

Towards Reasoning About Teleo-Reactive Programs for Robust Real-Time Systems

Ian J. Hayes

School of Information Technology and Electrical Engineering
The University of Queensland
Brisbane, 4072, Australia
ianh@itee.uq.edu.au

ABSTRACT

The teleo-reactive programming approach was developed by Nilsson for application in domains like robotics. It has a high-level programming model that allows real-time control programs to be written in a manner that allows them to react robustly to changes in the environment. In this paper we give a formalisation of the semantics of teleo-reactive programs and provide rely/guarantee rules for reasoning about them. The semantics are given in a form that partitions the behaviour of the system into its behaviour over a sequence of time intervals.

1. INTRODUCTION

Real-time computer systems are being employed to control both mission-critical systems, where failures can be costly, and safety-critical systems, where failure can endanger lives, for example, in robotics and automated railway systems. The development of such systems needs to support the verification and validation of their dependability in a rigorous manner [19, 25]. Because of the growing sophistication of the applications in which real-time control systems are used, we need more sophisticated languages to express the control programs. Teleo-reactive programming notation is an excellent candidate.¹

Our aim is to extend the theory of teleo-reactive programs to provide better methods for developing and reasoning about the future generations of advanced real-time control applications, in a manner that provides the dependability required for use in critical applications. Teleo-Reactive (T-R) programs were invented by Nils Nilsson from Stanford University [22, 24]. To quote Nilsson [23]

A teleo-reactive (T-R) program is a mid-level agent control program that robustly directs an agent toward a goal in a manner that continuously takes into account the agent's changing perceptions of a dynamic environment. T-R programs are written in a production-rule-like language.

The main advantage of teleo-reactive programs is their ability to react robustly to changes in either their environment or their current

¹Greek: telos: end, purpose.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

SERENE 2008, November 17-19, 2008, Newcastle, UK.
Copyright 2008 ACM 978-1-60558-275-7/08/11...\$5.00

goal. On the web site for teleo-reactive programming [23] there is a sample block-stacking program, that given a requirement to stack blocks on top of one another in a particular order, carries out that task. The program is resilient to interference from the environment that can hinder the program by unstacking already stacked blocks, or help the program by stacking blocks in the correct order.

Because teleo-reactive programs allow a hierarchical nesting of procedures, they provide a simpler, higher-level model than, say, Action Systems [4, 3], or TLA [18], for expressing a mid-level control program. In addition, actions in teleo-reactive programs are durative (i.e., they take time) and hence it is easy to combine teleo-reactive programs with continuous components. Here we need to compare them with approaches like Continuous Action Systems [2] and Hybrid Automata [1, 15].

This paper aims to provide a reasoning framework to support the development and verification of teleo-reactive programs. We

- give an example teleo-reactive program to illustrate the notation and compare it with other approaches (Section 2);
- extend the notation with specification constructs including non-determinism (Section 3);
- develop a time-interval semantics for teleo-reactive programs (Section 4);
- develop rely/guarantee reasoning rules to allow the proof of desired properties of teleo-reactive programs in a compositional manner (Section 5); and
- apply the rules to the example program (Section 6).

2. EXAMPLE

To familiarise the reader with teleo-reactive programs, we give a short overview and a simple example. Teleo-reactive programs are described by a combination of

- sensed values, or values derived or inferred from sensed values,
- primitive durative actions, i.e., actions that take time, and
- processes defined via (durative) conditionals.

The example we provide is of a teleo-reactive program that controls the water level in a coal mine [8, 21, 17]. There are three sensed values:

methane, the level of methane in the mine;

water, the level of water in the mine; and

$pump_active$, an indicator of whether the pump is currently active.

When there is water in the mine shaft it should be pumped out, unless the methane level in the mine is above a critical level, in which case running the pump could cause an explosion. To avoid rapid cycling of the pump on and off, we use two water levels High and Low, where $Low < High$. When the water level reaches High the pump should be turned on and remain on until it reaches Low, at which stage it will remain off until it reaches High again. There are three primitive durative actions:

- **alarm**, that, while active, sounds an alarm;
- **pump**, that, while active, pumps water out of the mine; and
- **nil**, that does nothing.

The teleo-reactive program is represented by the two procedures `mine_pump` and `operate`, which are expressed as prioritised lists of conditions and associated actions. If a procedure is active, then at any time the action corresponding to the first true condition will be executing. For the procedure `mine_pump`, whenever the methane level is above the critical level, the alarm will sound, otherwise the procedure `operate` will be used, which will pump water out of the mine when necessary.

$$\begin{aligned} \text{mine_pump} &\hat{=} \\ &\begin{array}{l} \text{Critical} \leq \text{methane} \rightarrow \text{alarm,} \\ \text{true} \rightarrow \text{operate} \end{array} \\ \text{operate} &\hat{=} \\ &\begin{array}{l} \left(\begin{array}{l} \text{High} < \text{water} \vee \\ \text{Low} < \text{water} \wedge \text{pump_active} \end{array} \right) \rightarrow \text{pump,} \\ \text{true} \rightarrow \text{nil} \end{array} \end{aligned}$$

The conditions of the branches are continually monitored, so that as soon as an active branch's condition becomes false or the condition of an earlier (higher priority) branch becomes true, a switch of branch will occur. For example, if at any time while the procedure `operate` is executing, the methane level goes critical, the condition of the first branch of the procedure `mine_pump` will become true and execution of procedure `operate` will be terminated immediately. Note that procedure `operate` has no control over when its execution will be terminated: its termination is under the control of procedure `mine_pump`.

When the primitive action `pump` is active it guarantees that the water is extracted from the mine at a minimum rate of $MinOut$ and that the boolean $pump_active$ is true, i.e.,

$$g_pump \hat{=} \square(\text{MinOut} \leq \text{water_out} \wedge \text{pump_active})$$

where the operator \square (which is defined below) indicates that the predicate after it holds at all times over the interval that the pump is active. Action `pump` only needs to guarantee the above condition provided the water level is above Low and methane level is not critical, that is, when active, `pump` relies on the following condition

$$r_pump \hat{=} \square(\text{Low} < \text{water} \wedge \text{methane} < \text{Critical}).$$

We will make use of rely and guarantee conditions like r_pump and g_pump for reasoning about teleo-reactive programs in Sections 5 and 6.

