EasyChair Preprint

№ 1491

Temporal Logic Semantics for Teleo-Reactive Robotic Agent Programs

Keith Clark, Brijesh Dongol and Peter Robinson

September 12, 2019

Temporal Logic Semantics for Teleo-Reactive Robotic Agent Programs*

Keith Clark[1], Brijesh Dongol[2], and Peter Robinson[3]

[1] Imperial College London, UK k.clark@imperial.ac.uk
[2] University of Surrey, UK b.dongol@surrey.ac.uk
[3] University of Queensland, Australia pjr@itee.uq.edu.au

Abstract. Teleo-Reactive (TR) robotic agent programs comprise sequences of guarded action rules clustered into named parameterised procedures. Their ancestry goes back to the first cognitive robot, Shakey. Like Shakey, a TR programmed robotic agent has a deductive *Belief Store* comprising constantly changing predicate logic *percept* facts, and *knowledge* facts and rules for querying the percepts. In this paper we introduce TR programming using a simple example expressed in the teleo-reactive programming language TeleoR, which is a syntactic extension of QuLog, a typed logic programming language used for the agent's *Belief Store*. The example is sufficient to illustrate key properties that a TR and a TeleoR program should have. We give formal definitions of the key properties, and an informal operational semantics of the evaluation of a TeleoR procedure call. We then formally express the key properties in LTL. Finally we show how their LTL formalisation can be used to prove key properties of TeleoR procedures by using the example TeleoR program.

1 Introduction

A Teleo-Reactive (TR) programmed robotic agent has a deductive *Belief Store* comprising constantly changing predicate logic *percept* facts generated from the latest sensor readings. The facts are updated using fresh sensor readings at an application dependent frequency, which may be every few milliseconds. The percept facts are the agent's current *beliefs* about the state of the robotic device or devices it controls, and the state of the physical environment in which the devices operate.

Augmenting these dynamic percept facts are fixed facts about the robotic devices and their environment, and Prolog style rules for relations disjoint from the percept relations. Together they allow higher level interpretations of subsets of the percept facts. The rules are its *knowledge*.

In this paper we introduce TR programming using a simple example expressed in the TR subset of TeleoR [1], which is a syntactic extension of QuLog[2], a typed logic programming language.

* Dongol is supported by EPSRC Grants EP/R019045/2, EP/R032556/1 and EP/R025134/2.

A TeleoR procedure named p comprises a parameterised *sequence* of *guarded action rules* rules of the form:

$\texttt{p(X}_1\texttt{,..,X}_k\texttt{)\{}$
$\quad \texttt{G}_1 \texttt{ \~{}> A}_1$
$\quad .$
$\quad .$
$\quad \texttt{G}_n \texttt{ \~{}> A}_n$
$\quad \}$

The $\texttt{G}_i$ are the QuLog guards, the $\texttt{A}_i$ the actions, and the parameters $\texttt{X}_1\texttt{,..,X}_k$ can appear in any guard or action.

Note that a rule may contain variables other than $\texttt{X}_1\texttt{,..,X}_k$. These are the local variables of the rule. All local variables in $\texttt{A}_i$ must also appear in $\texttt{G}_i$.

Each $\texttt{G}_i$ is a *Belief Store* query and each $\texttt{A}_i$ is either a call to a TeleoR procedure, including a recursive call, or it comprises one or more action calls for robotic devices.

When the p procedure is called, the parameter values must be *ground* (fully instantiated) values. These values partially instantiate the sequence of guarded rules of the procedure body, giving the modified rule sequence:

$\texttt{G}'_1 \texttt{ \~{}> A}'_1$
$\ldots$
$\texttt{G}'_i \texttt{ \~{}> A}'_i$
$\ldots$
$\texttt{G}'_j \texttt{ \~{}> A}'_j$
$\ldots$
$\texttt{G}'_n \texttt{ \~{}> A}'_n$

In before/after order, the partly instantiated guard queries are evaluated one by one against the current state of the agent's *Belief Store*. This is in order to find the *first* rule with a guard $\texttt{G}'_j$ that is inferable from the *Belief Store*. This typically further instantiates remaining variables in $\texttt{G}'_j$, and always results in a ground $\texttt{A}''_j$ action of the rule.

Rule j is *fired* and action $\texttt{A}''_j$ is executed. If it is a TeleoR procedure call the first fireable rule of this new call is found and, in turn, its fully instantiated action executed. Eventually a procedure will be called in which the fired rule for the current *Belief Store* has robotic device actions. They are dispatched to the robotic devices, and typically result in changes in the position of the robotic devices, and/or cause changes in their physical environment.

