Reasoning About Teleo-Reactive Programs

Ian J. Hayes

School of ITEE The University of Queensland Brisbane, 4072, Australia

ICTAC Summer School 2010
August 29, 2010

Overview

Teleo-reactive programming was invented by Nils Nilsson [Nil94, Nil01, Nil08]. We

- develop a time-interval semantics
- develop rely/guarantee reasoning rules and
- apply the rules to an example program

(Greek: telos: end, purpose)

Teleo-reactive Programs

Teleo-reactive programs are described by a combination of

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

Nilsson's Tower Program

makeTower(s) — s is an non-empty list with no duplicates

$$
\begin{aligned}
\mathit{Tower}(s) &\rightarrow \mathit{nil},\\
\mathit{Ordered}(s) &\rightarrow \mathit{unpile}(\mathit{head}(s)),\\
\mathit{Null}(\mathit{tail}(s)) &\rightarrow \mathit{move\_to\_table}(\mathit{head}(s)),\\
\mathit{Tower}(\mathit{tail}(s)) &\rightarrow \mathit{move}(\mathit{head}(s), \mathit{head}(\mathit{tail}(s))),\\
\mathit{true} &\rightarrow \mathit{makeTower}(\mathit{tail}(s))
\end{aligned}
$$

move_to_table(x) — x is a block

$$
\begin{aligned}
\mathit{On}(x, \mathit{Ta}) &\rightarrow \mathit{nil},\\
\mathit{Holding}(x) &\rightarrow \mathit{put\_down}(x, \mathit{Ta}),\\
\mathit{Clear}(x) &\rightarrow \mathit{pick\_up}(x),\\
\mathit{true} &\rightarrow \mathit{unpile}(x)
\end{aligned}
$$

move(x,y) — x and y are distinct blocks

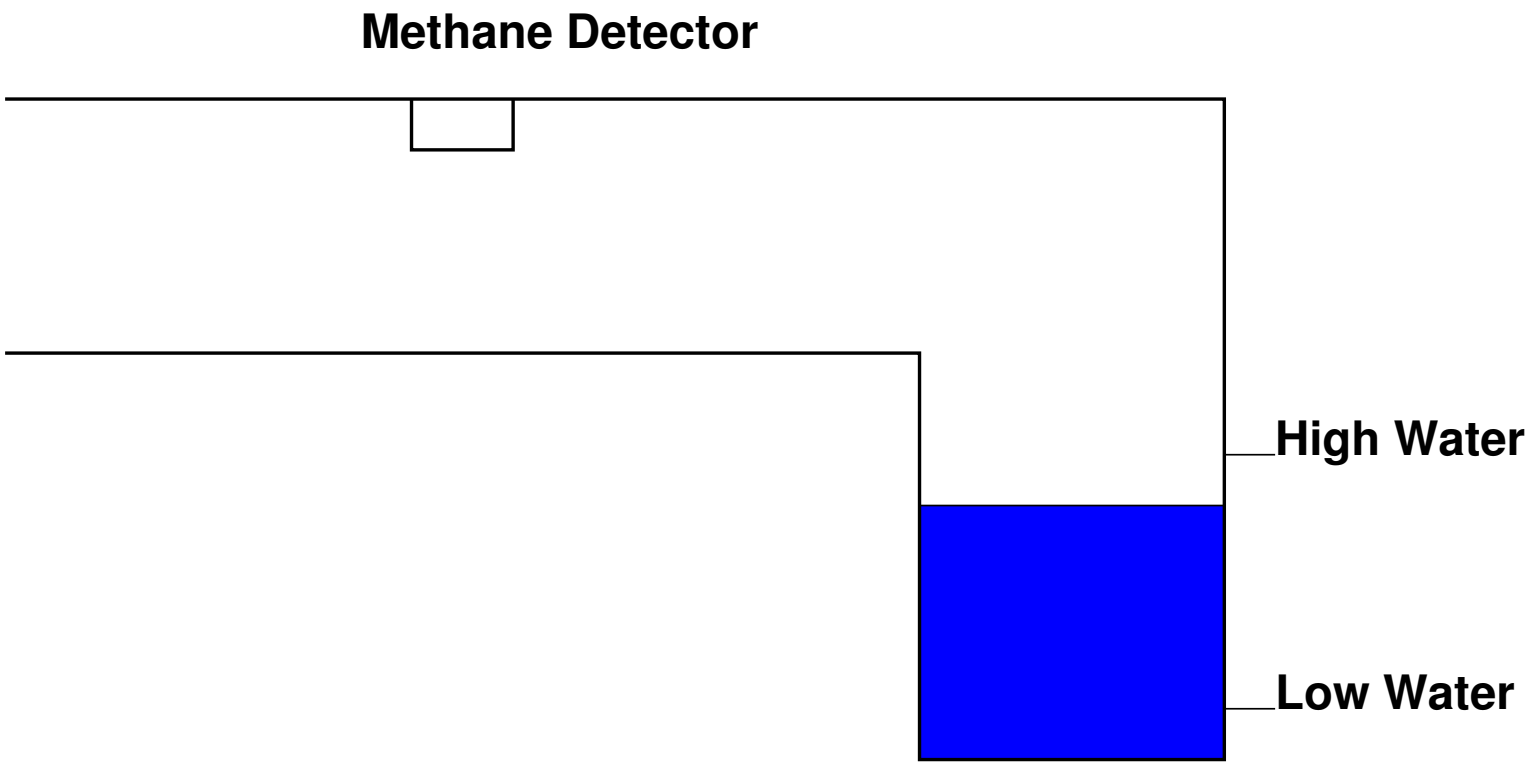
$$
\begin{aligned}
On(x, y) &\rightarrow nil, \\
Holding(x) \wedge Clear(y) &\rightarrow put\_down(x, y), \\
Holding(z) \wedge x \neq z &\rightarrow put\_down(z, Ta), \\
Clear(x) \wedge Clear(y) &\rightarrow pick\_up(x), \\
Clear(y) &\rightarrow unpile(x), \\
true &\rightarrow unpile(y)
\end{aligned}
$$