The definition of `mine_pump` above is equivalent to the version in Figure 1, in which procedure `operate` has been expanded out, but with the condition for invoking `operate` (in this case just $true$) conjoined to each branch of `operate`. From this it is easy

$$\begin{array}{l} \text{Critical} \leq \text{methane} \rightarrow \text{alarm,} \\ \text{true} \wedge \left(\begin{array}{l} \text{High} < \text{water} \vee \\ \text{Low} < \text{water} \wedge \text{pump_active} \end{array} \right) \rightarrow \text{pump,} \\ \text{true} \wedge \text{true} \rightarrow \text{nil} \end{array}$$

Figure 1: Expanded teleo-reactive program

to see that the pump is only ever active while the methane level is not critical and the water level is at least above Low. Using other formalisms deriving such a property is no where near as easy. In other approaches either, if a procedure is invoked, it then becomes responsible for testing all conditions that may cause it to terminate, or, in the case of action systems, there isn't really a concept of a nested procedure. The hierarchical nesting of procedures afforded by teleo-reactive programs is an important difference, because it is easier to write programs that are required to react robustly to an unpredictable changing environment. Statechart notation [12] provides a similar ability in the sense that it supports hierarchical nesting of states and a transition at a higher level takes precedence over lower-level transitions, however, Statechart transitions are based on events, rather than continual monitoring of the environment.

We have provided only a simple example of a teleo-reactive program, but they are capable of representing quite sophisticated mid-level control systems.

2.1 Comparison with action systems

For comparison purposes we give a version of the mine pump system using action systems [4, 3]. Although there is some similarity between the syntax for guarded actions and that for teleo-reactive programs, the interpretation is quite different because the actions aren't durative. The action system initialises the variables $pump$ and $alarm$, and then repeatedly chooses a guarded action and executes it. The action system is designed so that there is always one guard which is true, even if the corresponding action is the null operation, `skip`, which does nothing and terminates immediately. The other actions are defined in terms of concurrent assignments to $pump$ and/or $alarm$. The sensors for the water and methane levels are assumed to be sampled once at the start of each iteration, so that consistent values are used throughout the guard evaluation.

```

initially alarm := Off || pump := Off;
do alarm = Off ^ methane < Critical ^
  High <= water ^ pump = Off -> pump := On
|| alarm = Off ^ methane < Critical ^
  water <= Low ^ pump = On -> pump := Off
|| alarm = Off ^ methane < Critical ^
  Low < water < High -> skip
|| alarm = Off ^ Critical <= methane ->
  alarm := On || pump := Off
|| alarm = On ^ Critical <= methane ->
  skip
|| alarm = On ^ methane < Critical ->
  alarm := Off
od

```

This action system has some interesting invariants.

$$\begin{aligned} \text{alarm} = \text{On} &\implies \text{pump} = \text{Off} \\ \text{alarm} = \text{On} &\iff \text{Critical} \leq \text{methane} \\ \text{pump} = \text{On} &\implies \text{Low} < \text{water} \end{aligned}$$

In the second and third invariants, *methane* and *water* refer to the values sampled at the beginning of the iteration, rather than the actual values of *methane* and *water*, which change over time. While these invariants can be shown to hold for the action system, the corresponding properties are obvious in the teleo-reactive program in Figure 1.

In order to express properties of continuous variables, one needs to switch to Continuous Action Systems [2]. These are similar to action systems, but allow the assignment of traces of future values to variables in order to model values that vary over time. These allow the expression of the fact that, when the pump is turned on, *water_out* is at least *MinOut*, but there doesn't seem to be any way to make assumptions about the behaviour of environment variables, like *water_in*. To give a general characterisation, when the focus is on the behaviour over intervals or responding to changes in the environment, teleo-reactive systems appear simpler, but when the focus is on the state changes or sequencing of actions (a state machine), action systems appear simpler.

3. ABSTRACT SYNTAX

To allow reasoning about teleo-reactive programs we first extend the notation with specification constructs:

- nondeterminism, and
- more expressive guarded conditionals.

Nondeterminism is a fundamental building block for system specification because it allows an implementer the freedom to postpone design choices. Dijkstra in his guarded commands [10, 11] replaced the standard prioritised conditional statement by a non-deterministic conditional, in which there was no priority between the branches and if more than one condition is true, any of the branches with a true condition may be executed. The same approach can be used within teleo-reactive programs: the prioritised conditional (as used in the example above) can be replaced by a more general non-deterministic conditional, although the semantics in the teleo-reactive case is quite different. As with Dijkstra's conditional, this requires that the condition for each branch must be expressed in full, because one can no longer rely on the negation of the earlier conditions. For example, the *mine_pump* procedure is equivalent to

$$\begin{array}{l} \text{Critical} \leq \text{methane} \rightarrow \text{alarm} \\ \sqcap \text{methane} < \text{Critical} \rightarrow \text{operate} \end{array}$$

Any conditional can be expressed in terms of a nondeterministic choice between guarded actions, for example, the conditional

$$\begin{array}{l} c0 \rightarrow a0, \\ c1 \rightarrow a1 \end{array}$$

is equivalent to

$$\begin{array}{l} c0 \rightarrow a0 \\ \sqcap \neg c0 \wedge c1 \rightarrow a1 \\ \sqcap \neg c0 \wedge \neg c1 \rightarrow \text{chaos} \end{array}$$

In the expanded form, the negation of the guard of the first branch appears explicitly in the second branch, and the chaotic behaviour in the case where neither guard holds is also made explicit.

In general, the guard conditions of a nondeterministic choice do not need to be mutually exclusive. The following is refined by the procedure *operate* above. When the water level is between Low

and High, this version allows either pump or nil to be active, and for control to switch between the two at any time.

$$\begin{array}{l} \text{Low} < \text{water} \rightarrow \text{pump} \\ \sqcap \text{water} \leq \text{High} \rightarrow \text{nil} \end{array}$$

If $\text{High} < \text{water}$ only the first branch is enabled, i.e., the action is *pump*, and if $\text{water} \leq \text{Low}$ only the second branch is enabled, i.e., the action is *nil*.

The following gives the definition of the abstract syntax for teleo-reactive programs, including our extensions.

