$ where $\beta \in \mathcal{L}_B$ and $\pi_1, \pi_2 \in Plans$, and $\pi_2$ may be empty.

One writes a PG rule to intend that an agent with an empty plan base will generate a plan $\pi$ if its current belief and goal bases satisfy the condition encoded in $\beta$. If the agent has certain goals in its goal base, it will generate a plan based on its beliefs to hopefully achieve those goals. A PR rule proposes a possible revision $\pi_2$ to (the prefix of) a plan $\pi_1$ which is applicable if the belief base satisfies the condition encoded in $\beta$. That is, if the agent has certain beliefs which imply that the current plan will be unable to achieve the intended goal(s) or that the plan is redundant and can be simplified, it can modify the plan. Note that $\pi_2$ can be empty, allowing the agent to drop part or all of a plan.

We have slightly modified the meaning of PR rules given in [18]. In Dribble, these rules apply to complete plans ($\pi_1$ is the agent's plan in its entirety, not a plan prefix, for example a name for an abstract plan). In contrast we allow $\pi_1$ to be a prefix of the agent's plan, which is replaced by $\pi_2$ followed by the continuation of the original plan. We could have written PR rules as $\pi_1'; \pi \mid \beta \rightarrow \pi_2; \pi$ where $\pi$ is a plan variable. In cases where $\pi_1$ matches the entire plan, our PR rules are equivalent to those in [18]. We believe that our generalisation is justified programmatically, and it presents an interesting challenge for logical formalisation, in particular model-checking. To enforce our assumption about the length of plans, we require that Dribble agents consist of PG and PR rules which do not produce plans of length more than $len_{MAX}$.

A Dribble agent only has one intention at a time, i.e., its plan base contains at most one plan and it can apply a goal rule only when its plan is empty, and is strongly committed to its goals, i.e., an agent drops a goal only when it believes that the goal has been achieved.

A **Dribble agent** is a tuple $\langle \delta, \gamma, \Gamma, \Delta \rangle$ in which $\Gamma$ is a set of goal rules, $\Delta$ is set of practical reasoning rules, $\delta$ and $\gamma$ are the initial belief base and goal base and both satisfy the following conditions:

1. $\delta$ is consistent
2. $\forall \alpha \in \gamma, \delta \not\models_{Prop} \alpha$
3. $\forall \alpha \in \gamma$, $\alpha$ is consistent

that is, the agent's beliefs are consistent, it does not have as a goal any formula it already believes to be the case, and each of its goals is consistent (though its goals may be inconsistent with each other as they can be achieved at different times).

## 2.4 Operational semantics

In this section, we describe how a Dribble agent operates. A Dribble program $P$ is a pair $(\Gamma, \Delta)$ of PG rules and PR rules.

A configuration of an agent is a tuple $\langle \delta, \gamma, \{\pi\} \rangle$ where $\delta$, $\gamma$ and $\pi$ are the agent's current belief base, goal base and plan base (where $\pi$ is the current plan, possibly partially executed), respectively. In what follows, we will omit the set brackets around the plan for readability, as in $\langle \delta, \gamma, \pi \rangle$. The plan base can also be empty, which we will write as $\langle \delta, \gamma, \emptyset \rangle$. The *initial configuration* of an agent is $\langle \delta_0, \gamma_0, \emptyset \rangle$.

We specify the operational semantics of a Dribble agent as a set of transition rules. Each transition corresponds to a single execution step and takes the system from one configuration/state to another. In the cases corresponding to applying PG and PR rules,

we have additional conditions to guarantee that we do not produce a plan of length more than $len_{MAX}$. Notice that transitions from a configuration in which the plan base begins with an abstract plan are included in application of PR rules.
Application of a goal rule

$$\frac{\varphi \to \pi \in \Gamma \qquad len(\pi) \leq len_{MAX} \qquad \langle \delta, \gamma \rangle \models_{BG} \varphi}{\langle \delta, \gamma, \emptyset \rangle \quad \longrightarrow_{apply(\varphi \to \pi)} \quad \langle \delta, \gamma, \pi \rangle}$$

Application of a plan revision rule

$$\frac{\pi_1 \mid \beta \to \pi_2 \in \Delta \qquad len(\pi_2; \pi) \leq len_{MAX} \qquad \langle \delta, \gamma \rangle \models_{BG} \beta}{\langle \delta, \gamma, \pi_1; \pi \rangle \quad \longrightarrow_{apply(\pi_1 \mid \beta \to \pi_2)} \quad \langle \delta, \gamma, \pi_2; \pi \rangle}$$

Basic action execution

$$\frac{\mathcal{T}(b, \delta) = \delta' \qquad \gamma' = \gamma \setminus \{g \in \gamma \mid \delta' \models_{Prop} g\}}{\langle \delta, \gamma, b; \pi \rangle \quad \longrightarrow_{execute(b)} \quad \langle \delta', \gamma', \pi \rangle}$$

where $\mathcal{T}$ is a belief update function which takes an action and a belief base and returns the resulting belief base. $\mathcal{T}$ is a partial function since an action may not be applicable in some situations.
Conditional statement

$$\frac{\langle \delta, \gamma \rangle \models_{BG} \beta}{\langle \delta, \gamma, \textit{if } \beta \textit{ then } \pi_1 \textit{ else } \pi_2; \pi \rangle \to_{execute(if)} \langle \delta, \gamma, \pi_1; \pi \rangle}$$

$$\frac{\langle \delta, \gamma \rangle \not\models_{BG} \beta}{\langle \delta, \gamma, \textit{if } \beta \textit{ then } \pi_1 \textit{ else } \pi_2; \pi \rangle \to_{execute(if)} \langle \delta, \gamma, \pi_2; \pi \rangle}$$

Note that in the last three rules, $\pi$ may be absent (or be an empty string), in which case for example executing $b; \pi$ will result in $\langle \delta', \gamma', \emptyset \rangle$.

For technical reasons, if in a configuration $\langle \delta, \gamma, \pi \rangle$ no transition rule is applicable, we assume that there is a special 'stall' transition to the same configuration: $\langle \delta, \gamma, \pi \rangle \to_{stall} \langle \delta, \gamma, \pi \rangle$.