unpile(x) — x is a block

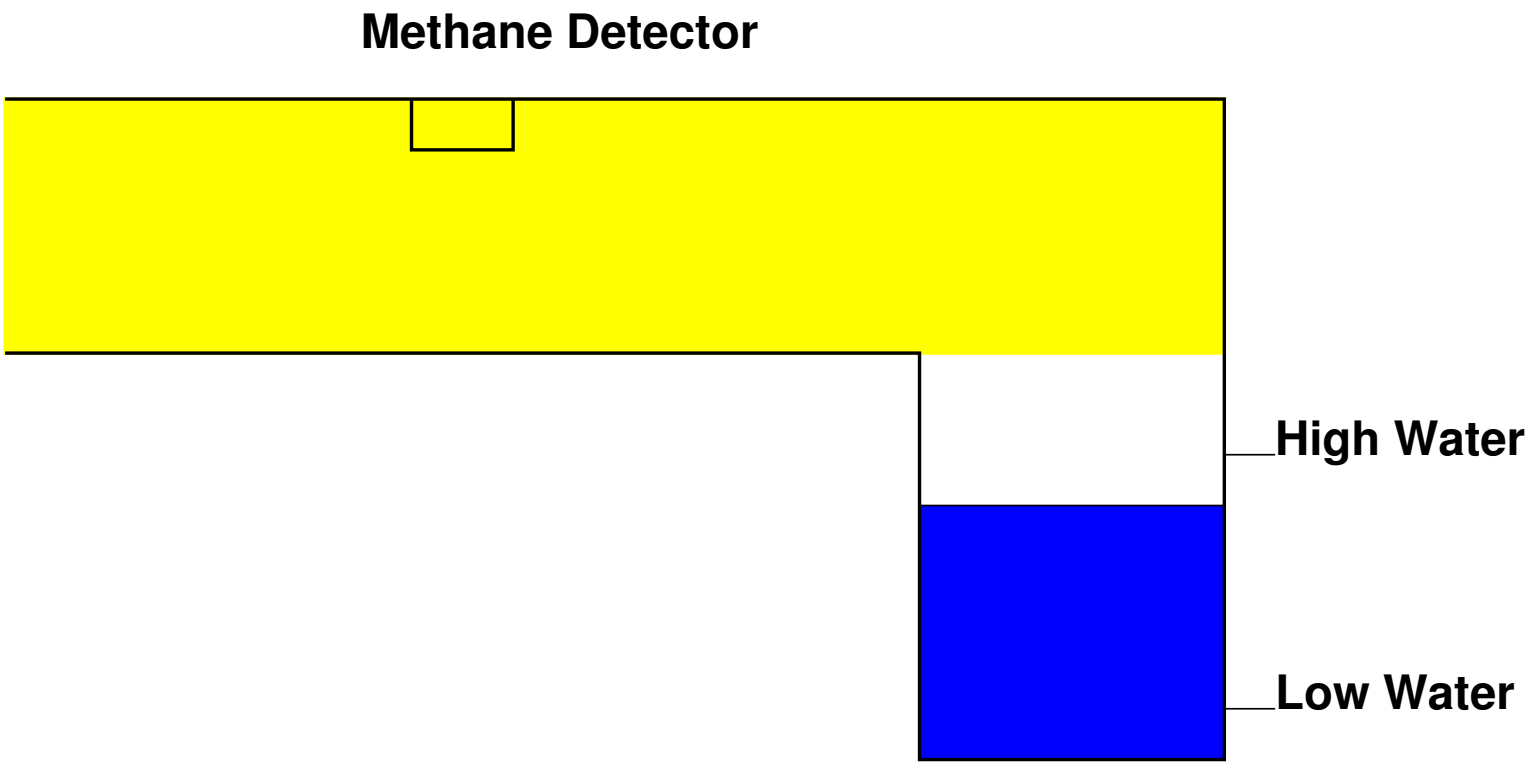
$$
\begin{aligned}
Clear(x) &\rightarrow nil, \\
On(y, x) &\rightarrow move\_to\_table(y)
\end{aligned}
$$

pick_up and *put_down* are primitive actions.