Definition 1. The abstract syntax for teleo-reactive programs is defined by the following productions.

$$\begin{array}{l} GA ::= \text{Pred} \rightarrow A \\ A ::= \text{cond.}(\text{seq}.GA) \mid A \sqcap A \mid GA \mid \text{Identifier} \end{array}$$

The syntax allows an action (*A*) to be either: a conditional consisting of a sequence of guarded actions (*GA*); a nondeterministic choice (\sqcap) between actions; a single guarded action; or an identifier giving the name of an action. A guarded action (*GA*) consists of a predicate on the state of the system and an action. We use standard notation for predicates and hence don't give their syntax here. When writing teleo-reactive programs below we use the convention that nondeterministic choice (\sqcap) has lower precedence than guarding (\rightarrow). We use the operator " $\widehat{}$ " for sequence concatenation and the notation $\langle x_0, x_1, \dots, x_{n-1} \rangle$ for a sequence with *n* elements.

Our guarded actions are more general than standard teleo-reactive guarded actions, in that they allow the action part to be an arbitrary action, including a sequence of guarded actions or a nondeterministic choice. This generalisation helps simplify the presentation of the semantics because a procedure is syntactically just an action and any occurrence of a procedure identifier can be replaced by the corresponding action.

4. SEMANTICS

Nilsson only gives an informal description of the simple "circuit-like" semantics of teleo-reactive programs [22, 24]. Here we give a formal treatment of the semantics necessary to validate rules for reasoning about teleo-reactive programs.

The behaviour of a teleo-reactive system can be decomposed by partitioning time into intervals over which a single action is active. That action determines the behaviour of the system for that interval. Hence the ability to reason over time intervals and combine reasoning over time intervals forms the basis of a semantics for teleo-reactive programs. The motivation for our approach to the semantics comes from the Duration Calculus [9] and research on real-time refinement [21, 20, 14, 13].

A trace of the behaviour of an action over time gives the state of the variables at every time, i.e., it is a mapping from time to the state, Σ , which is itself a mapping from variables names (*Var*) to values (*Value*). We will take *Time* to be the set of non-negative real numbers. A trace predicate is either true or false for a given trace.

$$\begin{array}{l} \Sigma \cong \text{Var} \rightarrow \text{Value} \\ \text{Trace} \cong \text{Time} \rightarrow \Sigma \\ \text{TracePredicate} \cong \text{Trace} \rightarrow \text{Boolean} \end{array}$$

We promote the standard boolean operators such as conjunction and disjunction to trace predicates by defining them pointwise on the traces, e.g., for trace σ ,

$$(p \wedge q).\sigma = p.\sigma \wedge q.\sigma.$$

For predicate c ; actions a, a_0, a_1 ; time interval Δ ; and sequence of guarded actions s ; behaviours are defined as follows.

$$\begin{aligned}
bbeh.chaos &\cong True & (1) \\
bbeh.(c \rightarrow a) &\cong \Box c \wedge beh.a & (2) \\
bbeh.(a_0 \sqcap a_1) &\cong bbeh.a_0 \vee bbeh.a_1 & (3) \\
bbeh.(cond.\langle \rangle) &\cong True & (4) \\
bbeh.(cond.\langle c \rightarrow a \rangle \frown s) &\cong bbeh.(c \rightarrow a \sqcap \neg c \rightarrow cond.s) & (5) \\
beh.a.\Delta &\cong \exists \delta : part.\Delta \bullet \forall i : dom.\delta \bullet bbeh.a.(\delta.i) & (6)
\end{aligned}$$

Figure 2: Semantics of T-R programs

To give the semantics of teleo-reactive programs we need to restrict our attention to the behaviour of an action over a contiguous time interval for which the action is active. The set *Interval* contains all the contiguous subsets of *Time*. We also promote the boolean operators one step further to act on the type $(Interval \rightarrow TracePredicate)$, e.g., for time interval Δ ,

$$(p \wedge q).\Delta = p.\Delta \wedge q.\Delta$$

and hence from the promotion to traces

$$(p \wedge q).\Delta.\sigma = p.\Delta.\sigma \wedge q.\Delta.\sigma$$

To promote a boolean condition, c , to hold for all times in an interval, we use the notation “ $\Box c$ ”.

Definition 2. For any time interval Δ , trace σ , and state predicate, c , (i.e., $c \in \Sigma \rightarrow \text{Boolean}$)

$$(\Box c).\Delta.\sigma \cong (\forall t : \Delta \bullet c.(\sigma.t)).$$

The operator “ \Box ” has higher precedence than the other operators.

Behaviours of actions over a time interval are characterised by a trace predicate. We give the semantics in two steps: the basic behaviour function, *bbeh*, defines the behaviour over an interval over which a single action is enabled and the behaviour function *beh* pieces together a contiguous sequence of these intervals to provide the behaviour over an arbitrary interval. Both *beh* and *bbeh* map teleo-reactive actions from the abstract syntax to functions from time intervals to trace predicates.

$$beh, bbeh : A \rightarrow (Interval \rightarrow TracePredicate)$$

We assume that the names of any procedures are replaced by the teleo-reactive action they are defined to equal. The semantics of actions are given in Figure 2. The primitive durative action *chaos* is completely unpredictable: any trace is allowed in its behaviour. Note that *True* is the promoted version of the boolean *true*, so that $True.\Delta.\sigma = true$. For a single guarded action, $c \rightarrow a$, and time interval Δ , a trace is a basic behaviour of the guarded action over Δ if c holds over Δ and the trace is a behaviour of the action a over Δ (2). Again, the definition uses the promoted operator, so that

$$\begin{aligned}
bbeh.(c \rightarrow a).\Delta.\sigma &= (\Box c \wedge beh.a).\Delta.\sigma \\
&= (\Box c).\Delta.\sigma \wedge beh.a.\Delta.\sigma \\
&= (\forall t : \Delta \bullet c.(\sigma.t)) \wedge beh.a.\Delta.\sigma
\end{aligned}$$

For a nondeterministic choice, $a_0 \sqcap a_1$, a trace is a basic behaviour of the choice over an interval Δ if it is a basic behaviour of a_0 over Δ or a basic behaviour of a_1 over Δ (3). The empty sequence of guarded actions behaves as the totally unconstrained action (4) (i.e., *chaos*). A sequence of guarded actions, with head the guarded

action $(c \rightarrow a)$ and tail the sequence of guarded actions, s , behaves as a if c holds over Δ , or as *cond.s* if $\neg c$ holds over Δ (5).

