Calculational Verification of Reactive Programs with Reactive Relations and Kleene Algebra

Simon Foster, Kangfeng Ye, Ana Cavalcanti, Jim Woodcock

Monday 29^{th} October, 2018

Outline

Outline

Reactive Programs

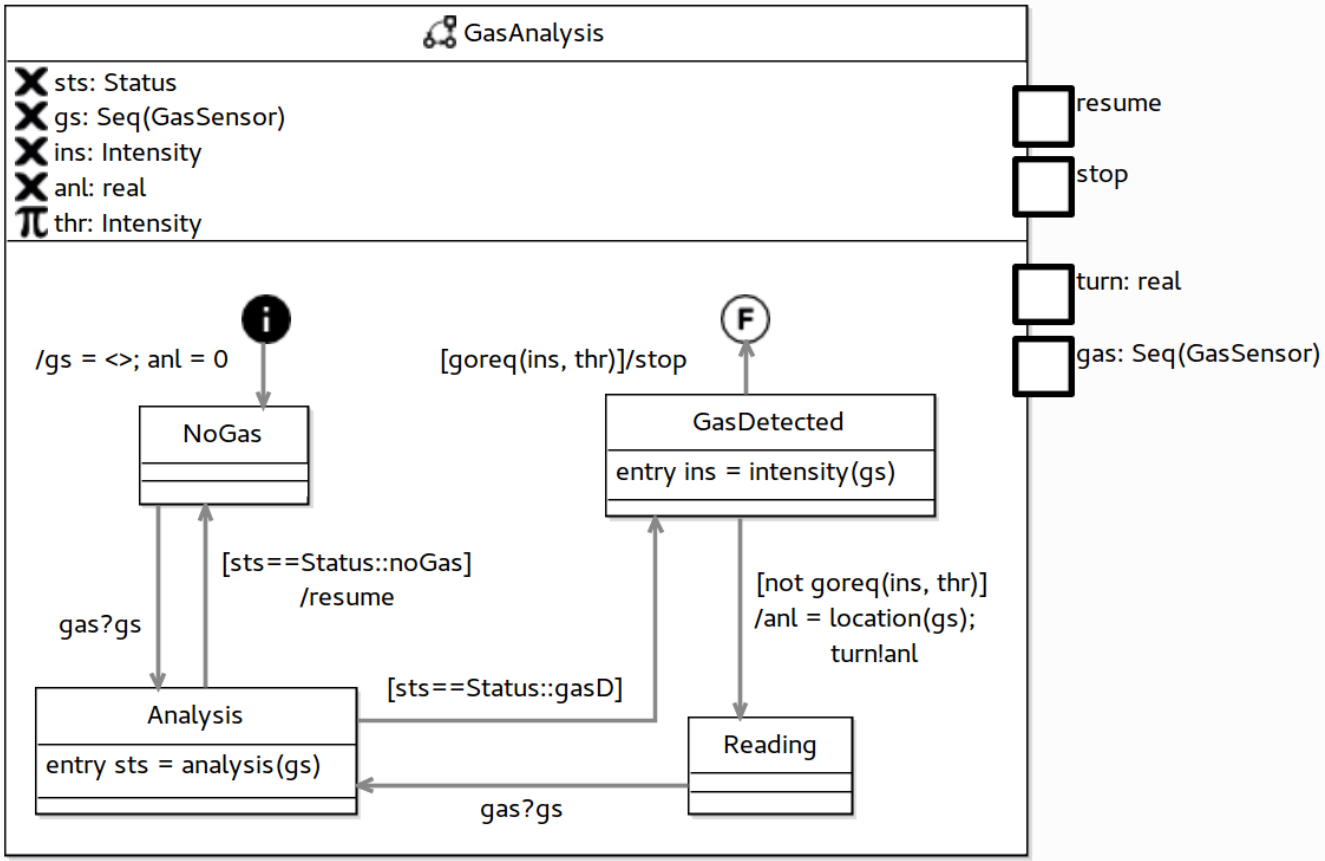
- sequential programs calculate a final state from an initial state
- reactive programs pause for interaction with the environment
- the environment can direct the choices an agent makes (event driven)

Example (CSP Process)

$$(a \rightarrow \textbf{\textit{Skip}}) \Box (b \rightarrow c \rightarrow \textbf{\textit{Stop}})$$

- one terminated observation, following occurrence of event a
- three quiescent observations:
  1. willing to accept an a or b event
  2. willing to accept a c having done b
  3. deadlocked, accepting nothing but not terminated
- this paper concerns theorem proving for reactive programs

RoboChart State Machines



- graphical statechart-like notation for robotic controllers
- developed on the RoboCalc project at University of York
- state machines communicate via (a)synchronous channels
- denotational semantics based on reactive programs

Semantics of Reactive Programs

- CSP denotational semantics based on the failures-divergences model
- a relational semantics for CSP was given in Hoare and He's Unifying Theories of Programming semantic framework
- based on a UTP theory called reactive processes
  - relational variables $wait$, tr, and ref encode traces and failures
  - healthiness conditions ensure well-formedness of observations
  - processes as healthy fixed-points of idempotent function ***CSP***(P)
- UTP theory extended with mutable state variables (*Circus* family)

Verification of Reactive Programs

- relational semantics can be applied to verification
- verification of reactive programs using our tool, Isabelle/UTP
  - relational semantics used to derive verification conditions
  - discharge using Isabelle tactics (sledgehammer etc. – cf. Struth et al.)
  - however: underlying relations are extremely complicated
- this paper: efficient verification of reactive programs