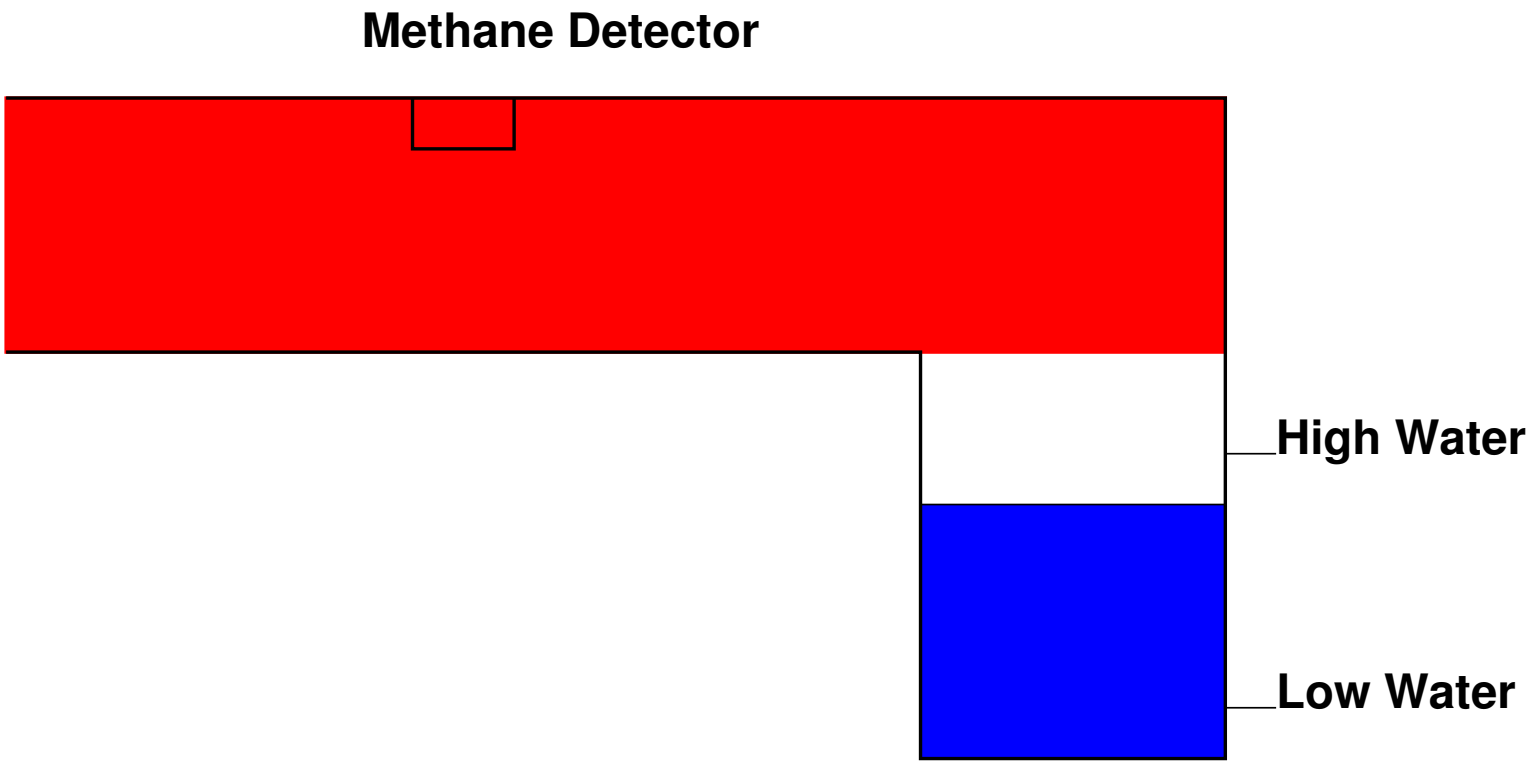
Coal mine with sump



Mine with methane



Mine with methane - explodes if pump on



Mine Pump Example [BL91, MH92, Jos96]

There are three sensed values:

- the level of methane in the mine, *methane*,
- the level of water in the mine, *water*, and
- an indicator of whether the pump is active, *pump_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.

Teleo-reactive mine pump program

mine_pump

$$\begin{array}{lcl} \mathit{Critical} \leq \mathit{methane} & \rightarrow & \text{alarm}, \\ \mathit{true} & \rightarrow & \text{operate} \end{array}$$

operate

$$\begin{array}{lcl} \left(\begin{array}{l} \mathit{High} < \mathit{water} \vee \\ \mathit{Low} < \mathit{water} \wedge \mathit{pump\_active} \end{array} \right) & \rightarrow & \text{pump}, \\ \mathit{true} & \rightarrow & \text{nil} \end{array}$$