A computation tree $CT(c_0, P)$ for a Dribble agent with a program $P = (\Gamma, \Delta)$ is a tree with root $c_0$ where each node is a configuration, such that for each node $c$ and each child $c'$ of $c$, $c \to c'$ is a transition in the transition system for $P$. The meaning of a Dribble agent $\langle \delta_0, \gamma_0, P \rangle$ is a tree $CT(\langle \delta_0, \gamma_0, \emptyset \rangle, P)$.

## 3 A logic of Dribble programs

In this section, we introduce a logic which allows us to formalize the properties of Dribble agent programs. Formulas of the logic will be used as input to the model-checker. In addition, we give a complete axiomatisation of the models of the logic. Axiomatisation is, of course not necessary for model-checking, but it helps us to understand the logic and its models; for example, the axioms may be more intuitive or clearer than the semantic conditions on models.

The language of our logic $\mathcal{L}_D$ is based on Computation Tree Logic (CTL) [7] which is a logic for reasoning about branching time. The syntax of CTL is as follows:

$$\mathcal{L}_{CTL}: \quad \varphi \leftarrow p \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid EX\varphi \mid E(\varphi U \psi) \mid A(\varphi U \psi) \quad \text{where } p \in \mathcal{L}$$

The meaning of the temporal operators is as follows: $EX\varphi$ means there is a successor state which satisfies $\varphi$; $E(\varphi U \psi)$ means that there is a branch where $\varphi$ holds until $\psi$ becomes true; $A(\varphi U \psi)$ means that on all branches, $\varphi$ holds until $\psi$.

## 3.1 Syntax

$\mathcal{L}_D$ extends $L_{CTL}$ with belief, goal and plan operators ($\mathbf{Bs}$, $\mathbf{Gs}$ and $\mathbf{P}$).

$$\mathcal{L}_D: \quad \varphi \ \leftarrow \ \mathbf{Bs}\delta \mid \mathbf{Gs}\gamma \mid \mathbf{P}\pi \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid EX\varphi \mid E(\varphi U \psi) \mid A(\varphi U \psi)$$

where $\delta, \gamma \subseteq \mathcal{L}$; $\pi \in \mathit{Plans} \cup \{E\}$. $\mathbf{Bs}$ and $\mathbf{Gs}$ describe the belief base and goal base of the agent. Note that these operators apply to *sets* of formulas. $\mathbf{P}$ is used to describe the plan base of the agent. If the agent's plan is $\pi$, this is expressed as $\mathbf{P}\pi$, and if the plan base is empty, this is expressed as $\mathbf{P}E$.

We will use the usual abbreviation:

$AX\varphi = \neg EX\neg\varphi$ (in all successor states, $\varphi$)
$AF\varphi = A(\top U \varphi)$ (on all branches, in some future state, $\varphi$)
$EF\varphi = E(\top U \varphi)$ (there exists a branch, where in some future state, $\varphi$)
$AG\varphi = \neg EF\neg\varphi$ (on all branches, in all states, $\varphi$)
$EG\varphi = \neg AF\neg\varphi$ (there is a branch, where in all states, $\varphi$).

## 3.2 Semantics

In this section we define models for the logic. We show in section 4 that they correspond exactly to the computation trees for Dribble agents generated by the operational semantics.

Given a Dribble agent program $P = (\Gamma, \Delta)$, a Dribble model of P is a triple $M_P = (S, R, V)$ in which:

- $S$ is a nonempty set of states
- $R \subseteq S \times S$ satisfies the properties below
- $V = (V_b, V_g, V_p)$ a collection of three functions, $V_b(s) : S \rightarrow 2^{\mathcal{L}}$, $V_g(s) : S \rightarrow 2^{\mathcal{L}}$ and $V_p(s) : S \rightarrow 2^{Plans}$ satisfying the following conditions, for all $s \in S$:
  1. $V_b(s)$ and $V_g(s)$ are finite subsets of propositional formulas
  2. $V_b(s)$ is consistent
  3. $\forall \alpha \in V_g(s) : V_b(s) \not\models_{Prop} \alpha$
  4. $\forall \alpha \in V_g(s) : \alpha$ is (propositionally) consistent
  5. $V_p(s)$ is a singleton or an empty set.

To simplify the definition, we use the following conventions: $\forall g \in \mathit{\Gamma}$ of the form $\varphi \rightarrow \pi$ then $guard(g) = \varphi$, $body(g) = \pi$, i.e. $guard(g)$ specifies the mental condition for which situation is a good idea to execute the plan, and $body(g)$ is the plan generated after firing the goal rule. Also, $\forall r \in \mathit{\Delta}$ of the form $\pi_1 \mid \beta \rightarrow \pi_2$ then $head(r) = \pi_1$, $guard(r) = \beta$ and $body(r) = \pi_2$. If $V_p(s) = \{\pi\}$, we will write $V_p(s) = \pi$ for readability.

Further requirements for $R$ are listed below.

**EPG:** For all $s \in S$, if $V_p(s) = \emptyset$ and there exists $g \in \mathit{\Gamma}$ such that $\langle V_b(s), V_g(s)\rangle \models_{BG} guard(g)$ then there is $s' \in S$ such that $(s, s') \in R$ with $V_b(s') = V_b(s)$, $V_g(s') = V_g(s)$ and $V_p(s') = body(g)$

**APG:** For all $(s, s') \in R$ such that $V_p(s) = \emptyset$, then $V_b(s') = V_b(s)$, $V_g(s') = V_g(s)$ and there is $g \in \mathit{\Gamma}$ such that $\langle V_b(s), V_g(s)\rangle \models_{BG} guard(g)$ and $V_p(s') = body(g)$

**EBA:** For all $s \in S$, if $V_p(s) = b; \pi$ then there is $s' \in S$ such that $(s, s') \in R$ with $V_b(s') = T(b, V_b(s))$, $V_g(s') = V_g(s) \setminus \{g \in V_g(s) | V_b(s') \models_{Prop} g\}$ and $V_p(s') = \pi$