A two thread robotic agent architecture is depicted in Figure 1. The percepts handler atomically updates the *Belief Store* when a new batch of percept facts arrives from the sensor interpretation routines. These routines may be poled or send their percept interpretations of the sensor readings at fixed short intervals. We assume they are batched so that each new set of percepts gives a comprehensive view of the state of the robotic devices and their environment.

The TeleoR evaluator thread executes some initial call to a TeleoR procedure, this is the *task procedure call*. The existential closure of the partly instantiated

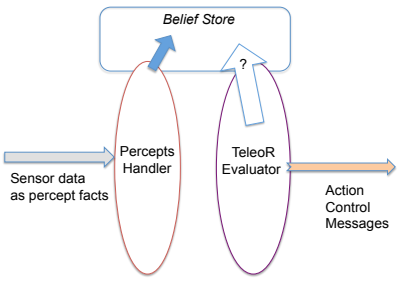

Fig. 1. Two Thread Agent Architecture

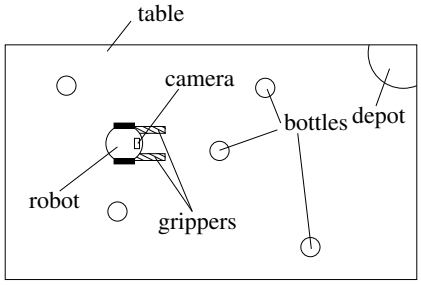

Fig. 2. Top-down view of bottle collecting robot

guard of the first rule of this procedure call is the *task goal*. This first rule usually has the empty action `()`.

After each percept thread update the task call is re-evaluated to determine if new robotic device actions need to be dispatched to the robotic devices. If the re-evaluation results in the same robotic actions as were determined on the last percepts update, these actions are allowed to continue.

However, the hope is that after a repetition of an initial sequence of the same procedure calls and rule firings as last time, eventually a repeated procedure call `PCall` will fire a rule *earlier* than the last rule that was fired for that call. Almost certainly the changed rule firing will result in different robotic actions being determined for the updated *Belief Store*.

Suppose rule `j` was fired last time in `PCall`. Suppose the robotic actions that were executed directly or indirectly as a result of firing this rule have caused their intended external changes resulting in new sensor readings mapped into different percept facts. For the agent's updated *Belief Store* an earlier rule `i` now becomes the first partly instantiated rule of `PCall` with an inferable guard.

`TeleoR` programs are written so that this normally, eventually happens for every procedure call. The existential closure of guard G'_j is viewed as a direct sub-goal of the existential closure of G'_i. More generally, the existential closure of every partly instantiated rule guard of a call, except that of the first rule, should be a *direct* sub-goal of the existential closure of the partly instantiated guard of an *earlier* rule in the call's rule sequence. In addition, the existential closure of every partly instantiated call guard should be an indirect sub-goal of the existential closure of the partly instantiated guard G'_1 of the first rule of the call.

Example 1 (An example `TeleoR` *program).* Here are two `TeleoR` procedures, with associated type definitions and type declarations. The goal of the first procedure is to get a mobile robot close to the object, `Th`, making use of independent `move` and `turn` actions, and a general `see` percept. We use the Prolog convention that variables begin with an upper case letter or underscore.

```
def thing ::= bottle | drop | ...
```

```
def dir ::= left | centre | right
def dist ::= close | near | far
percept see(thing,dist,dir)
durative move(num), turn(dir,num)

tel get_close_to(thing)
get_close_to(Th){
    see(Th,close,_) ~> ()
    see(Th,near,_) ~>  approach_until(close,Th,3.0,1.0)
    see(Th,far,_) ~> approach_until(near,Th,4.5,0.5)
    true ~> turn(right,0.5)
    }
tel approach_until(dist,thing,num,num)
approach_until(Dist,Th,Fs,Ts){
    see(Th,Dist,_) ~> () % Th being approached is now Dist away
    see(Th,_,centre) ~> move(Fs)
    see(Th,_,Dir) ~> move(Fs),turn(Dir,Ts)
    % Dir is left or right. move forward turning Dir to bring back into centre view.
    }
```

The underscores in the `see` conditions of the first procedure, and the first rule of the second procedure, indicate that the orientation of the seen `Th` is not relevant for the action of the rule. Those in the last two rules of the second procedure indicate that the distance is not relevant.

An example of a percept fact is `see(bottle,near,centre)`, reporting that the analysis of the image from a forward pointing camera of a mobile robot has determined that a bottle of known size is near to the camera, more or less in centre view.

`see` gives a qualitative measure of its distance from the robot's on-board camera, as `close`, `near` or `far`, and indicates whether the seen `thing` is within, or to the left or right of a central area of the camera's field of view.