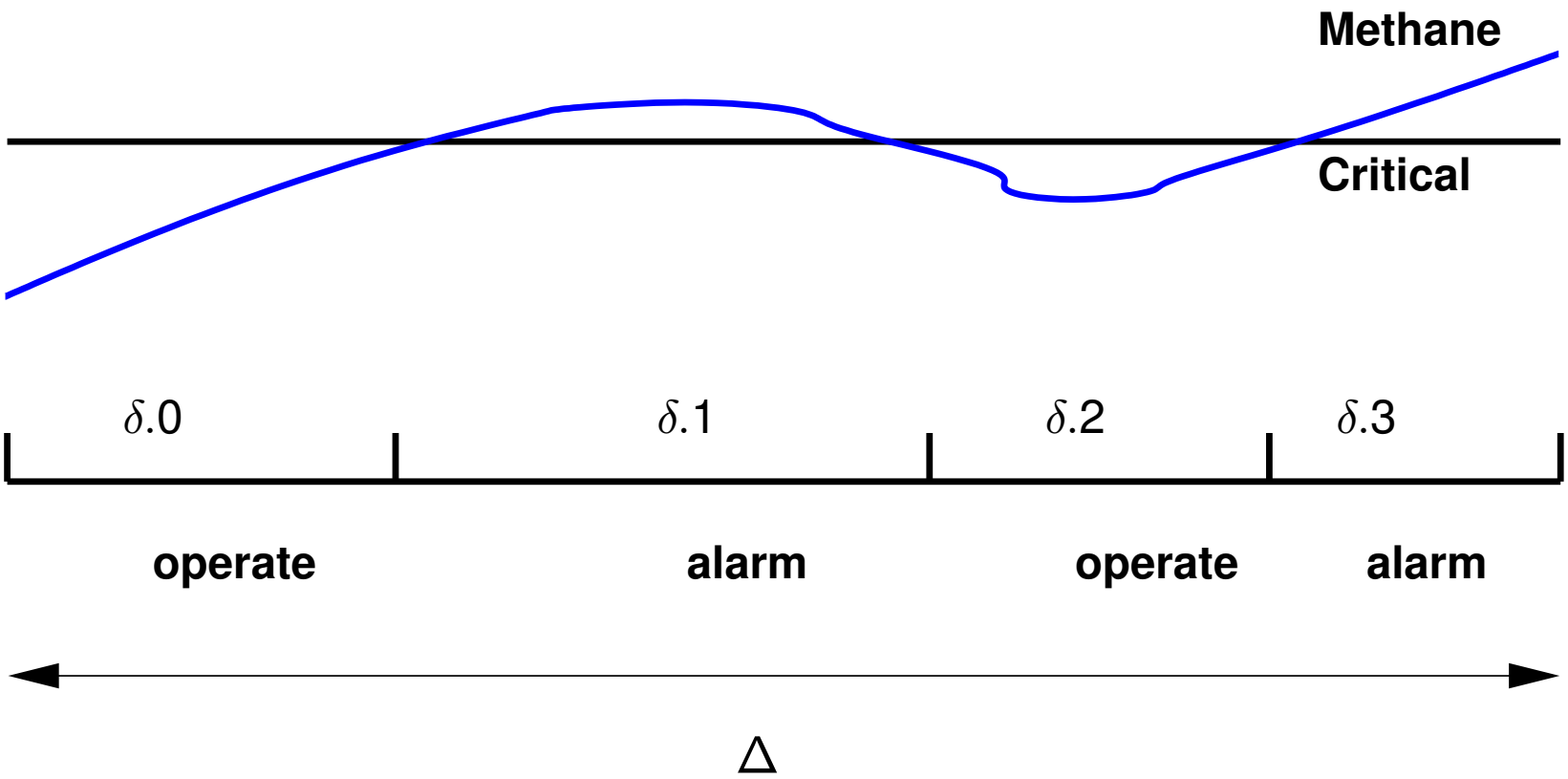
It is not an Action System

Primitive durative actions

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.

Durative actions



Expanded Program

The above is equivalent to the following,

mine_pump

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

From this it is easy 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: $\mathit{methane} < \mathit{Critical} \wedge \mathit{Low} < \mathit{water}$.

Rely/Guarantee for the Pump

When the primitive action pump is active it guarantees

Pump guarantee

$$g\_pump \mathrel{\widehat{=}} \Box(MinOut \leq water\_out \wedge pump\_active)$$

but relies on

Pump rely

$$r\_pump \mathrel{\widehat{=}} \Box(Low < water \wedge methane < Critical)$$

If c is a predicate (condition) on the state of the program variables, we use the notation $\Box c$ as a predicate over a time interval, Δ, that states that condition c holds at all times in Δ, i.e., $(\Box c).\Delta$.

Nondeterminism choice for deterministic procedure

For example, the mine_pump procedure

mine_pump

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

is equivalent to earlier:

mine_pump

$$\begin{array}{rcl} \mathit{Critical} \leq \mathit{methane} & \rightarrow & \text{alarm}, \\ \mathit{true} & \rightarrow & \text{operate} \end{array}$$

Nondeterminism

In general, the guard conditions of a nondeterministic choice do not need to be mutually exclusive. The following

operate specification

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

is refined by earlier:

operate

$$\begin{array}{rcl} \left(\begin{array}{l} \mathit{High} < \mathit{water} \vee \\ \mathit{Low} < \mathit{water} \wedge \mathit{pump\_active} \end{array} \right) & \rightarrow & \text{pump}, \\ \mathit{true} & \rightarrow & \text{nil} \end{array}$$

Conditional

A conditional

$$c0 \rightarrow a0,$$
$$c1 \rightarrow a1$$

is expanded to

$$\begin{array}{rrl} & c0 & \rightarrow a0 \\ \sqcap & \neg c0 \wedge c1 & \rightarrow a1 \\ \sqcap & \neg c0 \wedge \neg c1 & \rightarrow \mathit{chaos} \end{array}$$

in which the guards are mutually exclusive.

Abstract syntax

Definition

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

$$\begin{aligned} GA &::= Pred \rightarrow A \\ A &::= Identifier \mid GA \mid A \sqcap A \mid \mathrm{seq}\,.GA \end{aligned}$$

Timed traces and intervals

- The semantics of a teleo-reactive program are given by specifying its set of behaviours over a time interval corresponding to the interval over which it is active.
- A behaviour is a trace of the values of program's variables over time.