**EIF:** For all $s \in S$, if $V_p(s) = \pi_{if}; \pi$, where $\pi_{if} =$ *if* $\beta$ *then* $\pi_1$ *else* $\pi_2$, then there is $s' \in S$ such that $(s, s') \in R$ with $V_b(s') = V_b(s)$, $V_g(s') = V_g(s)$ and

$$V_p(s') = \begin{cases} \pi_1; \pi & \text{if } \langle V_b(s), V_g(s)\rangle \models_{BG} \beta \\ \pi_2; \pi & \text{otherwise} \end{cases}$$

**EPR:** For all $s \in S$, if $V_p(s) = \pi_1; \pi$ and there exists $r \in \mathit{\Delta}$ such that $head(r) = \pi_1$ and $\langle V_b(s), V_g(s)\rangle \models_{BG} guard(r)$ then there is $s' \in S$ such that $(s, s') \in R$ with $V_b(s') = V_b(s)$, $V_g(s') = V_g(s)$ and $V_p(s') = body(r); \pi$

**ABAvPR:** For all $(s, s') \in R$, such that $V_p(s) = b; \pi'; \pi$, where $\pi'$ might be empty, then either of the following is true:
1. $V_b(s') = T(b, V_b(s))$, $V_g(s') = V_g(s) \setminus \{g \in V_g(s) \mid V_b(s') \models_{Prop} g\}$ and $V_p(s') = \pi'; \pi$
2. $V_b(s') = V_b(s)$, $V_g(s') = V_g(s)$, and there is $r \in \mathit{\Delta}$ such that $\langle V_b(s), V_g(s)\rangle \models_{BG} guard(r)$, $head(r) = b; \pi'$ and $V_p(s') = body(r); \pi$

**AIFvPR** For all $(s, s') \in R$ such that $V_p(s) = \pi_{if}; \pi'; \pi$ ($\pi'$ might be empty), then either of the following is true:
1. $V_b(s') = V_b(s)$, $V_g(s') = V_g(s)$, and

$$V_p(s') = \begin{cases} \pi_1; \pi'; \pi & \text{if } \langle V_b(s), V_g(s)\rangle \models_{BG} \beta \\ \pi_2; \pi'; \pi & \text{otherwise} \end{cases}$$

2. $V_b(s') = V_b(s)$, $V_g(s') = V_g(s)$, and there is $r \in \mathit{\Delta}$ such that $\langle V_b(s), V_g(s)\rangle \models_{BG} guard(r)$, $head(r) = \pi_{\text{if}}; \pi'$ and $V_p(s') = body(r); \pi$

**APR** For all $(s, s') \in R$ such that $V_p(s) = a; \pi'; \pi$ ($\pi'$ might be empty), then $V_b(s') = V_b(s)$, $V_g(s') = V_g(s)$, and there is $r \in \mathit{\Delta}$ such that $\langle V_b(s), V_g(s)\rangle \models_{BG} guard(r)$, $head(r) = a; \pi'$ and $V_p(s') = body(r); \pi$

For any state $s$ such that there are no transitions $R(s, s')$ required by the conditions above, we stipulate $R(s, s)$. This is required to make the transition relation serial.

No other transitions apart from those required by the conditions above exist in the model.

Given a model $M_P$ and a state $s$ of $M_P$, the truth of a $\mathcal{L}_D$ formula is defined inductively as follows:

- $M_P, s \models \mathbf{Bs}\delta \Leftrightarrow V_b(s) = \delta$
- $M_P, s \models \mathbf{Gs}\gamma \Leftrightarrow V_g(s) = \gamma$
- $M_P, s \models \mathbf{P}\pi \Leftrightarrow V_p(s) = \pi$
- $M_P, s \models \mathbf{P}E \Leftrightarrow V_p(s) = \emptyset$
- $M_P, s \models \neg\varphi \Leftrightarrow M_P, s \not\models \varphi$
- $M_P, s \models \varphi_1 \wedge \varphi_2 \Leftrightarrow M_P, s \models \varphi_1$ and $M_P, s \models \varphi_2$
- $M_P, s \models EX\varphi \Leftrightarrow \exists s' : (s, s') \in R : M_P, s' \models \varphi$
- $M_P, s \models E(\varphi U \psi) \Leftrightarrow \exists$ path $(s_0, s_1, ..., s_n)$ such that: $s_0 = s; n \geq 0; (s_i, s_{i+1}) \in R\ \forall 0 \leq i < n$ and $M_P, s_n \models \varphi$, and for all $i < n$, $M_P, s_i \models \psi$
- $M_P, s \models A(\varphi U \psi) \Leftrightarrow \forall$ paths $(s_0, s_1, ...)$ such that: $s_0 = s$ and $\forall i \geq 0\ (s_i, s_{i+1}) \in R$, exists $n \geq 0$: $M_P, s_n \models \varphi$, and for all $i < n$, $M_P, s_i \models \psi$

Note that the formulas of CTL are evaluated in state $s$ on a tree corresponding to an unravelling of $M_P$ with the root $s$. Without loss of generality, we can assume that each model of $P$ is a tree with the root which intuitively corresponds to the initial configuration of the agent.

### 3.3 Axiomatization

We will refer to the axiom system below as the Dribble logic of a program $P$, $DL_P$. To simplify the axioms, we use $guard(g)$, $body(g)$, $head(r)$, $guard(r)$ and $body(r)$ with the same meanings as in the model. Finally, we use $\pi_{if}$ for *if* $\beta$ *then* $\pi_1$ *else* $\pi_2$.

**CL** classical propositional logic
**CTL** axioms of CTL
**A1a** $\bigvee_{\delta \subseteq \mathcal{L}} \mathbf{Bs}\delta$
**A1b** $\mathbf{Bs}\delta \rightarrow \neg\mathbf{Bs}\delta', \forall\delta' \neq \delta$
An agent has only one belief base.
**A2a** $\bigvee_{\gamma \subseteq \mathcal{L}} \mathbf{Gs}\gamma$
**A2b** $\mathbf{Gs}\gamma \rightarrow \neg\mathbf{Gs}\gamma', \forall\gamma' \neq \gamma$
An agent has only one goal base.
**A3a** $\bigvee_{\pi \in Plans \cup \{E\}} \mathbf{P}\pi$
**A3b** $\mathbf{P}\pi \rightarrow \neg\mathbf{P}\pi'$, where $\pi, \pi' \in Plans \cup \{E\}, \forall\pi' \neq \pi$
An agent has only one plan.
**A4** $\neg\mathbf{Bs}\delta, \forall\delta$ such that $\delta \models_{Prop} \bot$
Belief base is consistent.
**A5** $\mathbf{Bs}\delta \rightarrow \neg\mathbf{Gs}\gamma$ for all $\gamma$ such that $\exists g \in \gamma : \delta \models_{Prop} g$
All goals in goal base are not consequences of belief base.
**A6** $\neg\mathbf{Gs}\gamma$ for all $\gamma$ such that $\exists g \in \gamma : g \models_{Prop} \bot$
Each goal in goal base is consistent.