An example of a primitive action is `move(3.0)` which causes both drive wheels of a mobile robot to turn at the same speed so that normally the robot will move forward more or less in a straight line at the specified speed of `3.0` centimetres per second.

Suppose a task is started with a call `get_close_to(bottle)`. Before the call let us assume the robot is stationary near to and facing a bottle. Analysis of the image of its forward pointing camera has resulted in percept `see(bottle,near,centre)` being in the agent's *Belief Store*.

The second rule of call `get_close_to(bottle)` will be fired with action the procedure call `approach_until(close,bottle,3.0,1.0)`. The second rule of this auxiliary procedure call will be fired because its guard `see(bottle,_,centre)` matches the percept `see(bottle,near,centre)`. The rule's action will be `move(3.0)` . The control action `start(move(3.0))` will be sent to the mobile robot.

The durative action `move(3.0)` should *normally, eventually* result in the mobile robot getting close to the bottle. Then, analysis of the robot's camera image will result in a percept `see(bottle,close,centre)` being received by the agent, replacing `see(bottle,near,centre)` in its *Belief Store*. When this happens the first rule of the call `get_close_to(bottle)` with action `()` will be fired and control action `stop(move(3.0))` sent to the robot. The goal of the task call

will has been achieved. ∃`(Dir)see(Th,near,Dir)` is an immediately sub-goal of ∃`(Dir)see(Th,close,Dir)`

However, before the action of getting near to the seen `bottle` is achieved, the `bottle` may be moved by an interfering person to be `far` from the robot but still in view. Suppose that immediately after the move percept `see(bottle,far,right)` is received, replacing `see(bottle,near,centre)`. Re-evaluation of the task call will result in its rule 4 being fired with action `approach_until(near,bottle,4.5,0.5)`.

The percept fact `see(bottle,far,right)` is still in the *Belief Store*, the last rule of the new `approach_until` call will be fired and control actions

`start(turn(0.5,right)) mod(move(3.0),move(4.5))`

are sent to the robot. The result is that the robot will move forward more quickly at a speed of 4.5, with a correctional turn to the right at speed 0.5. This is in order to get `near` to the bottle again, and hopefully to bring it into centre view.

The aim of the paper is to enable reasoning about `TeleoR` procedures in a modular manner. Our contributions are: **(1)** an adaptation of linear temporal logic for `TeleoR` procedure call evaluations based on action sequences; **(2)** technique of decoupling environment assumptions into action, physics, and sensor assumptions; and **(3)** an application of the logic to verify properties of the bottle collecting robot. The paper builds on existing work on logic programming [7] and robot behavioural programming [5,8].

2 Key properties of `TeleoR` procedures

Sub-goal structure and regression The existential closures of the partially instantiated guards, in which each local variable is existentially quantified, lie on an implicit sub-goal tree rooted at the existential closure of the guard of the first rule.

Definition 2. *An action* $\mathtt{A}_j$ *satisfies* regression *iff whenever it is started when its guard is the first inferable guard, and continued whilst this is the case, it will eventually result in progression up the sub-goal tree of guards.*

That is, eventually the guard of an earlier rule $\mathtt{G}_i$, $i < j$, should become the first inferable guard. Nilsson calls $\mathtt{G}_j$ the *regression* of $\mathtt{G}_i$ through $\mathtt{A}_j$.

Guarantee of ground actions When a guard query is evaluated against the *Belief Store* using a Prolog style evaluation its local variables will be given values if the evaluation succeeds. In `TeleoR`, in which each percept is typed, and each rule relation is both typed and moded, compile time analysis of each guarded rule ensures that should the guard query succeed, the rule's action will be ground with arguments of the required type. The primary modes are `!`, indicating that a relation argument must be ground when the relation is queried, and `?`, indicating an argument may not be given as an un-ground term, even as a variable, when the relation is queried but that the given argument will be instantiated to a ground term if the query succeeds. The `QuLog` compiler checks all the mode declarations

and checks that each variable of a `TeleoR` rule appearing in an action term is either a procedure parameter or it appears in the rule guard in a `?` argument position of some relation query. This analysis not only ensures that a `TeleoR` procedure call action is always ground with arguments of their declared type or sub-type, it also ensures any primitive robotic actions dispatched to robot devices will also be fully instantiated and type correct.

Covering all eventualities. The partially instantiated guards of a procedure call should also be such that for every *Belief Store* state in which the call may be active there will be at least one inferable guard. Nilsson calls this the *completeness* property of a procedure. This property holds for both our example procedures. For the first it trivially holds since the last rule will always be fired if no earlier rule can be fired. It holds for the second procedure given the two guard contexts from which it is called in the first procedure, both of which require a `see` percept to be in the *Belief Store* while the call is active.