Traces

Let

Var be the set of names of variables,

Value be the universe of values, and

Time be the set of non-negative real numbers.

$$\begin{aligned} \Sigma &\mathrel{\widehat{=}} \mathit{Var} \to \mathit{Value} \\ \mathit{Trace} &\mathrel{\widehat{=}} \mathit{Time} \to \Sigma \\ \mathit{TracePredicate} &\mathrel{\widehat{=}} \mathit{Trace} \to \mathrm{Boolean} \end{aligned}$$

A trace predicate is either true or false for a given trace.

Lifted Predicates

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,

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

Intervals and Lifted Predicates

The set *Interval* contains all the contiguous subsets of *Time*. We also promote the boolean operators one step further to act on the type $(\mathit{Interval} \rightarrow (\mathit{Trace} \rightarrow \mathrm{Boolean}))$, e.g., for time interval Δ,

$$\begin{aligned} (p \wedge q).\Delta &= p.\Delta \wedge q.\Delta \\ (p \wedge q).\Delta.\sigma &= p.\Delta.\sigma \wedge q.\Delta.\sigma \end{aligned}$$

At all points in an interval

Definition (All points)

For a boolean condition, $c \in \Sigma \to \text{Boolean}$, we define the notation "$\Box c$", of type $\mathit{Interval} \to (\mathit{Trace} \to \text{Boolean})$ by

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

$(\Box c).\Delta$ reads "c holds at every time in the interval Δ".

Behaviours

The behaviour of an action over the time interval over which it is active is characterised by an interval trace predicate, i.e., a function that given the time interval over which the action is active gives a predicate defining the possible traces of the action. The function *beh* and maps teleo-reactive actions from the abstract syntax (A) to functions from time intervals to trace predicates.

$$beh : A \rightarrow (Interval \rightarrow (Trace \rightarrow \mathrm{Boolean}))$$

Behaviours of actions

For a state predicate c; actions a, a_0, a_1; and a time interval Δ

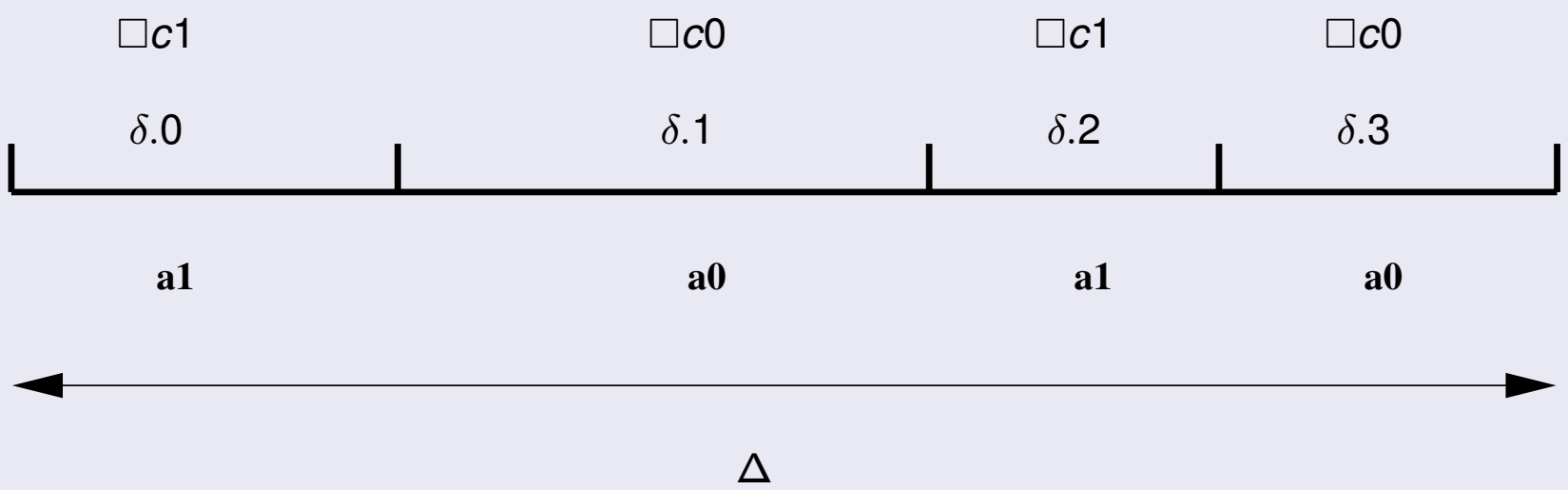
beh

$$beh.chaos.\Delta \;\widehat{=}\; True \tag{1}$$
$$beh.(c \rightarrow a).\Delta \;\widehat{=}\; (\Box c).\Delta \wedge beh.a.\Delta \tag{2}$$
$$\tag{3}$$

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

Behaviour of conditional

A partition for "$c0 \rightarrow a0, c1 \rightarrow a1$"



We use the notation *part*.Δ 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.