**EPG** $\mathbf{Bs}\delta \wedge \mathbf{Gs}\gamma \wedge \mathbf{P}E \rightarrow EX(\mathbf{Bs}\delta \wedge \mathbf{Gs}\gamma \wedge \mathbf{P}\pi)$ if $\exists g \in \Gamma$ such that $\langle \delta, \gamma \rangle \models_{BG} guard(g)$ and $\pi = body(g)$
In a state $s$ where some planning goal rule is applicable, i.e. the current plan is empty, there exists a next state $s'$ where its plan is the one generated by firing the planning goal rule.

**APG** $\mathbf{Bs}\delta \wedge \mathbf{Gs}\gamma \wedge \mathbf{P}E \rightarrow AX(\bigvee_{g \in \Gamma'} (\mathbf{Bs}\delta \wedge \mathbf{Gs}\gamma \wedge \mathbf{P}\pi_g))$ where $\Gamma'$ is a set of planning goal rules $g$ that satisfies the following two conditions: $\langle \delta, \gamma \rangle \models_{BG} guard(g)$ and $\pi_g = body(g)$, provided $\Gamma' \neq \emptyset$.
In a state $s$ where the current plan is empty, all possible next states from $s$ are only reachable by applying some PG rule, i.e. its plan is generated by firing the PG rule.

**EBA** $\mathbf{Bs}\delta \wedge \mathbf{Gs}\gamma \wedge \mathbf{P}(b;\pi) \rightarrow EX(\mathbf{Bs}\delta' \wedge \mathbf{Gs}\gamma' \wedge \mathbf{P}\pi)$ where $\delta' = T(b, \delta)$ and $\gamma' = \gamma \setminus \{g \mid \delta' \models_{Prop} g\}$
In a state $s$ where a basic action is applicable, there exists a next state $s'$ in which the basic action is removed from its plan, and the belief base is updated according to the basic action (the goal base, therefore, also has to be changed in order to maintain the disjointness with the belief base).

**EIF** $\mathbf{Bs}\delta \wedge \mathbf{Gs}\gamma \wedge \mathbf{P}(\pi_{if};\pi) \rightarrow EX(\mathbf{Bs}\delta \wedge \mathbf{Gs}\gamma \wedge \mathbf{P}(\pi_i;\pi))$
where

$$\pi_i = \begin{cases} \pi_1 & \text{if } \langle V_b(s), V_g(s) \rangle \models_{BG} \beta \\ \pi_2 & \text{otherwise} \end{cases}$$

In a state $s$ where the current plan begins with a conditional plan, there exists a next state $s'$ in which the conditional plan is replaced by one of its two sub plans depending on whether its condition is derivable or not from the belief base in $s$, respectively.

**EPR** $\mathbf{Bs}\delta \wedge \mathbf{Gs}\gamma \wedge \mathbf{P}(\pi_1;\pi) \rightarrow EX(\mathbf{Bs}\delta \wedge \mathbf{Gs}\gamma \wedge \mathbf{P}(\pi_2;\pi))$
if $\exists r \in \Delta$ such that $\langle \delta, \gamma \rangle \models_{BG} guard(r)$, $head(r) = \pi_1$ and $\pi_2 = body(r)$
In a state $s$ where a plan revision rule is applicable, i.e. the head of the rule is the beginning of the current state and the guard of the rule is derivable from the current belief and goal base, there exists a next state $s'$ in which the beginning of the plan in $s$ is replaced by the body of the rule.

**ABAvPR** $\mathbf{Bs}\delta \wedge \mathbf{Gs}\gamma \wedge \mathbf{P}(b;\pi';\pi) \rightarrow AX((\mathbf{Bs}\delta' \wedge \mathbf{Gs}\gamma' \wedge \mathbf{P}(\pi';\pi)) \vee \bigvee_{r \in \Delta'} (\mathbf{Bs}\delta' \wedge \mathbf{Gs}\gamma' \wedge \mathbf{P}(\pi'';\pi)))$ where $\Delta'$ is a set of plan revision rules $r$ that satisfies the following three conditions: $head(r) = b;\pi'$, $\langle \delta, \gamma \rangle \models_{BG} guard(r)$ and $body(r) = \pi''$, provided $\Delta' \neq \emptyset$ or $\mathcal{T}(\lfloor, \delta)$ is defined.
In a state $s$ where a basic action $b$ is the first element of the plan, we can only transit to another state by executing the action or applying an applicable practical reasoning rule.

**AIFvPR** $\mathbf{Bs}\delta \wedge \mathbf{Gs}\gamma \wedge \mathbf{P}(\pi_{if};\pi';\pi) \rightarrow AX((\mathbf{Bs}\delta' \wedge \mathbf{Gs}\gamma' \wedge \mathbf{P}(\pi_i;\pi';\pi)) \vee \bigvee_{r \in \Delta'} (\mathbf{Bs}\delta' \wedge \mathbf{Gs}\gamma' \wedge \mathbf{P}(\pi'';\pi)))$ where $\Delta'$ is a set of plan revision rules $r$ that satisfy three following conditions: $head(r) = \pi_{if};\pi'$, $\langle \delta, \gamma \rangle \models_{BG} guard(r)$ and $body(r) = \pi''$; and

$$\pi_i = \begin{cases} \pi_1 & \text{if } \langle V_b(s), V_g(s) \rangle \models_{BG} \beta \\ \pi_2 & \text{otherwise} \end{cases}$$

In a state $s$ where an if-then-else statement is the first element of the plan, we can only transit to another state by executing the if-then-else statement or applying an applicable practical reasoning rule.