Smooth transitions between primitive actions. When each new batch of percepts arrives, perhaps via a ROS [9] interface, this process of finding and firing the first rule of each call with an inferable guard is restarted. This is in order to determine, as quickly as possible, the appropriate tuple of robotic actions response to the new percepts. Thus,

1. actions that were in the last tuple of actions are allowed to continue, perhaps modified
2. other actions of the last tuple are stopped
3. new actions are started.

For example, if the last tuple of actions was `move(4.5), turn(left,0.5)` and the new tuple is just `move(3)`, the `turn` action is stopped and the `move` action argument is modified to `3`.

Elasticity of complete procedure programs. This reactive operational semantics means that each `TeleoR` procedure is not only a universal conditional plan for its call goals, it is also a plan that recovers from setbacks and immediately responds to opportunities. If, after a *Belief Store* update a higher rule of some call *PCall* can unexpectedly be fired, perhaps because of a helping exogenous event, that rule will be fired *jumping* upwards in the task's sub-goal tree. If instead a lower rule of *PCall* must be fired, a detected unexpected result of some robotic action, or a detected result of a interfering exogenous event, this means that the climb up the sub-goal tree of *PCall*'s rule guards must be re-attempted from a different sub-goal of its call goal. There has been a setback in the progress towards the task goal but the recovery response action should *normally and eventually* result in its being achieved.

There is a scenario in which the task goal will never be achieved. This will happen if, whenever the robot is about to get close to a bottle, the bottle is either moved out of sight or further away. The robot will doggedly chase the bottle until its battery runs out.

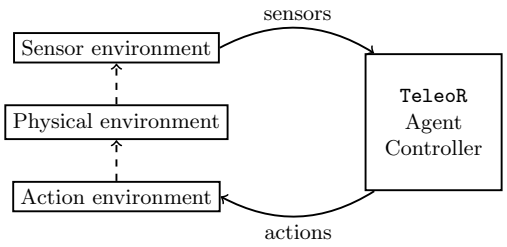

Fig. 3. Decomposed environment and `TeleoR` controller

From deliberation to reaction. Although not the case for our example procedures, typically, initially called `TeleoR` procedures query the percept facts through several levels of defined relations. Via procedure call actions, they eventually call a `TeleoR` procedure that directly queries the percept facts and mostly has non-procedure call actions. So, for `TeleoR` and `TeleoR` the interface between deliberation about what sub-plans to invoke to achieve a task goal, to the invoking of a sensor reactive behaviour to directly control robotic resources, is a sequence of procedure calls.

3 Temporal Logic Semantics

The core of the semantics is that it decouples the controller (agent program) from the environment (see Figure 3). This is achieved using two coupling relations that describe how closely the internal state of an agent matches the real world. Any actions that a robot performs are interpreted through the coupling relation that in turn produces some physical change. In turn, the physical changes are sampled by the robot to potentially update the *Belief Store*.

The *agent controller* is defined with respect to the agent's *Belief Store*, whereas the *physical environment* describes how object properties change in the real world as a result of robot actions. We typically do not model the actual physics of any component (e.g., via differential equations). Instead, the behaviour of the physical world is formalised as abstract assertions. This section focuses on the agent controller; we formalise its interaction with the environment in Section 4.

Our formal model for the agent controller assumes that each agent generates traces of the form:

$$\beta_0\, \beta_1\, \beta_2\, \beta_3\, \ldots$$

where each β_i is a *Belief Store* that describes the agent's view of the world. For any of these traces, it is possible to interpret standard LTL formulae, with predicates that support beliefs and intentions. In particular, we admit formulae of the form

$$\varphi ::= P \mid \neg\varphi \mid \varphi_1 \oplus \varphi_2 \mid \mathbf{X}\varphi \mid \mathbf{G}\varphi \mid \mathbf{F}\varphi \mid \varphi_1 \mathbf{U} \varphi_2 \mid \varphi_1 \mathbf{W} \varphi_2$$

where P is a *Belief Store* predicate and **X**, **G**, **F**, **U** and **W** denote the standard next, globally, finally, until and unless modalities of linear temporal logic and $\oplus$ is a binary boolean operator. Examples of belief store states over which P is evaluated are given in Section 4; the traces these induce are made more precise in 3. In particular, it is worth noting that they may include first-order quantifiers.