Semantics of conditional

Consider the conditional

$$a \mathrel{\widehat{=}} c_0 \rightarrow a_0, c_1 \rightarrow a_1, \ldots, c_{n-1} \rightarrow a_{n-1}$$

The effective guard of action a_i is the explicit guard on action a_i (i.e., c_i) conjoined with the negation of all earlier guards.

$$\hat{c}_i \mathrel{\widehat{=}} c_i \wedge (\forall j : 0..i-1 \bullet \neg c_j)$$

We define $c_n \mathrel{\widehat{=}} \mathit{true}$, so that $\hat{c}_n = (\forall j : 0..n-1 \bullet \neg c_j)$, and we define $a_n \mathrel{\widehat{=}} \mathit{chaos}$.
The action a above is equivalent to

$$\hat{c}_0 \rightarrow a_0, \hat{c}_1 \rightarrow a_1, \ldots, \hat{c}_n \rightarrow a_n$$

A behaviour of a conditional over an interval Δ is constructed by partitioning Δ into a sequence of subintervals δ such that an action of the conditional is active over a subinterval δ_i only if its effective guard holds over δ_i.
The sequence s defines the corresponding sequence of indices of actions, so that action a_{s_i} takes place over subinterval δ_i.
The subintervals δ_i should be maximal, and hence we require s has no immediate repetitions $s \in \mathit{noreps}(0..n)$.

beh

$$\begin{aligned} \mathit{beh}.a.\Delta \quad \widehat{=} \quad & \exists \delta : \mathit{part}.\Delta \bullet \\ & \exists s : \mathit{noreps}.(0..n) \bullet \mathrm{dom}\,.s = \mathrm{dom}\,.\delta \wedge \\ & \forall i : \mathrm{dom}\,.\delta \bullet \mathit{beh}.(\hat{c}_{s_i} \rightarrow a_{s_i}).(\delta_i) \end{aligned} \tag{4}$$

Properties of behaviours

Theorem

For a condition, c, action a, and sequence of guarded actions s,

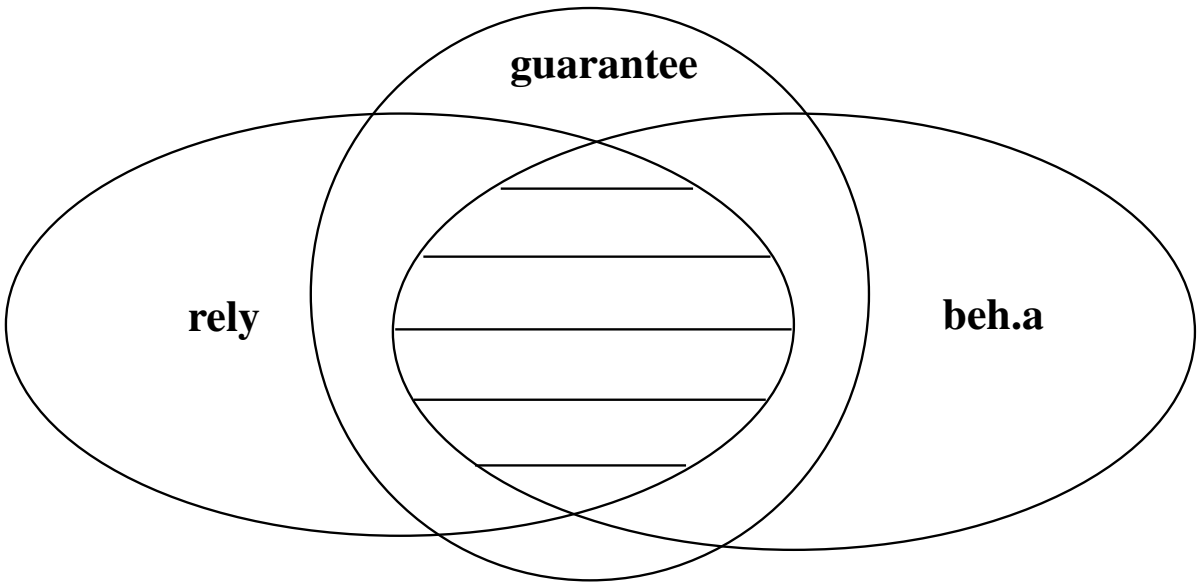
$$
\begin{aligned}
c \to a, s &= c \to a, \neg c \to s \\
c_0 \to (c_1 \to a_1, s) &= \begin{pmatrix} c_0 \wedge c_1 \to a_1, \\ c_0 \wedge \neg c_1 \to s \end{pmatrix}
\end{aligned}
$$

Reasoning About T-R Programs

We require the rely and guarantee conditions to be predicates of the interval over which the procedure operates, i.e., the rely and guarantee are functions from time intervals to trace predicates, i.e., of type

$$\mathit{Interval} \rightarrow (\mathit{Trace} \rightarrow \mathrm{Boolean})$$

Satisfying a Rely/Guarantee Pair



Definition (Satisfies)

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, i.e., $beh.a \wedge r \Rrightarrow g$

Basic Rely/Guarantee Theorem

Theorem (Basic rely/guarantee)

For rely conditions rs and rw, guarantee conditions gs and gw, and an action a, if

$$rs \Rrightarrow rw$$
$$\{rw\}\ a\ \{gs\}$$
$$gs \wedge r \Rrightarrow gw$$

then

$$\{rs\}\ a\ \{gw\}$$

Guarded Actions

Theorem (Guarded action)

For a rely condition r, a guarantee condition g, a predicate c, and an action a,

$$\{r\}\ c \to a\ \{g\} \equiv \{r \wedge \Box c\}\ a\ \{g\}$$

Decomposing (Rely) Conditions

Definition (Decomposes)

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

$$\forall \Delta, \Delta' : \mathit{Interval} \bullet \Delta' \subseteq \Delta \land r.\Delta \Rrightarrow r.\Delta'$$

For example, $\Box c$ decomposes over intervals,
but $r.\Delta \mathrel{\widehat{=}} \mathit{length}.\Delta \geq 10$ does not.