**APR** $\mathbf{Bs}\delta \wedge \mathbf{Gs}\gamma \wedge \mathbf{P}(a;\pi_1;\pi) \rightarrow AX \bigvee_{r\in\Delta'} (\mathbf{Bs}\delta' \wedge \mathbf{Gs}\gamma' \wedge \mathbf{P}(\pi_2;\pi))$ where $\Delta'$ is a set of plan revision rules $r$ that satisfy three following conditions: $head(r) = a;\pi_1$, $\langle\delta,\gamma\rangle \models_{BG} guard(r)$ and $body(r) = \pi_2$; provided $\Delta' \neq \emptyset$.
In a state $s$ where an abstract plan is the first element of the plan, we can only transit to another state by applying a practical reasoning rule.

**Stall** $\mathbf{Bs}\delta \wedge \mathbf{Gs}\gamma \wedge \mathbf{P}\pi \rightarrow AX(\mathbf{Bs}\delta \wedge \mathbf{Gs}\gamma \wedge \mathbf{P}\pi)$ where $\mathbf{Bs}\delta \wedge \mathbf{Gs}\gamma \wedge \mathbf{P}\pi$ describes a configuration from which no normal transitions are available.

We have the following result.

**Theorem 1.** *$DL_P$ is sound and complete with respect to the class of models of the program $P$.*

*Proof.* The proof of soundness is straightforward and is omitted. In the rest of this section, we show the completeness of $DL_P$. Most of the proof is from that of $CTL$ [13].

Let $BGP = 2^{\mathcal{L}} \times 2^{\mathcal{L}} \times (Plans \cup \{E\})$. $BGP$ intuitively corresponds to the set of all possible configurations. Note that this is a finite set.

Given a consistent formula $\varphi_0$, we construct the generalised Fischer-Ladner closure of $\varphi_0$, $FL(\varphi_0)$, as the least set $H$ of formulas containing $\varphi_0$ such that:

1. $\mathbf{Bs}\delta \in H$ for all $\delta \subseteq \mathcal{L}$
2. $\mathbf{Gs}\gamma \in H$ for all $\gamma \subseteq \mathcal{L}$
3. $\mathbf{P}\pi \in H$ for all $\pi \in Plans \cup \{E\}$
4. $EX(\mathbf{Bs}\delta \wedge \mathbf{Gs}\gamma \wedge \mathbf{P}\psi)$ for all $(\delta,\gamma,\pi) \in BGP$
5. $EX(\bigvee_{(\delta,\gamma,\pi)\in BGP'} (\mathbf{Bs}\delta \wedge \mathbf{Gs}\gamma \wedge \mathbf{P}\psi))$ for all $BGP' \subseteq BGP$
6. $\neg\varphi \in H$, then $\varphi \in H$
7. $\varphi \wedge \psi \in H$, then $\varphi, \psi \in H$
8. $E(\varphi U \psi) \in H$, then $\varphi$, $EXE(\varphi U \psi) \in H$
9. $A(\varphi U \psi) \in H$, then $\varphi$, $AXA(\varphi U \psi) \in H$
10. $EX\varphi \in H$, then $\varphi \in H$
11. $AX\varphi \in H$, then $\varphi \in H$
12. $\varphi \in H$ and $\varphi$ is not of the form $\neg\psi$, then $\neg\varphi \in H$

It is obvious that $FL(\varphi_0)$ is finite. As usual, we define a subset $s$ of $FL(\varphi_0)$ that is maximally consistent if $s$ is consistent and for all $\varphi$, $\neg\varphi \in FL(\varphi_0)$, either $\varphi$ or $\neg\varphi$ is in $s$. Repeat the construction of a model $M$ for $\varphi_0$ as in [13] based on the set of maximally consistent sets of $FL(\varphi_0)$, with the condition that the assignments are as follows:

– $V_b(s) = \delta$ for any $\delta$ such that $\mathbf{Bs}\delta \in s$
– $V_g(s) = \gamma$ for any $\gamma$ such that $\mathbf{Gs}\gamma \in s$
– $V_p(s) = \pi$ for any $\pi$ such that $\mathbf{P}\pi \in s$ (and $V_p(s) = \emptyset$ if $\mathbf{P}E \in s$).

The above definition is well-defined because axioms **A1x**, **A2x** and **A3x** guarantee that there are exactly one $\mathbf{Bs}\delta \in s$, $\mathbf{Gs}\gamma \in s$ and $\mathbf{P}\pi \in s$. Our remaining task is to show that $M$ is, in fact, a model of $P$.

**EPG:** Assume that $V_b(s) = \delta$, $V_g(s) = \gamma$ and $V_p(s) = \emptyset$, then we have $\mathbf{Bs}\delta \wedge \mathbf{Gs}\gamma \wedge \mathbf{P}E \in s$. Furthermore, assume that there is $g \in \Gamma$ such that $\langle \delta, \gamma \rangle \models_{BG} guard(g)$. By axiom **EPG** and modus ponens (MP), $EX(\mathbf{Bs}\delta \wedge \mathbf{Gs}\gamma \wedge \mathbf{P}body(g)) \in s$. According to the construction of $M$, there is $s'$ such that $\mathbf{Bs}\delta \wedge \mathbf{Gs}\gamma \wedge \mathbf{P}body(g) \in s'$ and $(s, s') \in R$. It is obvious that $V_b(s') = \delta$, $V_g(s') = \gamma$ and $V_p(s') = body(g)$.