A behaviour of an action a over an interval Δ is formed from basic behaviours of a over some partition of Δ (6). Figure 3 illustrates a possible partition for a conditional.

Definition 3. A sequence of time intervals, δ , *partitions* a time interval Δ , if the union of the intervals in δ equals Δ and consecutive intervals within δ adjoin, (i.e., the end of interval $\delta.i$ equals the start of interval $\delta.(i+1)$). We use the notation $part.\Delta$ for the set of all partitions of an interval Δ . We do not rule out the sequence δ having an infinite number of elements, but (to avoid Zeno-like behaviour) we do require that if δ is infinite then Δ must be an infinite interval.

We now give some properties derived from these definitions.

THEOREM 1. *The behaviours of action $(c \rightarrow a)$ over a time interval Δ are exactly its basic behaviours, i.e., $beh.(c \rightarrow a) = bbeh.(c \rightarrow a)$.*

PROOF. The proof proceeds by first decomposing the behaviour of the guarded action, $(c \rightarrow a)$, into a partition δ of its basic behaviours. For each interval $\delta.i$ in δ , the behaviour of the action a is then decomposed into the basic behaviours of a over a partition (β) of $\delta.i$. Next we form a partition γ of Δ that is composed of the union of the partitions of every $\delta.i$. For any time interval Δ we have

$$\begin{aligned}
&beh.(c \rightarrow a).\Delta \\
&\equiv \text{definition (6) of } beh \\
&\quad \exists \delta : part.\Delta \bullet \forall i : dom.\delta \bullet bbeh.(c \rightarrow a).(\delta.i) \\
&\equiv \text{definition (2) of } bbeh.(c \rightarrow a) \\
&\quad \exists \delta : part.\Delta \bullet \forall i : dom.\delta \bullet (\Box c).\delta.i \wedge beh.a.(\delta.i) \\
&\equiv \text{Definition 3; Definition 2; property of } \Box \\
&\quad (\Box c).\Delta \wedge \exists \delta : part.\Delta \bullet \forall i : dom.\delta \bullet beh.a.(\delta.i) \\
&\equiv \text{definition (6) of } beh \\
&\quad (\Box c).\Delta \wedge \exists \delta : part.\Delta \bullet \forall i : dom.\delta \bullet \\
&\quad \quad \exists \beta : part.(\delta.i) \bullet \forall j : dom.\beta \bullet bbeh.a.(\delta.j) \\
&\equiv \text{form a new partition } \gamma \text{ by combining partitions for each } \delta.i \\
&\quad (\Box c).\Delta \wedge \exists \gamma : part.\Delta \bullet \forall i : dom.\gamma \bullet bbeh.a.(\gamma.i) \\
&\equiv \text{definition (6) of } beh \\
&\quad (\Box c).\Delta \wedge beh.a.\Delta \\
&\equiv \text{definition (2) of } bbeh \\
&\quad bbeh.(c \rightarrow a).\Delta
\end{aligned}$$

□

THEOREM 2. *For any two actions a_0 and a_1 , if $bbeh.a_0 = bbeh.a_1$, then $beh.a_0 = beh.a_1$.*

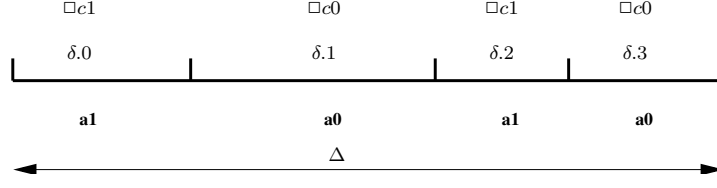


Figure 3: Possible partition for “ $c_0 \rightarrow a_0, c_1 \rightarrow a_1$ ”

PROOF. Assume $bbeh.a_0 = bbeh.a_1$, then for any time interval Δ we have

$$\begin{aligned}
& beh.a_0.\Delta \\
\equiv & \text{definition (6) of } beh \\
& \exists \delta : part.\Delta \bullet \forall i : dom.\delta \bullet bbeh.a_0.(\delta.i) \\
\equiv & \text{assumption} \\
& \exists \delta : part.\Delta \bullet \forall i : dom.\delta \bullet bbeh.a_1.(\delta.i) \\
\equiv & \text{definition (6) of } beh \\
& beh.a_1.\Delta
\end{aligned}$$

□

From this theorem we can deduce the following corollary using the definition of $bbeh$ for a conditional (5).

COROLLARY 1. *The action $cond.((c \rightarrow a) \frown s)$ is equivalent to the action $(c \rightarrow a \sqcap \neg c \rightarrow cond.s)$.*

THEOREM 3. *The action $(c_0 \rightarrow cond.((c_1 \rightarrow a_1) \frown s))$ is equivalent to the action $(c_0 \wedge c_1 \rightarrow a_1 \sqcap c_0 \wedge \neg c_1 \rightarrow cond.s)$.*