In order to give the semantics of TR procedures we need to keep track of which actions (both TR procedure calls and primitive actions) are currently being executed and which guards are currently true (and what instantiations of variables make a given guard true). To simplify the semantics we assume the *Belief Store* contains facts of the form $\mathsf{do}\, A$ to indicate that the action A is currently executing. For a given goal G, we write $\mathsf{bel}\, G\theta$, where θ is a substitution (instantiation of variables in G), that makes $G\theta$ the first inferable instance of G.

Each ground instance, C, of a TR procedure call is defined as a sequence of guarded actions instances $\mathtt{G}_1$ `~>` $\mathtt{A}_1, \mathtt{G}_2$ `~>` $\mathtt{A}_2, \ldots, \mathtt{G}_n$ `~>` $\mathtt{A}_n$. The following LTL formula gives the semantics of the execution of C.

$$\mathbf{G}[\forall 1 \leq i \leq n, \theta.\ (\mathsf{do}\, C \wedge \mathsf{bel}\, G_i\theta \wedge (\forall 1 \leq j < i.\ \neg\exists \theta'.\ \mathsf{bel}\, G_j\theta') \Rightarrow \mathsf{do}\, A_i\theta)] \qquad (Action)$$

Informally, if we are executing C and the i'th rule is the active rule (because its guard is believed to be true but no earlier guard is) then we will be executing the action A_i with the instantiation of all of the variables in A_i determined by the corresponding guard. Note that we only need to consider ground instances of TR procedures as the top-level TR procedure that we call will be ground and the θ in the above formula will ground the action (which may be another TR procedure call).

In general, because TR programs allow hierarchical nesting, we typically need to repeatedly apply the above formula to determine what primitive actions will be executing for a given *Belief Store*. For example, if we are executing `get_close_to(bottle)` and the *Belief Store* contains only the belief `see(bottle, far, left)` then applying (*Action*) for `get_close_to(bottle)` will tell us that `approach_until(near, bottle, 4.5, 0.5)` will be executing. Applying (*Action*) to this procedure then tells us that the primitive actions `move(4.5),turn(left, 0.5)` will be executing.

4 `TeleoR` Environment

As depicted in Figure 3, the `TeleoR` controller only forms part of the overall system. More work is required to ensure that (1) actions have an effect on the real-time environment; (2) that the action's effects in the environment bring about some physical changes; and (3) these physical changes are sensed by the agent. This inter-dependency between a controller and its environment has long been studied in the control systems and cyber-physical domains. We treat the three requirements above as environment assumptions and express the assumptions as abstractly as possible. We model separate assumptions that cover actions (outputs), physics, and sensors (inputs). In the context of autonomous systems, a

decoupled approach has recently been proposed by Kamali et al [6], who describe a separation of concerns between physical and discrete assumptions.

We describe environment assumptions in Section 4.1, which we formalise in terms of `TeleoR` traces in Section 4.2. An example verification is provided in Section 4.3.

4.1 Environment assumptions

Consider, for example, a call to the `turn` action, which spins the robot with value `0.5`. We require that this action effectively rotates the robot at a velocity of $0.5\pi/s$. This is captured, more generally, by an action *assumption* of the form

$$\forall x.\ \texttt{do}\,\texttt{turn}(\texttt{right},x) \vdash \mathit{rot\_spd} = x \tag{1}$$

linking the intended action to the actual action, where *rot_spd* is an *environment* variable recording the actual physical rotational speed of the robot. On the other hand, $\texttt{do}\,\texttt{turn}(\texttt{right},x)$ is a belief store assertion, stating that the robot believes it is executing $\texttt{turn}(\texttt{right})$ with a speed of x. Of course, one might have environments in which the rotational speed is less accurate (e.g., due to slippage of the wheels). Such phenomena can be modelled by an equation of the form $\forall x.\ \texttt{do}\,\texttt{turn}(\texttt{right},x) \vdash \mathit{rot\_spd} = x \pm x * 0.1$ which states that the `turn` action translates to an actual rotational velocity within some tolerance bounds.

Recall that the semantics of a `TeleoR` procedure guarantees that actions are not executed unless explicitly fired by a program, and that it is possible to perform more than one action in parallel. For example, in Example 1, the second branch of `approach_until` only executes `move` (and does not execute `turn`). On the other hand, the third branch executes both `move` and `turn` together. Action assumptions must therefore also link non-execution of actions to environment effects. One property that we will use is, where *fwd_spd* is a physical variable that models the forward speed of the robot.

$$\neg(\exists x.\ \texttt{do}\,\texttt{move}(x)) \vdash \mathit{fwd\_spd} = 0 \tag{2}$$

Thus, by (2), if `move` is not being executed, then the robot is not moving forward. In Example 1, assumptions (1) and (2) together ensure that executing the last branch of `get_close_to` causes the robot to spin on its axis.