**APG:** Assume that $V_b(s) = \delta$, $V_g(s) = \gamma$ and $V_p(s) = \emptyset$, then we have $\mathbf{Bs}\delta \wedge \mathbf{Gs}\gamma \wedge \mathbf{P}E \in s$. Let $\Gamma'$ is the set of PG rules $g \in \Gamma$ such that $\langle \delta, \gamma \rangle \models_{BG} guard(g)$. By axiom **APG** and modus ponens (MP), $AX(\bigvee_{g \in \Gamma'} (\mathbf{Bs}\delta \wedge \mathbf{Gs}\gamma \wedge \mathbf{P}body(g))) \in s$. According to the construction of $M$, for any $s'$ such that $(s, s') \in R$,

$$\bigvee_{g \in \Gamma'} (\mathbf{Bs}\delta \wedge \mathbf{Gs}\gamma \wedge \mathbf{P}body(g)) \in s$$

Then, there exists $g \in \Gamma'$ such that $\mathbf{Bs}\delta \wedge \mathbf{Gs}\gamma \wedge \mathbf{P}body(g) \in s'$. It is obvious that $V_b(s') = \delta$, $V_g(s') = \gamma$ and $V_p(s') = body(g)$.

The proof of other conditions on $R$ is similar to the two cases above and are omitted. This shows that $M$ is a model of the program $P$.

## 4 Verification

We can express properties of Dribble programs using CTL operators in the usual way. For example, a safety property that 'nothing bad will happen' can be expressed as $AG\neg\phi$, where $\phi$ is a description of the 'bad' situation. Similarly, a liveness property that 'something good will happen', for example the agent will achieve its goal, can be expressed as $EF\phi$, where $\phi$ is a description of the 'good' situation. It is essential however that we know that we are verifying the properties with respect to the computation trees which precisely correspond to the operational semantics of the agent. We prove in the next section that models of the logic correspond to computation trees, and hence that a CTL formula is true at the root of a Dribble model for $P$ if, and only if, the corresponding property holds for the initial configuration $c_0$ of its computation tree $CT(c_0, P)$.

### 4.1 Correspondence theorem

We say that a state $s$ of some model $M_P$ is corresponding to a configuration $c \in CT(c_0, P)$ (notation: $s \sim c$) iff $c = \langle V_b(s), V_g(s), V_p(s) \rangle$.

Consider a Dribble agent $\langle \delta_0, \gamma_0, P \rangle$ and its computational tree $CT(\langle \delta_0, \gamma_0, \emptyset \rangle, P)$. We claim that it is isomorphic to the Dribble model of $P$ with the root $s_0$ such that $s_0 \sim \langle \delta_0, \gamma_0, \emptyset \rangle$ and for every state $s$, all children of $s$ are distinct. The last condition is required for isomorphism; there may be Dribble models for $P$ where there are duplicated transitions to identical states. Such duplication of identical successors does not affect the truth of $DL_P$ formulas.

**Theorem 2.** *$CT(c_0, P)$ is isomorphic to the Dribble model $M_P$ of $P$ with the root $s_0$ such that $s_0 \sim c_0$ satisfying the condition that for every state $s$, all children of $s$ are distinct.*

*Proof.* We are going to show that $\sim$ defines a bijection between the states of $CT(c_0, P)$ and $M_P$. We prove the theorem by induction on the distance from the root of the tree.

**Base case:** Assume that $(s_0, s) \in R$ in $M_P$. We will show that there exists a unique $c_0$ with $c_0 \to c$ in $CT(c_0, P)$ and $s \sim c$. The other direction (if $c_0 \to c$ then there is a unique $s$ such that $R(s_0, s)$ and $s \sim c$) is similar.

Case 1: Assume that $V_p(s_0) = \emptyset$, by **APG**, there is $g \in \Gamma$ such that

$$\langle V_b(s_0), V_g(s) \rangle \models_{BG} \mathit{guard}(g)$$

Furthermore, $V_b(s) = V_b(s_0)$, $V_g(s) = V_g(s_0)$ and $V_p(s) = \mathit{body}(g)$. Let $c = \langle V_b(s), V_g(s), V_p(s) \rangle$. By the operational semantics, $c_0 \to_{\mathit{apply}(g)} c$.

Case 2: Assume that $V_p(s_0) = b; \pi'; \pi$ ($\pi'$ might be empty). As $(s_0, s) \in R$, **ABAvPR** implies that there are two cases to consider. In the first case, $V_b(s) = T(b, V_b(s_0))$, $V_g(s) = V_g(s_0) \setminus \{\alpha \in V_g(s_0) \mid V_b(s) \models_{\mathit{Prop}} \alpha\}$ and $V_p(s) = \pi'; \pi$. Simply let $c = \langle V_b(s), V_g(s), \pi'\pi \rangle$, we have that $c_0 \to_{\mathit{execute}(b)} c$. In the second case, we have $V_b(s) = V_b(s_0)$, $V_g(s) = V_g(s_0)$ and there is $r \in \Delta$ such that $\mathit{head}(r) = b; \pi'$ and

$$\langle V_b(s_0), V_g(s_0) \rangle \models_{BG} \mathit{guard}(r)$$

and $V_p(s) = \mathit{body}(r); \pi$. Let $c = \langle V_b(s), V_g(s), \mathit{body}(r); \pi \rangle$, then we have $c_0 \to_{\mathit{apply}(r)} c$.

For the other cases of $V_p(s_0)$, the proof is done in a similar manner by using the suitable conditions of $R$.

**Induction step:** Assume that the path from $s_0$ to $s$ has length $n > 1$. That means there are $s_1, \ldots, s_n = s$ in $M_P$ such that $(s_i, s_{i+1}) \in R$ for all $i > 0$. By the induction hypothesis, there are $c_1, \ldots, c_{n-1}$ such that $s_i \sim c_i$ for all $i = 0, \ldots, n-1$ and $c_i \to_{x_i} c_{i+1}$ by some

$$x_i \in \{\mathit{execute}(b), \mathit{execute}(if), \mathit{apply}(g), \mathit{apply}(r)\}$$

By repeating the proof of the base case, we have that there is $c_n$ such that $s_n \sim c_n$ and $c_{n-1} \to_x c_n$ by some

$$x \in \{\mathit{execute}(b), \mathit{execute}(if), \mathit{apply}(g), \mathit{apply}(r)\}.$$

## 4.2 Automated verification

In this section, we show how to encode $DL_P$ models for a standard CTL model checker to allow the automated verification of properties of Dribble programs. For the examples reported here, we have used the MOCHA model checker [3], due to the ease with which we can specify Dribble programs in *reactive modules*, the description language used by MOCHA.[1]

[1] Note that model checkers such as MCMAS [15] intended for the verification of multi-agent systems are not appropriate, as they assume that epistemic modalities are defined in terms of accessibility relations, rather than syntactically as in $\mathcal{L}_D$.