Composing (Guarantee) Conditions

Definition (Composes)

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

$$\forall \Delta : \mathit{Interval} \bullet \forall \delta : \mathit{part}.\Delta \bullet \\ (\forall i : \mathrm{dom}\,.\delta \bullet g.(\delta.i)) \Rrightarrow g.\Delta$$

For example, $\Box c$ composes over intervals,
and hence $\Box(x = 0 \vee x = 1)$ composes
but $\Box(x = 0) \vee \Box(x = 1)$ does not.

Conditional

Theorem

For rely condition r that decomposes over intervals, guarantee condition g that composes over intervals, predicate c, action a, and sequence of guarded actions, s,

$$\{r\}\ c \rightarrow a, s\ \{g\}$$

provided

$$(\{r\}\ c \rightarrow a\ \{g\}) \wedge (\{r\}\ \neg c \rightarrow s\ \{g\}).$$

Two-Branch Conditional

Theorem (Conditional)

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,

$$\{r\}\ c_0 \rightarrow a_0, c_1 \rightarrow a_1\ \{g\}$$

provided

$$\{r \wedge \Box c_0\}\ a_0\ \{g\} \tag{5}$$

$$\{r \wedge \Box(\neg c_0 \wedge c_1)\}\ a_1\ \{g\} \tag{6}$$

$$r \Rrightarrow \Box(c_0 \vee c_1) \tag{7}$$

The above theorem can be generalised.

Example: Mine Pump

$water$	the level of water in the mine shaft
$\frac{dwater}{dt}$	the derivative of the water level with respect to time, i.e., the rate of change of the water level.
$water\_in$	the rate of flow of water into the mine
$water\_out$	the rate of flow of water out of the mine

Rely condition for mine pump system

$$r \mathrel{\widehat{=}} \Box \left(\begin{array}{l} \frac{dwater}{dt} = water\_in - water\_out \wedge \\ 0 \leq water\_out \wedge \\ 0 \leq water\_in \leq MaxIn \end{array} \right)$$

Pump

When the primitive action pump is active it guarantees

Guarantee for pump

$$g\_pump \mathrel{\widehat{=}} \Box(MinOut \leq water\_out \wedge pump\_active)$$

but relies on

Rely for pump

$$r\_pump \mathrel{\widehat{=}} \Box(Low < water \wedge methane < Critical)$$

Guarantee

Guarantee for the mine pump system

$$g \mathrel{\widehat{=}} \Box \left(\begin{array}{l} methane < Critical \wedge High < water \\ \Longrightarrow \\ \frac{dwater}{dt} \leq MaxIn - MinOut \end{array} \right)$$

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

Mine Pump Rely/Guarantee

mine_pump

$$\begin{array}{lcl} \mathit{Critical} \leq \mathit{methane} & \rightarrow & \text{alarm}, \\ \mathit{true} & \rightarrow & \text{operate} \end{array}$$

satisfies the rely/guarantee pair, i.e.,

$$\{r\}\ \text{mine\_pump}\ \{g\}$$

provided

$$\{r \wedge \Box(\mathit{Critical} \leq \mathit{methane})\}\ \text{alarm}\ \{g\}$$
$$\{r \wedge \Box(\mathit{methane} < \mathit{Critical})\}\ \text{operate}\ \{g\}$$
$$r \Rrightarrow \Box(\mathit{Critical} \leq \mathit{methane} \vee \mathit{true})$$

$$g \mathrel{\widehat{=}} \left(\begin{array}{l} methane < Critical \wedge High < water \implies \\ \frac{dwater}{dt} + MinOut \leq MaxIn \end{array} \right)$$

$$\{r \wedge \Box(Critical \leq methane)\} \text{ alarm } \{g\} \quad (8)$$

$$\{r \wedge \Box(methane < Critical)\} \text{ operate } \{g\} \quad (9)$$

$$r \Rrightarrow \Box(Critical \leq methane \vee true) \quad (10)$$

Requirement (10) holds trivially.
Requirement (8) holds because $Critical \leq methane$ is the complement of $methane < Critical$ used on the left side of the implication within g.