PROOF. For any time interval Δ we have

$$\begin{aligned}
& beh.(c_0 \rightarrow cond.((c_1 \rightarrow a_1) \frown s)).\Delta \\
\equiv & \text{Theorem 1 and definition (2)} \\
& (\sqcap c_0).\Delta \wedge beh.(cond.((c_1 \rightarrow a_1) \frown s)).\Delta \\
\equiv & \text{Corollary 1} \\
& (\sqcap c_0).\Delta \wedge beh.(c_1 \rightarrow a_1 \sqcap \neg c_1 \rightarrow cond.s).\Delta \\
\equiv & \text{definition (6) of } beh \\
& (\sqcap c_0).\Delta \wedge \exists \delta : part.\Delta \bullet \forall i : dom.\delta \bullet \\
& \quad bbeh.(c_1 \rightarrow a_1 \sqcap \neg c_1 \rightarrow cond.s).(\delta.i) \\
\equiv & \text{distributing } \sqcap c_0 \text{ over the partition } \delta \\
& \exists \delta : part.\Delta \bullet \forall i : dom.\delta \bullet (\sqcap c_0).(\delta.i) \wedge \\
& \quad bbeh.(c_1 \rightarrow a_1 \sqcap \neg c_1 \rightarrow cond.s).(\delta.i) \\
\equiv & \text{definitions (3) and (2) of } bbeh \\
& \exists \delta : part.\Delta \bullet \forall i : dom.\delta \bullet (\sqcap c_0).(\delta.i) \wedge \\
& \quad ((\sqcap c_1).(\delta.i) \wedge beh.a_1.(\delta.i) \vee \\
& \quad (\sqcap \neg c_1).(\delta.i) \wedge beh.(cond.s).(\delta.i)) \\
\equiv & \text{properties of } \sqcap \\
& \exists \delta : part.\Delta \bullet \forall i : dom.\delta \bullet \\
& \quad ((\sqcap(c_0 \wedge c_1)).(\delta.i) \wedge beh.a_1.(\delta.i) \vee \\
& \quad (\sqcap(c_0 \wedge \neg c_1)).(\delta.i) \wedge beh.(cond.s).(\delta.i)) \\
\equiv & \text{definitions (2) and (3) of } bbeh \\
& \exists \delta : part.\Delta \bullet \forall i : dom.\delta \bullet \\
& \quad bbeh.(c_0 \wedge c_1 \rightarrow a_1 \sqcap c_0 \wedge \neg c_1 \rightarrow cond.s).(\delta.i) \\
\equiv & \text{definition (6) of } beh \\
& beh.(c_0 \wedge c_1 \rightarrow a_1 \sqcap c_0 \wedge \neg c_1 \rightarrow cond.s).\Delta
\end{aligned}$$

□

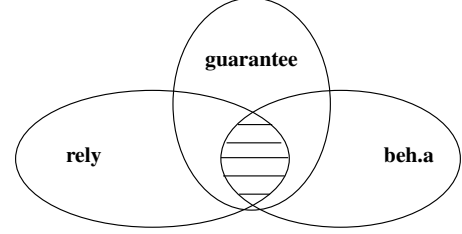


Figure 4: Satisfaction of a rely/guarantee pair

5. REASONING

The rely/guarantee reasoning of Jones [16] has been used to provide compositional reasoning about programs. Other related approaches include Mahony’s network of dataflow processes [20], and Broy and Stølen’s Focus approach [6], amongst others. The rely/guarantee conditions of Jones [16] are relations between two states only. For real-time teleo-reactive programs we want to be able to reason about the behaviour of the system over time and hence for teleo-reactive programs we require the rely and guarantee conditions to be predicates of the time interval over which the procedure operates, i.e., the rely and guarantee are functions from time intervals to trace predicates (i.e., of type $Interval \rightarrow TracePredicate$). For example, in Section 2 the primitive duration action `pump` was specified by a combination of a rely condition, r_pump , that the methane level is below critical over the interval that the `pump` action is active, and a guarantee condition, g_pump , that the rate of flow of water out of the mine is greater than some minimum rate over the same interval.

Whenever the rely condition holds the behaviour of the action must be such that the guarantee condition holds. This is illustrated in Figure 4.

Definition 4. An action, a , satisfies a rely/guarantee pair, written $r \{a\} g$, if all behaviours of a for which the rely condition holds also satisfy the guarantee condition:

$$r \wedge beh.a \Rightarrow g$$

Note that, due to the promotion of the boolean operators over intervals, this is equivalent to

$$\forall \Delta : Interval \bullet r.\Delta \wedge beh.a.\Delta \Rightarrow g.\Delta$$

which in turn, due the promotion of the boolean operators over traces, is equivalent to

$$\forall \sigma : Trace \bullet \forall \Delta : Interval \bullet r.\Delta.\sigma \wedge beh.a.\Delta.\sigma \Rightarrow g.\Delta.\sigma$$

THEOREM 4. For rely conditions r and r' , guarantee conditions g and g' , and action a , if $r \Rightarrow r'$ and $r' \{a\} g'$ and $r \wedge g' \Rightarrow g$, then $r \{a\} g$.

PROOF. The proof follows from the definition of ‘‘satisfies’’.

$$\begin{aligned}
& r \{a\} g \\
& \equiv \text{Definition 4} \\
& r \wedge \text{beh}.a \Rightarrow g \\
& \Leftrightarrow \text{as } r \wedge g' \Rightarrow g \\
& r \wedge \text{beh}.a \Rightarrow g' \\
& \Leftrightarrow \text{as } r \Rightarrow r' \\
& r' \wedge \text{beh}.a \Rightarrow g' \\
& \equiv \text{Definition 4} \\
& r' \{a\} g'
\end{aligned}$$

□

We provide rely/guarantee rules for reasoning about teleo-reactive programs in each of the forms given in the abstract syntax. We start with a single guarded command, $(c \rightarrow a)$. This will satisfy a rely/guarantee pair, (r, g) , if a satisfies the pair for all intervals for which c holds over the whole interval.

THEOREM 5. For a rely condition r , a guarantee condition g , a state predicate c , and an action a ,

$$r \{c \rightarrow a\} g \equiv (r \wedge \Box c) \{a\} g.$$

PROOF.

$$\begin{aligned}
& r \{c \rightarrow a\} g \\
& \equiv \text{Definition 4 of satisfies} \\
& r \wedge \text{beh}.(c \rightarrow a) \Rightarrow g \\
& \equiv \text{Theorem 1 and definition (2) of } b\text{beh}.(c \rightarrow a) \\
& r \wedge \Box c \wedge \text{beh}.a \Rightarrow g \\
& \equiv \text{Definition 4 of satisfies} \\
& (r \wedge \Box c) \{a\} g
\end{aligned}$$

□

The execution of a nondeterministic choice or a conditional may switch between subactions during its execution. The subactions may only execute for some subinterval of the overall execution time interval. To reason about a subaction we need to be able to assume a rely condition over the subinterval over which it executes. Hence we impose a requirement on the rely condition that, if it holds for an interval, it holds for any subinterval of that interval.

Definition 5. A (rely) condition, r , decomposes over intervals if whenever r holds for an interval Δ , it holds for its subintervals:

$$\forall \Delta, \Delta' : \text{Interval} \bullet \Delta' \subseteq \Delta \wedge r.\Delta \Rightarrow r.\Delta'$$

For example, for state predicate, c , $\Box c$ decomposes over intervals, but $r.\Delta \hat{=} 10 \leq \text{length}.\Delta$ does not because a subinterval of Δ may be of length less than 10.