States of the $DL_P$ models correspond to an assignment of values to state variables in the model checker. In particular, the agent's mental state is encoded as a collection of state variables. The agent's goals and beliefs are encoded as boolean variables with the appropriate initial values. The agent's plan is encoded as an array of *steps* of length $len_{MAX} + 1$. *step* is an enumeration type which includes all basic actions and abstract plans declared in the agent's program, and a set of special *if-tokens*. Each if-token $(\beta, u, v)$ corresponds to an if-then-else construct appearing one of the agent's plans, and encodes the belief(s) tested, $\beta$ and the lengths of the 'then' and 'else' branches (denoted by $u$ and $v$ respectively). All elements of the plan array are initially assigned the value *null*

The execution of plans and the application of goal and practical reasoning rules are encoded as a MOCHA *atom* which describes the initial condition and transition relation for the variables corresponding to the agent's belief and plan bases. A basic action is performed if the corresponding step token is the first element in the plan array. Executing the action updates the agent's beliefs appropriately and advances the plan, i.e., for each plan element $i > 0$, the step at location $i$ is moved to location $i - 1$. If the first element of the plan array is an if-token, a test is performed on the appropriate belief(s) and the plan advanced accordingly. For example, if the first element of the plan array is $(\beta, u, v)$ and $\beta$ is false, the plan is advanced $u + 1$ steps. Goal rules can be applied when the agent has no plan, i.e., when the first element of the plan array contains 'null', and the rule's mental condition holds (i.e., the agent has the appropriate beliefs and goals). Firing the rule writes the plan which forms the body of the goal rule into the plan array. Practical reasoning rules can be applied when the plan to be revised matches a prefix of the plan array and the rule's belief condition is believed by the agent. Firing the rule writes the plan which forms the body of the rule into the plan array and appends the suffix of the original plan (if any). In the case in which the first element of the plan array is an abstract action, application of the appropriate practical reasoning rule inserts the plan corresponding to the abstract action at the beginning of the plan array. An additional atom encodes the agent's commitment strategy, and drops any goals the agent has come to believe as a result of executing a basic action.

The evolution of the system's state is described by an initial round followed by an infinite sequence of update rounds. State variables are initialised to their initial values in the initial round and new values are assigned to the variables in the subsequent update rounds. At each update round, MOCHA non-deterministically chooses between executing the next step in the plan (if any) and firing any applicable goal and practical reasoning rules.

## 4.3 Example

As an illustration, we show how to prove properties of a simple agent program written in Dribble. Consider the following Dribble program for a simple 'vacuum cleaner' agent. The agent's environment consists of two rooms, room1 and room2, and its goal is to clean both rooms. The agent has actions which allow it to move between rooms and to clean a room, and goal rules which allow it to select an appropriate plan to clean a room. To clean a room the agent's battery must be charged and cleaning discharges the battery. The agent has a PR rule which revises a plan which is not executable because the battery has become discharged. The agent's beliefs and goals are expressed using the following propositional variables: $c1$, $c2$ which mean that room1 and room2 are clean, $r1$, $r2$ which mean that the agent is in room1 and room2, and $b$ means that the agent's battery is charged. The agent has the following five basic actions:[2]

- $mR$ for 'move right'. It is applicable if the agent's belief base $\delta$ is such that $\delta \models_{prop} r1$, for example $\delta = \{b \wedge \neg c1 \wedge \neg c_2 \wedge r1 \wedge \neg r2\}$. For this particular $\delta$, $\mathcal{T}(mR, \delta) = \{b \wedge \neg c1 \wedge \neg c_2 \wedge \neg r1 \wedge r2\}$, that is, the agent no longer believes that it is in $r1$ but believes that it is in $r2$. In general, $\mathcal{T}(mR, \delta)$ is defined as follows: for every $\alpha \in \delta$, in every disjunct in $\alpha$ containing $r1$, replace $r1$ with $\neg r1$ and for every disjunct containing $\neg r2$, replace $\neg r2$ with $r2$.
- $mL$ for 'move left'. The belief update function is defined analogously.
- $cR$ for 'clean room'. It is applicable if the agent's belief base $\delta$ is such that $\delta \models_{prop} b$. If the action is executed in room1, it makes $c1$ true, similarly for executing $cR$ in room2. In both cases $b$ becomes false.
- $cB$ for 'charge battery'. It is only applicable in $r2$ (intuitively because this is where the charger is) and it makes $b$ true.

The agent's program consists of the following two goal rules:

$$g_1 = c1 \ \rightarrow \ \texttt{if}\ r1\ \texttt{then}\ cR\ \texttt{else}\ mL; cR$$
$$g_2 = c2 \ \rightarrow \ \texttt{if}\ r2\ \texttt{then}\ cR\ \texttt{else}\ mR; cR$$

and the PR rule:

$$r_1 = cB; \pi \mid r1 \ \rightarrow mR; cB; \pi$$

We would like to verify that, starting in a state in which the agent believes that it is in room1, $r1$, and its battery is charged, $b$, the above program results in the agent achieving its goals $c1$ and $c2$. This can be achieved by verifying, for the corresponding model, that $AF(Bc1 \wedge Bc2)$ is true in the initial configuration where $b$, $\neg c_1$, $\neg c_2$, $r1$ and $\neg r2$ are true.

## 5 Related work

A logic for proving properties of Dribble agents is presented in [18], based on dynamic logic rather than on CTL. However, no axiomatisation of the logic or automated verification procedure is given. In [1, 2], Alechina et al. introduced a logic which is also based on dynamic logic for verification of agent programs written in a sublanguage of 3APL, but this work does not consider rules to revise plans. In [10] Dastani et al. prove a correspondence between an APL-like agent programming language and a specification language based on CTL, and show that any agent implemented in the language satisfies some desirable properties, e.g., relating to commitment strategies. In contrast, our aim in this paper is to verify whether a given property holds for a particular agent program. In addition, the APL considered by Dastani et al. does not include practical reasoning

[2] For simplicity, we omit abstract actions.

rules, and hence their results are confined to agents which are unable to revise their plans.