Note that there is a difference between (2) and the predicate $\forall x.\ \texttt{do}\,\texttt{move}(x) \vdash \mathit{fwd\_spd} = x$, which for $x = 0$ gives

$$\texttt{do}\,\texttt{move}(0) \vdash \mathit{fwd\_spd} = 0\ . \tag{3}$$

The antecedent of (2) states that there is no percept fact $\texttt{do}\,\texttt{move}(x)$ recorded in the belief store, for any value of x, whereas the antecedent of (3) states that the value recorded for $\texttt{do}\,\texttt{move}(\_)$ is $\texttt{do}\,\texttt{move}(0)$. In other words, for (2), we assume the *Belief Store* query $\texttt{do}\,\texttt{move}(X)$ returns false, whereas in (3), the *Belief Store* query returns a value 0 for X. In both cases, the (physical) value of *fwd_spd* is required to be 0.

Given that there is a stationary bottle on the table, we require that the `turn` action is such that it eventually causes the bottle to be seen. This first of all requires a physical assumption, that turning on its axis at a rate of $rot\_spd = 0.5$ is adequate for the robot to be pointing towards a bottle. We state a bottle being in front of the robot abstractly using a predicate *bot_in_front*, which we assume holds precisely when there is some bottle in front of the robot. An implementation may guarantee *bot_in_front* in more than one way, e.g., as with *rot_spd* above, may be within certain tolerance bounds calculated using the angle of vision of the robot's camera, the robot's current position and the bottle's current position. Thus *bot_in_front* may mean that a bottle is directly in front of the robot, or slightly to the right or left of centre. Such details can be described by the logic, but are ignored for the purposes of this paper.

Formally, we assume that the environment ensures the following.

$$\mathbf{G}(((rot\_spd = 0.5 \wedge fwd\_spd = 0)\mathbf{W}\, bot\_in\_front) \Rightarrow \mathbf{F}\, bot\_in\_front) \quad (4)$$

The antecedent, i.e., $(rot\_spd = 0.5 \wedge fwd\_spd = 0)\mathbf{W}\, bot\_in\_front$, states that the robot continues to rotate on its axis unless there is a bottle in front of the robot. This alone does not guarantee progress (i.e., that *bot_in_front* holds). However, the consequent of (4) ensures that a bottle will eventually be seen.

We note that (4) is an abstract property that encompasses a number of different scenarios. For example, it guarantees that if there is only one bottle on the table, then this bottle will eventually be seen, provided the robot continues to rotate at a velocity of $0.5\pi/s$. This means that the bottle is not moved away by the environment (unless another bottle is placed on the table as a replacement!) If there are multiple bottles on the table, then the environment guarantees that the robot will eventually rotate towards at least one of these (e.g., the bottles will not all be removed from the table). Condition (4) also guarantees that if there are no bottles on the table, then the environment eventually places at least one bottle on the table, and given that the robot continues to spin on its axis, then some bottle will be in front of the robot.

Finally, we require that a *bot_in_front* predicate triggers a new *Belief Store* update. In particular, if there is a bottle in front of the robot, then `see(bottle, _, _)` must become true in the *Belief Store*. This is formalised by the assertions

$$bot\_in\_front \vdash \exists x, y.\, \mathsf{bel}\, \mathtt{see}(\mathtt{bottle}, x, y) \quad (5)$$

$$\neg bot\_in\_front \vdash \neg\exists x, y.\, \mathsf{bel}\, \mathtt{see}(\mathtt{bottle}, x, y) \quad (6)$$

Again, there are different levels of detail one can apply in modelling reality when making sensor assumptions, e.g., sensor inaccuracies, timing delays etc.

Putting these together, from (1), we have that the `turn` action with value `0.5` induces a physical rotation; from (4), we have that a physical rotation induces that the bottle is in front of the robot; and from (5) the fact that the bottle is in front of the robot can be sensed. Moreover, this guarantees regression, i.e., that `turn` causes the higher priority guard `see` to become true.

4.2 TeleoR traces

We now provide a formalisation of the ideas above for a given TeleoR program (and its environment). First, we define an *environment state* to be a function mapping from (physical) variables to values. We let *Init* be the set of all possible initial environment states. A *sensor assertion* is a predicate of the form $E \vdash P$, where E is a ground predicate on the environment state and P is a ground predicate on the belief store. Given an environment state ϵ and belief store β, we say $E \vdash P$ holds in (ϵ, β), denoted $E \vdash_{\epsilon,\beta} \mathsf{bel}\, P$ iff $(\epsilon \models E) \Rightarrow (\beta \models P)$. Similarly, an *action assertion* is a predicate of the form $\mathsf{do}\, \mathtt{A}_i \vdash E$, where $\mathtt{A}_i$ is an ground action. We say $\mathsf{do}\, \mathtt{A}_i \vdash E$ holds in (β, ϵ), denoted $\mathsf{do}\, \mathtt{A}_i \vdash_{\beta,\epsilon} E$ iff $(\beta \models \mathsf{do}\, \mathtt{A}_i) \Rightarrow (\epsilon \models E)$.