In a similar fashion, we would like to be able to guarantee a condition g holds over the whole of the execution interval, Δ , of a nondeterministic choice or a conditional. To do this we need to be able to combine the guarantees of the subintervals over which the subactions execute to give an overall guarantee over the whole interval Δ . Hence we require that if a guarantee condition, g , holds for every subinterval in a partition of Δ , it will hold over Δ .

Definition 6. A (guarantee) condition, g , composes over intervals if whenever g holds for every subinterval in a partition δ of Δ , it holds for Δ :

$$\begin{aligned}
& \forall \Delta : \text{Interval} \bullet \forall \delta : \text{part}.\Delta \bullet \\
& (\forall i : \text{dom}.\delta \bullet g.(\delta.i)) \Rightarrow g.\Delta
\end{aligned}$$

For example, for a predicate c , $\Box c$ composes over intervals and hence the condition $\Box(x = 0 \vee x = 1)$ composes over intervals, but $\Box(x = 0) \vee \Box(x = 1)$ does not, because it is possible that an interval, Δ , may be split into two subintervals such that $\Box(x = 0)$ holds for one subinterval and $\Box(x = 1)$ holds for the other, but neither $\Box(x = 0)$ nor $\Box(x = 1)$ holds for the whole of Δ .

THEOREM 6. For a rely condition r that decomposes over intervals, a guarantee condition g that composes over intervals, and actions a_0 and a_1 ,

$$r \{a_0 \sqcap a_1\} g$$

provided

$$(r \{a_0\} g) \wedge (r \{a_1\} g).$$

PROOF. Assume $r \{a_0\} g$ and $r \{a_1\} g$, and hence for any time interval Δ ,

$$r.\Delta \wedge \text{beh}.a_0.\Delta \Rightarrow g.\Delta \quad (7)$$

$$r.\Delta \wedge \text{beh}.a_1.\Delta \Rightarrow g.\Delta \quad (8)$$

The proof then proceeds as follows:

$$\begin{aligned}
& r \{a_0 \sqcap a_1\} g \\
& \equiv \text{Definition 4 of satisfies} \\
& r \wedge \text{beh}.(a_0 \sqcap a_1) \Rightarrow g \\
& \equiv \text{expanding promoted operators} \\
& \forall \Delta : \text{Interval} \bullet r.\Delta \wedge \text{beh}.(a_0 \sqcap a_1).\Delta \Rightarrow g.\Delta \\
& \equiv \text{definition (6) of } \text{beh} \\
& \forall \Delta : \text{Interval} \bullet r.\Delta \wedge \\
& (\exists \delta : \text{part}.\Delta \bullet \forall i : \text{dom}.\delta \bullet b\text{beh}.(a_0 \sqcap a_1).(\delta.i)) \Rightarrow g.\Delta \\
& \equiv \text{definition (3) of } b\text{beh}.(a_0 \sqcap a_1) \\
& \forall \Delta : \text{Interval} \bullet r.\Delta \wedge \\
& (\exists \delta : \text{part}.\Delta \bullet \forall i : \text{dom}.\delta \bullet (b\text{beh}.a_0.(\delta.i) \vee b\text{beh}.a_1.(\delta.i))) \\
& \Rightarrow g.\Delta \\
& \Leftrightarrow \text{as } r.\Delta \Rightarrow r.(\delta.i) \text{ for all } i \text{ in } \text{dom}.\delta \\
& \forall \Delta : \text{Interval} \bullet \\
& (\exists \delta : \text{part}.\Delta \bullet \forall i : \text{dom}.\delta \bullet \\
& r.(\delta.i) \wedge (b\text{beh}.a_0.(\delta.i) \vee b\text{beh}.a_1.(\delta.i))) \\
& \Rightarrow g.\Delta \\
& \equiv \\
& \forall \Delta : \text{Interval} \bullet \\
& (\exists \delta : \text{part}.\Delta \bullet \forall i : \text{dom}.\delta \bullet \\
& (r.(\delta.i) \wedge b\text{beh}.a_0.(\delta.i) \vee \\
& r.(\delta.i) \wedge b\text{beh}.a_1.(\delta.i))) \\
& \Rightarrow g.\Delta \\
& \Leftrightarrow \text{from (7) and (8)} \\
& \forall \Delta : \text{Interval} \bullet \\
& (\exists \delta : \text{part}.\Delta \bullet (\forall i : \text{dom}.\delta \bullet g.(\delta.i))) \Rightarrow g.\Delta
\end{aligned}$$

This holds because g composes over intervals. □

THEOREM 7. For a rely condition, r , that decomposes over intervals, a guarantee condition, g , that composes over intervals, predicates, c_0 and c_1 , actions, a_0 and a_1 ,

$$r \{ \text{cond}.\langle c_0 \rightarrow a_0, c_1 \rightarrow a_1 \rangle \} g$$

provided

$$(r \wedge \Box c_0) \{a_0\} g \quad (9)$$

$$(r \wedge \Box (\neg c_0 \wedge c_1)) \{a_1\} g \quad (10)$$

$$r \Rightarrow \Box (c_0 \vee c_1) \quad (11)$$

PROOF.