For requirement (9) we need to show procedure operate

operate

$$\begin{array}{ll} \left(\begin{array}{l} \mathit{High} < \mathit{water} \vee \\ \mathit{Low} < \mathit{water} \wedge \mathit{pump\_active} \end{array}\right) & \rightarrow \ \text{pump}, \\ \mathit{true} & \rightarrow \ \text{nil} \end{array}$$

satisfies

$$\{r \wedge \Box(\mathit{methane} < \mathit{Critical})\} \ \text{operate} \ \{g\}$$

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

$$\{r \wedge \Box(c \wedge \mathit{methane} < \mathit{Critical})\} \ \text{pump} \ \{g\}$$
$$\{r \wedge \Box(\neg c \wedge \mathit{methane} < \mathit{Critical})\} \ \text{nil} \ \{g\}$$
$$r \wedge \Box(\mathit{methane} < \mathit{Critical}) \Rrightarrow \Box(c \vee \mathit{true})$$

$$c \mathrel{\widehat{=}} (\mathit{High} < \mathit{water}) \vee (\mathit{Low} < \mathit{water} \wedge \mathit{pump\_active})$$

$$\{r \wedge \Box(c \wedge \mathit{methane} < \mathit{Critical})\}\ \mathrm{pump}\ \{g\} \quad (11)$$

$$\{r \wedge \Box(\neg c \wedge \mathit{methane} < \mathit{Critical})\}\ \mathrm{nil}\ \{g\} \quad (12)$$

$$\Box(\mathit{methane} < \mathit{Critical}) \wedge r \Rrightarrow \Box(c \vee \mathit{true}) \quad (13)$$

Requirement (13) is trivial.
For (12), the negation of c implies that $\mathit{water} \leq \mathit{High}$, which implies the left side of the implication in g is false.
Requirement (11) follows from the rely and guarantee conditions of pump, provided

$$r \wedge \Box(c \wedge \mathit{methane} < \mathit{Critical}) \Rightarrow r\_pump$$
$$g\_pump \wedge r \wedge \Box(c \wedge \mathit{methane} < \mathit{Critical}) \Rightarrow g$$

$$c \mathrel{\widehat{=}} (\mathit{High} < \mathit{water}) \vee (\mathit{Low} < \mathit{water} \wedge \mathit{pump\_active})$$

$$\begin{array}{rcl} r \wedge \Box(c \wedge \mathit{methane} < \mathit{Critical}) & \Rightarrow & \mathit{r\_pump} \\ \mathit{g\_pump} \wedge r \wedge \Box(c \wedge \mathit{methane} < \mathit{Critical}) & \Rightarrow & g \end{array}$$

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

$$\begin{array}{l} \frac{\mathit{dwater}}{\mathit{dt}} + \mathit{water\_out} = \mathit{water\_in} \\ \Rightarrow \text{ as } \mathit{g\_pump} \text{ implies } \mathit{MinOut} \leq \mathit{water\_out} \\ \frac{\mathit{dwater}}{\mathit{dt}} + \mathit{MinOut} \leq \mathit{water\_in} \\ \Rightarrow \text{ as } r \text{ implies } \mathit{water\_in} \leq \mathit{MaxIn} \\ \frac{\mathit{dwater}}{\mathit{dt}} + \mathit{MinOut} \leq \mathit{MaxIn} \end{array}$$

Action System

Action System

initially $alarm := \text{Off} \parallel pump := \text{Off};$
do $alarm = \text{Off} \wedge methane < \text{Critical} \wedge$
$\quad \text{High} \leq water \wedge pump = \text{Off} \rightarrow pump := \text{On}$
$[]\ alarm = \text{Off} \wedge methane < \text{Critical} \wedge$
$\quad water \leq \text{Low} \wedge pump = \text{On} \rightarrow pump := \text{Off}$
$[]\ alarm = \text{Off} \wedge methane < \text{Critical} \wedge$
$\quad \text{Low} < water < \text{High} \rightarrow \textbf{skip}$
$[]\ alarm = \text{Off} \wedge \text{Critical} \leq methane \rightarrow$
$\quad alarm := \text{On} \parallel pump := \text{Off}$
$[]\ alarm = \text{On} \wedge \text{Critical} \leq methane \rightarrow \textbf{skip}$
$[]\ alarm = \text{On} \wedge methane < \text{Critical} \rightarrow alarm := \text{Off}$
od

This is not a Teleo-reactive Program

Action System Invariants

This action system has some interesting invariants.

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

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.

Conclusions

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.

A. Burns and G. Baxter.
Time bands in systems structure.
In D. Besnard, C. Gacek, and C.B. Jones, editors, *Structure for Dependability: Computer-Based Systems from an Interdisciplinary Perspective*, pages 74–88.
Springer-Verlag, 2006.

A. Burns and A.M. Lister.
A framework for building dependable systems.
Computer Journal, 34(2):173–181, 1991.

M. Joseph, editor.
Real-time Systems: Specification, Verification and Analysis.
Prentice Hall, 1996.

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.


N. Nilsson.
Teleo-reactive programs for agent control.
Journal of Artificial Intelligence Research, 1:139–158, 1994.


N. J. Nilsson.
Teleo-reactive programs and the triple-tower architecture.
Electronic Transactions on Artificial Intelligence, 5:99–110, 2001.


N Nilsson.
Teleo-reactive programming web site, Accessed 28 August 2008.
http://robotics.stanford.edu/users/nilsson/trweb/tr.html.

Acknowledgments

This research was supported, in part, by Australian Research Council (ARC) Discovery Grant DP0987452 *Combining Time Bands and Teleo-Reactive Programs for Advanced Dependable Real-Time Systems*, and the EPSRC-funded *Trustworthy Ambient Systems (TrAmS)* Platform Project. I would like to thank This research has benefited from my collaborations with Alan Burns, Keith Clark, Brijesh Dongol, Cliff Jones, Peter Robinson and members of IFIP Working Group 2.3 on Programming Methodology.