A TeleoR *state* is triple $(\epsilon, \beta, \epsilon')$, where ϵ is the environment being sensed, β is the belief store that results from sensing ϵ, which includes the firing of a new action, and ϵ' is the new environment that results from firing this action.

Consistency of a TeleoR state is judged with respect to the set of sensor and action assertions that are assumed for the program. In particular, given a set of sensor assertions S and a set of action assertions Z, we say the TeleoR state $(\epsilon, \beta, \epsilon')$ is *consistent* with respect to S and Z iff

1. for each $(E \vdash P) \in S$, we have $E \vdash_{\epsilon,\beta} P$,
2. for each $(\mathsf{do}\, \mathtt{A}_i \vdash E) \in Z$, we have $\mathsf{do}\, \mathtt{A}_i \vdash_{\beta,\epsilon'} E$, and
3. ϵ and ϵ' are identical except for the environment variables that are changed as a result of the updated action in β.

Definition 3. *A* TeleoR trace *is a sequence*

$$\tau = (\epsilon_0, \beta_0, \epsilon'_0), (\epsilon_1, \beta_1, \epsilon'_1), (\epsilon_2, \beta_2, \epsilon'_2), \ldots$$

of TeleoR states *such that* $\epsilon_0 \in$ *Init, and each* $(\epsilon_i, \beta_i, \epsilon'_i)$ *is consistent.*

We define two projection functions π_p and π_e that restrict a given TeleoR trace to the belief stores and environment states, respectively. For τ above:

$$\pi_p(\tau) = \beta_0, \beta_1, \beta_2, \beta_3, \ldots \qquad \pi_e(\tau) = \epsilon_0, \epsilon'_0, \epsilon_1, \epsilon'_1, \epsilon_2, \epsilon'_2, \ldots$$

Note that above we are considering the TeleoR trace obtained from the execution of a top-level ground TR procedure call C and so for each β_i in $\pi_p(\tau)$ we have a corresponding set of primitive actions which are determined by repeated uses of (*Action*).

Finally, we must introduce *system assumptions*, which are assumptions over TeleoR traces. Such assumptions can be used to state that environment variables under the control of the robot are not arbitrarily modified by the environment. System assumptions are formalised as temporal formulae over TeleoR traces. This requires that we define predicates over TeleoR states (of the form $(\epsilon, \beta, \epsilon')$); LTL operators over TeleoR traces can be readily defined. We use unprimed and primed variables to distinguish environment variables in ϵ and ϵ', respectively.

4.3 Example verification

For our running example, we use system assumptions to ensure that the environment does not impede a robot's (physical) rotation. In particular, we require that for each consecutive pair of `TeleoR` states in τ, i.e., τ_i, τ_{i+1}, the value of *rot_spd* in the post-environment state of τ_i is the same as the value of *rot_spd* in the pre-environment state of τ_{i+1}. This is formalised as:

$$\forall k, l.\ \mathbf{G}(rot\_spd' = k \wedge fwd\_spd' = l \Rightarrow \mathbf{X}(rot\_spd = k \wedge fwd\_spd = l)) \quad (7)$$

We now return to the the program instance `get_close_to(bottle)` in Example 1 and describe how it can be shown to satisfy the regression property when it is executing the lowest priority action `turn(right, 0.5)`. That is, a bottle will be seen. Suppose

$$\tau = (\epsilon_0, \beta_0, \epsilon'_0), (\epsilon_1, \beta_1, \epsilon'_1), (\epsilon_2, \beta_2, \epsilon'_2), \ldots$$

is a `TeleoR` trace of the program.

Consider an arbitrary index i. Assume for each belief store up to and including β_i we have $\mathsf{bel}\, \neg\exists x, y.\mathtt{see}(\mathtt{bottle}, x, y)$. From this assumption, using equation (*Action*), we obtain

$$\beta_i \models \mathsf{do}\, \mathtt{turn}(\mathtt{right}, 0.5) \wedge \neg\exists x.\, \mathsf{do}\, \mathtt{move}(x).$$

Now, by assumptions (1) and (2) we have $\epsilon'_i \models (rot\_spd = 0.5 \wedge fwd\_spd = 0)$.

We now argue as follows by case analysis.