$$\begin{aligned}
& r \{ \text{cond.} \langle c_0 \rightarrow a_0, c_1 \rightarrow a_1 \rangle \} g \\
& \equiv \text{Corollary 1} \\
& r \{ c_0 \rightarrow a_0 \sqcap \neg c_0 \rightarrow \text{cond.} \langle c_1 \rightarrow a_1 \rangle \} g \\
& \Leftarrow \text{Theorem 6 and assumptions on } r \text{ and } g \\
& r \{ c_0 \rightarrow a_0 \} g \wedge \\
& r \{ \neg c_0 \rightarrow \text{cond.} \langle c_1 \rightarrow a_1 \rangle \} g \\
& \equiv \text{Theorem 3} \\
& r \{ c_0 \rightarrow a_0 \} g \wedge \\
& r \{ \neg c_0 \wedge c_1 \rightarrow a_1 \sqcap \neg c_0 \wedge \neg c_1 \rightarrow \text{cond.} \langle \rangle \} g \\
& \Leftarrow \text{Theorem 6 and assumptions on } r \text{ and } g \\
& r \{ c_0 \rightarrow a_0 \} g \wedge \\
& r \{ \neg c_0 \wedge c_1 \rightarrow a_1 \} g \wedge \\
& r \{ \neg c_0 \wedge \neg c_1 \rightarrow \text{cond.} \langle \rangle \} g \\
& \equiv \text{Definition 4 of satisfies} \\
& r \wedge \text{beh.} \langle c_0 \rightarrow a_0 \rangle \Rightarrow g \wedge \\
& r \wedge \text{beh.} \langle \neg c_0 \wedge c_1 \rightarrow a_1 \rangle \Rightarrow g \wedge \\
& r \wedge \text{beh.} \langle \neg c_0 \wedge \neg c_1 \rightarrow \text{cond.} \langle \rangle \rangle \Rightarrow g \\
& \equiv \text{Theorem 1 and definition (2)} \\
& r \wedge \square c_0 \wedge \text{beh.} a_0 \Rightarrow g \wedge \\
& r \wedge \square (\neg c_0 \wedge c_1) \wedge \text{beh.} a_1 \Rightarrow g \wedge \\
& r \wedge \square (\neg c_0 \wedge \neg c_1) \wedge \text{beh.} \langle \text{cond.} \langle \rangle \rangle \Rightarrow g \\
& \equiv \text{definition (4) of } b\text{beh.} \langle \text{cond.} \langle \rangle \rangle \\
& (r \wedge \square c_0) \wedge \text{beh.} a_0 \Rightarrow g \wedge \\
& (r \wedge \square (\neg c_0 \wedge c_1)) \wedge \text{beh.} a_1 \Rightarrow g \wedge \\
& r \wedge \square (\neg c_0 \wedge \neg c_1) \wedge \text{True} \Rightarrow g \\
& \equiv \text{Definition 4} \\
& (r \wedge \square c_0) \{ a_0 \} g \wedge \\
& (r \wedge \square (\neg c_0 \wedge c_1)) \{ a_1 \} g \wedge \\
& r \wedge \square (\neg c_0 \wedge \neg c_1) \Rightarrow g
\end{aligned}$$

These follow from the assumptions (9), (10) and (11). \square

The above theorem can be generalised to conditionals with more than two branches.

6. EXAMPLE CONTINUED

We would like to show that the mine pump example given in Section 2 satisfies a number of properties. The variable $water$ represents the level of water in the mine shaft. We use the notation $dwater/dt$ to stand for the derivative of the water level with respect to time, i.e., the rate of change of the water level. The variable $water_in$ ($water_out$) represents the instantaneous rate of flow of water into (out of) the mine (for simplicity these are expressed in terms of the rate of change of the level of water in the mine). We will assume that the environment of the mine pump satisfies certain properties, i.e., its ‘‘laws of nature’’, in particular, at any time the rate of change of the water level is equal to the flow of water into the mine minus the flow of water out of the mine, and that there is assumed to be an upper bound, $MaxIn$, on the flow of water into the mine.

$$r \hat{=} \square \left(\begin{array}{l} dwater/dt = water_in - water_out \wedge \\ 0 \leq water_out \wedge \\ 0 \leq water_in \leq MaxIn \end{array} \right)$$

When the primitive action **pump** is active it guarantees that the water is extracted from the mine at a minimum rate of $MinOut$, i.e.,

$$g_pump \hat{=} \square (MinOut \leq water_out \wedge pump_active)$$

provided the water level is above Low and methane level is not critical:

$$r_pump \hat{=} \square (Low < water \wedge methane < Critical)$$

These form the guarantee and rely conditions for the primitive action **pump**.

We would like to show our mine pump system guarantees the following condition

$$g \hat{=} \square \left(\begin{array}{l} methane < Critical \wedge High < water \implies \\ dwater/dt \leq MaxIn - MinOut \end{array} \right)$$

which, provided that $MaxIn < MinOut$, guarantees that the water level will decrease.

We can use Theorem 7 to show

$$r \{ \text{mine_pump} \} g.$$

where $mine_pump$ was defined as follows.

$$\text{mine_pump} \hat{=} \square$$

$Critical \leq methane$	\rightarrow	alarm ,
$true$	\rightarrow	operate

The theorem holds provided

$$(r \wedge \square (Critical \leq methane)) \{ \text{alarm} \} g \quad (12)$$

$$(r \wedge \square (methane < Critical)) \{ \text{operate} \} g \quad (13)$$

$$r \hat{=} \square (Critical \leq methane \vee true) \quad (14)$$

Requirement (14) holds trivially. Using Definition 4, requirement (12) holds because $Critical \leq methane$ is the complement of $methane < Critical$ used on the left side of the implication within g . For requirement (13) we apply Theorem 7 to procedure **operate**.

$$\text{operate} \hat{=} \square$$

$\left(\begin{array}{l} High < water \vee \\ Low < water \wedge pump_active \end{array} \right)$	\rightarrow	pump ,
$true$	\rightarrow	nil

Let c be the guard on the first branch of procedure **operate**. The conditions we need to show are

$$(r \wedge \square (c \wedge methane < Critical)) \{ \text{pump} \} g \quad (15)$$

$$(r \wedge \square (\neg c \wedge methane < Critical)) \{ \text{nil} \} g \quad (16)$$

$$r \wedge \square (methane < Critical) \hat{=} \square (c \vee true) \quad (17)$$

Requirement (17) holds trivially. The negation of c implies that $water \leq High$, which implies the left side of the implication in g is false, and hence requirement (16) holds. Requirement (15) follows using Theorem 4 and the rely and guarantee conditions of **pump**, provided

$$r \wedge \square (c \wedge methane < Critical) \hat{=} r_pump$$

$$g_pump \wedge r \wedge \square (c \wedge methane < Critical) \hat{=} g$$

The first requirement holds because c implies $Low < water$. For the second requirement, from r we have

$$\begin{aligned}
& dwater/dt = water_in - water_out \\
& \Rightarrow \text{as } g_pump \text{ implies } MinOut \leq water_out \\
& dwater/dt \leq water_in - MinOut \\
& \Rightarrow \text{as } r \text{ implies } water_in \leq MaxIn \\
& dwater/dt \leq MaxIn - MinOut
\end{aligned}$$

Using the above we can prove more complex properties, such as, if at time t the water level is initially above $High$, then the water level will be below $High$ by time

$$t + \frac{water.t - High}{MaxIn - MinOut}$$

provided the methane level is not critical over the time interval from t until this time.