A strand of work on model-checking properties of agent programming languages is represented by [4] and continued by [11, 12] who use Java Path Finder (JPF) which is a model checker for Java programs. This approach requires writing a Java interpreter for the language using Agent Infrastructure Layer (AIL). After that, verification proceeds as for a standard Java program, without exploiting any features specific to agent programming languages.

A model-checking approach to automated verification of ConGolog programs was described in [8]. The paper proposes a very expressive logic which includes first-order language for specifying properties of programs and defines a model-checking algorithm for the logic. Due to the first-order sublanguage, the algorithm is not guaranteed to terminate. While ConGolog is a very expressive language, it differs from the APL and Dribble family of languages in that it lacks an explicit mechanism for revising plans.

MetateM [14] is another language for agent programming which is based on executable temporal statements. Although it is easy to verify automatically properties of agents written in MetateM, the language is very different from agent programming languages such as 3APL, AgentSpeak and Dribble, where plan constructs are based on conventional imperative languages (e.g. plans with branching constructs and loops).

## 6 Conclusion

This paper describes a temporal logic which allows us to axiomatise the set of transition systems generated by the operational semantics of a Dribble program, and formulate properties of a Dribble agent, such as that the agent is guaranteed to achieve its goals or is not going to violate some safety restrictions. One of the interesting properties of Dribble are practical reasoning or plan rewriting rules, and we believe that they pose interesting challenges for logical formalisation; in particular, we had to introduce explicit 'plan operators' in the language to model those rules. We show how to encode the models of the logic as input to a standard model-checker, which gives an automatic procedure for verifying properties of a Dribble program.

## References

1. N. Alechina, M. Dastani, B. Logan, and J.-J. C. Meyer. A logic of agent programs. In *Proc. of the 22nd National Conf. on Artificial Intelligence (AAAI 2007)*, pages 795–800. AAAI Press, 2007.
2. N. Alechina, M. Dastani, B. Logan, and J.-J. C. Meyer. Reasoning about agent deliberation. In *Proc. of the 11th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR'08)*, pages 16–26. AAAI, 2008.
3. R. Alur, T. A. Henzinger, F. Y. C. Mang, S. Qadeer, S. K. Rajamani, and S. Tasiran. MOCHA: Modularity in model checking. In *Computer Aided Verification*, pages 521–525, 1998.
4. R. Bordini, M. Fisher, W. Visser, and M. Wooldridge. State-space reduction techniques in agent verification. In *Proceedings of the 3rd Int. Joint Conf. on Autonomous Agents and Multi-Agent Systems (AAMAS-2004)*, pages 896–903, New York, NY, 2004. ACM Press.

5. R. H. Bordini, M. Dastani, J. Dix, and A. E. Fallah-Seghrouchni, editors. *Multi-Agent Programming: Languages, Platforms and Applications*, volume 15 of *Multiagent Systems, Artificial Societies, and Simulated Organizations*. Springer, 2005.
6. R. H. Bordini, J. F. Hübner, and R. Vieira. ***Jason*** and the Golden Fleece of agent-oriented programming. In *Multi-Agent Programming: Languages, Platforms and Applications*, chapter 1. Springer-Verlag, 2005.
7. E. M. Clarke and E. A. Emerson. Design and synthesis of synchronization skeletons using branching time temporal logic. In D. Kozen, editor, *IBM Logics of Programs Workshop*, number 131 in LNCS, pages 52–71. Springer-Verlag, 1981.
8. J. Claßen and G. Lakemeyer. A logic for non-terminating Golog programs. In *Proceedings of the 11th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR'08)*, pages 589–599. AAAI, 2008.
9. M. Dastani, M. B. van Riemsdijk, F. Dignum, and J.-J. C. Meyer. A programming language for cognitive agents goal directed 3apl. In *Programming Multi-Agent Systems*, LNCS, pages 111–130. Springer, 2004.
10. M. Dastani, M. B. van Riemsdijk, and J.-J. C. Meyer. A grounded specification language for agent programs. In *Proceedings of the Sixth International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS'07)*, pages 578–585, New York, , USA, 2007. ACM Press.
11. L. A. Dennis, B. Farwer, R. H. Bordini, and M. Fisher. A flexible framework for verifying agent programs. In *7th Int. Joint Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2008)*, pages 1303–1306. IFAAMAS, 2008.
12. L. A. Dennis and M. Fisher. Programming verifiable heterogeneous agent systems. In *6th Int. Workshop on Programming in Multi-Agent Systems (ProMAS'08)*, pages 27–42, 2008.
13. E. A. Emerson and J. Y. Halpern. Decision procedures and expressiveness in the temporal logic of branching time. *J. Comput. Syst. Sci.*, 30(1):1–24, 1985.
14. M. Fisher. Metatem: The story so far. In *Programming Multi-Agent Systems, 3rd Int. Workshop, (ProMAS 2005)*, volume 3862 of *Lecture Notes in Computer Science*, pages 3–22. Springer, 2006.
15. A. Lomuscio and F. Raimondi. MCMAS: A model checker for multi-agent systems. In H. Hermanns and J. Palsberg, editors, *Proceedings of the 12th International Conference on Tools and Algorithms for the Construction and Analysis of Systems, (TACAS 2006)*, volume 3920 of *Lecture Notes in Computer Science*, pages 450–454. Springer, 2006.
16. D. Morley and K. Myers. The SPARK agent framework. In *Proc. of the 3rd Int. Joint Conf. on Autonomous Agents and Multiagent Systems (AAMAS'04)*, pages 714–721. IEEE Computer Society, 2004.
17. J. Thangarajah, J. Harland, D. Morley, and N. Yorke-Smith. Aborting tasks in bdi agents. In *Proc. of the 6th Int. Joint Conf. on Autonomous Agents and Multi Agent Systems (AAMAS'07)*, pages 8–15, 2007.
18. B. van Riemsdijk, W. van der Hoek, and J.-J. C. Meyer. Agent programming in dribble: from beliefs to goals using plans. In *Proc. of the 2nd Int. Joint Conf. on Autonomous Agents and Multiagent Systems (AAMAS'03)*, pages 393–400. ACM Press, 2003.