- If for some $j > i$, we have $\tau_j \models bot\_in\_front$, then we trivially have regression, since condition (5) ensures that this triggers the required belief store update in τ_j.
- So suppose that for all $j > i$, $\tau_j \models \neg bot\_in\_front$. Then by (6), for all $j > i$, we have

  $$\beta_j \models \neg\exists x, y.\mathtt{sees}(\mathtt{bottle}, x, y).$$

  This also means (by the semantics of the program) that for all each $\beta_j \models \mathsf{do}\, \mathtt{turn}(\mathtt{right}, 0.5)$. Hence, by (1) and (2), for each $j > i$, we have

  $$\epsilon'_j \models (rot\_spd = 0.5 \wedge fwd\_spd = 0).$$

  Now, since we also have $\epsilon'_i \models (rot\_spd = 0.5 \wedge fwd\_spd = 0)$, by (7), we have that for all $j > i$, $\epsilon_j \models (rot\_spd = 0.5 \wedge fwd\_spd = 0)$. Thus, the antecedent of (4) is satisfied, and hence we must have $\tau_j \models bot\_in\_front$ for some j, which contradicts our initial assumption.

This completes the proof and gives us the required regression result. That is, under reasonable assumptions about the environment, if the last rule of `get_close_to(bottle)` is chosen (because no bottle is seen) then the action of turning will make a bottle visible and hence an earlier rule will be chosen.

5 Conclusions

This paper develops an adaptation of LTL to reason about `TeleoR` programs that enables reasoning over belief stores in a decoupled manner. Actions (having an effect on the environment) and sensors (taking readings from the environment) are modelled as assumptions linking the belief store and the environment, which are subsequently used in system proofs. The logic also enables modelling of purely physical assumptions and those pertaining to the system as a whole in a straightforward manner. We apply this logic to show a regression property of a `TeleoR` program.

This separation of concerns enables less idealised sensor information (inputs) and robotic movements (outputs) to be modelled more easily. Such information may also be provided by domain experts, and further refined at different stages of development. Of course, to cater for a wider range of implementations (without requiring proofs to be replayed), one must use specifications that are as abstract as possible. Precisely what this entails, however, is a subject of further study.

Our ultimate aim for this work is to encode the framework in a verification tool withing a theorem proving environment. Hence, we do not consider questions such as decidability of the logic. However, for several of the example programs we have developed, it is possible to develop controllers that operate over discretised approximations that result in finitely many possibilities of the belief store. For example, for our bottle collecting robot, it is possible to record values `high` and `low` values instead of precise speeds, and record `far`, `near` and `very near` instead of precise distances.

Prior work on `TeleoR` has focussed on addressing real-time properties [3,4] using an interval temporal logic. The focus there has been to cope with timing issues, including those that lead to sampling errors. Although the formalism enables separation of properties into those of the environment (formalised by a rely) and those of the agent (formalised by a guarantee), both are asserted over a monolithic state containing all system variables. In contrast, this paper presents a separation of concerns, whereby the inputs and outputs to the `TeleoR` program are linked in a separate step.

References

1. Clark, K.L., Robinson, P.J.: Robotic Agent Programming in TeleoR. In: Proceedings of International Conference of Robotics and Automation. IEEE (2015)
2. Clark, K.L., Robinson, P.J.: Chapter 3: Introduction to QuLog. In: Programming Communicating Robotic Agents: A Multi-tasking Teleo-Reactive Approach. Springer (2020), to appear
3. Dongol, B., Hayes, I.J.: Rely/guarantee reasoning for teleo-reactive programs over multiple time bands. In: IFM. Lecture Notes in Computer Science, vol. 7321, pp. 39–53. Springer (2012)
4. Dongol, B., Hayes, I.J., Robinson, P.J.: Reasoning about goal-directed real-time teleo-reactive programs. Formal Asp. Comput. **26**(3), 563–589 (2014)
5. Jones, J., Roth, D.: Robot programming: a practical guide to behavior-based robotics. McGraw-Hill (2004)

6. Kamali, M., Linker, S., Fisher, M.: Modular verification of vehicle platooning with respect to decisions, space and time. In: FTSCS. Communications in Computer and Information Science, vol. 1008, pp. 18–36. Springer (2018)
7. Levesque, H.: Thinking as Computation. MIT Press (2012)
8. Mataric, M.J.: The Robotics Primer. MIT Press (2007)
9. Quigley, M., Gerkey, B., Conley, K., Faust, J., Foote, T., Leibs, J., Berger, E., Wheeler, R., Ng, A.: ROS: an open-source Robot Operating System (2009), at:www.robotics.stanford.edu/∼ang/papers/icraoss09-ROS.pdf