7. CONCLUSIONS

There is an increasing trend to use more sophisticated real-time computer systems to control safety-critical infrastructure. Failures in these systems can lead to injury or loss of life, and hence it is essential that they be developed to the highest standards of dependability. In addition, these systems are inherently more complex than conventional real-time control applications. To develop such systems we need methods and tools that can verify that the systems behave both safely and effectively, as well as help control the complexity of their design.

The teleo-reactive programming paradigm of Nilsson provides a remarkably simple notation for expressing robust real-time control programs. In this paper we have developed a time-interval semantics for teleo-reactive programs and provided rely/guarantee reasoning rules that have been shown correct with respect to these rules.

Future work consists of extending the semantics and reasoning rules to handle concurrency. For some teleo-reactive programs guarded actions may only be active for an instant during which time they change the state to disable themselves. There are two approaches we can follow to address these instantaneous actions, either

- enrich our semantics to allow a finite number of instantaneous actions to happen at the same real time, or
- make use of the notion of time bands [7, 5].

For the latter, a system description can be structured into a number of time bands each with its timing precision. At a higher-level time band an action may be considered instantaneous if it is within the precision of the band, but within a lower level band the same action may be viewed as taking time.

8. ACKNOWLEDGMENTS

This paper has benefited from my collaborations with members of IFIP Working Group 2.3 on Programming Methodology as well as Cliff Jones, Alan Burns, and Keith Clark (who introduced me to teleo-reactive programming). This research was supported by Australian Research Council (ARC) Discovery Grant DP0558408, *Analysing and generating fault-tolerant real-time systems* and the EPSRC-funded Trustworthy Ambient Systems (TrAmS) Platform Project.

9. REFERENCES

- [1] R. Alur, T. A. Henzinger, and P.-H. Ho. Automatic symbolic verification of embedded systems. *IEEE Transactions on Software Engineering*, 22:181–201, 1996.
- [2] R. J. Back, L. Petre, and I. P. Paltor. Continuous action systems as a model for hybrid systems. *Nordic Journal of Computing*, 8(1):2–21, Spring 2001.
- [3] R.-J. Back and K. Sere. Action systems with synchronous communication. In *Programming Concepts, Methods, and Calculi (PROCOMET'94)*, pages 107–126. North-Holland, 1994.
- [4] R.-J. Back and J. von Wright. Trace refinement of action systems. In B. Jonsson and J. Parrow, editors, *Proc. of CONCUR '94: Concurrency Theory*, volume 836 of *LNCS*, pages 367–384. Springer-Verlag, 1994.
- [5] G. Baxter, A. Burns, and K. Tan. Evaluating timebands as a tool for structuring the design of socio-technical systems. In P. Bust, editor, *Contemporary Ergonomics 2007*, pages 55–60. Taylor and Francis, 2007.
- [6] M. Broy and K. Stølen. *Specification and Development of Interactive Systems: FOCUS on Streams, Interfaces, and Refinement*. Springer, 2001.
- [7] A. Burns and G. Baxter. Time bands in systems structure. In D. Besnard, C. Gacek, and C. Jones, editors, *Structure for Dependability: Computer-Based Systems from an Interdisciplinary Perspective*, pages 74–88. Springer-Verlag, 2006.
- [8] A. Burns and A. Lister. A framework for building dependable systems. *Computer Journal*, 34(2):173–181, 1991.
- [9] Z. Chaochen and M. R. Hansen. *Duration Calculus: A Formal Approach to Real-Time Systems*. Springer Verlag, 1990.
- [10] E. W. Dijkstra. Guarded commands, nondeterminacy, and a formal derivation of programs. *CACM*, 18:453–458, 1975.
- [11] E. W. Dijkstra. *A Discipline of Programming*. Prentice-Hall, 1976.
- [12] D. Harel and M. Politi. *Modeling Reactive Systems with Statecharts: The STATEMATE Approach*. McGraw-Hill, 1998.
- [13] I. J. Hayes. A predicative semantics for real-time refinement. In A. McIver and C. C. Morgan, editors, *Programming Methodology*, pages 109–133. Springer Verlag, 2003.
- [14] I. J. Hayes and M. Utting. A sequential real-time refinement calculus. *Acta Informatica*, 37(6):385–448, 2001.
- [15] T. A. Henzinger, P.-H. Ho, and H. Wong-Toi. Hytech: A model checker for hybrid systems. *Software Tools for Technology Transfer*, 1:110–122, 1997.
- [16] C. B. Jones. Accommodating interference in the formal design of concurrent object-based programs. *Formal Methods in System Design*, 8(2):105–122, March 1996.
- [17] M. Joseph, editor. *Real-time Systems: Specification, Verification and Analysis*. Prentice Hall, 1996.
- [18] L. Lamport. *Specifying Systems: The TLA+ Language and Tools for Hardware and Software Engineers*. Addison Wesley, 2003.
- [19] N. G. Leveson. *Safeware: System Safety and Computers*. Addison-Wesley, 1995.
- [20] B. P. Mahony. The DOVE approach to the design of complex dynamical processes. In *TPHOLS, Workshop of Formalising Continuous Mathematics*, pages 167–187, 2002.
- [21] B. P. Mahony and I. J. Hayes. A case-study in timed refinement: A mine pump. *IEEE Trans. on Software Engineering*, 18(9):817–826, 1992.
- [22] N. Nilsson. Teleo-reactive programs for agent control. *Journal of Artificial Intelligence Research*, 1:139–158, 1994.
- [23] N. Nilsson. Teleo-reactive programming web site, Accessed 28 August 2008. <http://robotics.stanford.edu/users/nilsson/trweb/tr.html>.
- [24] N. J. Nilsson. Teleo-reactive programs and the triple-tower architecture. *Electronic Transactions on Artificial Intelligence*, 5:99–110, 2001.
- [25] N. Storey. *Safety-Critical Computer Systems*. Addison-Wesley, 1996.