Outline

Reactive Programming Operators

- $x := v$ – assignment
- $P \fatsemi Q$ – sequential composition
- $c?x$ – await input on channel c; place value in x
- $c!v$ – output v on channel c
- $b \mathbin{\&} P$ – deadlocks when $\neg\ b$, otherwise behaves as P
- $P \Box Q$ – external choice; resolves when P or Q performs an event
- $b \circledast P$ – iteration of P with condition b
- $[b]$ – assumption; miraculous when $\neg b$

Example (Reactive Buffer)

$$buf := \langle\rangle \fatsemi true \circledast \left(\begin{array}{l} inp?x \fatsemi buf := buf \frown \langle x \rangle \Box \\ \\ \#buf > 0 \mathbin{\&} out!hd(buf) \fatsemi buf := tl(buf) \end{array} \right)$$

where $buf : \mathrm{seq}\,\mathbb{N}$ and $inp, out : \mathbb{N} \to \mathcal{E}$

Reactive Specifications

Stateful Failure Reactive Contract

$$\left[\, P_1(tt, st) \vdash P_2(tt, st, ref') \;\middle|\; P_3(tt, st, st') \,\right]$$

where $tt : \mathrm{seq}\, \mathcal{E}$, $ref : \mathbb{P}\, \mathcal{E}$, and $st, st' : \Sigma$; for event set $\mathcal{E}$ and state space Σ

- specifies behaviour with three "reactive predicates":
  - precondition $P_1(tt, st)$ – permissible initial state and trace
  - pericondition $P_2(tt, st, ref')$ – quiescent observations
  - postcondition $P_3(tt, st, st')$ – terminated observations

Example (Vending Machine Contract)

$$\left[\, coins = 0 \;\middle|\; \begin{array}{l} tt \in \langle choc \rangle^* \\ \land\ choc, off \notin ref' \end{array} \;\middle|\; \exists\, n : \mathbb{N} \bullet \left(\begin{array}{l} tt = \langle choc \rangle^n \frown \langle off \rangle \\ \land\ coins' = n \end{array} \right) \right]$$

- reactive contracts are fixed-points of ***NCSP***
- reactive relations are fixed-points of ***RR***

Reactive Programs as Contracts

- also use reactive contracts to denote programs

Definition (Event Prefix and Assignment)

$$c!v \triangleq \left[\mathit{true}_r \vdash tt = \langle\rangle \wedge c.v \notin \mathit{ref}' \;\middle|\; tt = \langle c.v \rangle \wedge \mathit{st}' = \mathit{st} \right]$$

$$x := v \triangleq \left[\mathit{true}_r \vdash \mathit{false} \;\middle|\; \mathit{st}' = \mathit{st} \oplus \{x \mapsto v\} \wedge tt = \langle\rangle \right]$$

Theorem (Reactive Contract Composition)

$$\prod_{i \in I} [P(i) \vdash Q(i) \mid R(i)] = \left[\bigwedge_{i \in I} P(i) \vdash \bigvee_{i \in I} Q(i) \;\middle|\; \bigvee_{i \in I} R(i) \right]$$

$$[P_1 \vdash P_2 \mid P_3] \lhd b \rhd [Q_1 \vdash Q_2 \mid Q_3] = [\, P_1 \lhd b \rhd Q_1 \vdash P_2 \lhd b \rhd Q_2 \mid P_3 \lhd b \rhd Q_3 \,]$$

$$[P_1 \vdash P_2 \mid P_3] \fatsemi [Q_1 \vdash Q_2 \mid Q_3] = [P_1 \wedge (P_3 \,\mathit{wp}\, Q_1) \vdash P_2 \vee (P_3 \fatsemi Q_2) \mid P_3 \fatsemi Q_3 \,]$$

- we can thus calculate the contract of a reactive program
- depends on a suitable equational theory for the reactive relations

Verification Technique

- verification of properties expressed using refinement:

$$[\, S_1 \vdash S_2 \mid S_3 \,] \sqsubseteq P$$

- reactive contract S specifies implementation P
- automated verification in Isabelle/UTP follows three steps:
  1. calculate reactive contract for program: $P \Longrightarrow [P_1 \vdash P_2 \mid P_3]$
  2. show $S \sqsubseteq P$ by $(S_1 \Rightarrow P_1)$, $(S_1 \wedge P_2) \Rightarrow S_2$, and $(S_1 \wedge P_3) \Rightarrow S_3$
  3. discharge three proof obligations with rel-auto and sledgehammer
- allows model checking of reactive programs with infinite state
- requires equational laws to calculate peri- and postconditions

Contributions of this paper

- an equational theory for stateful-failure reactive relations
- Kleene-algebra based analysis of iterative reactive programs
- verification technique mechanised in Isabelle/UTP

Outline

Stateful-Failure Reactive Relations

Example

The postcondition of $inp?x \fatsemi buf := buf \frown \langle x \rangle$ is $(\bigvee v : \mathbb{N} \bullet \mathit{tt} = \langle inp.v \rangle \wedge \mathit{st}' = \mathit{st} \oplus \{x \mapsto v\}) \fatsemi (\mathit{st}' = \mathit{st} \oplus \{buf \mapsto buf \frown \langle x \rangle\} \wedge \mathit{tt} = \langle\rangle)$

- how do we automatically simplify this relation?
- need higher level patterns for quiescent and terminated observations

Definition (Reactive Relational Operators)

$$\mathcal{E}[b(\mathit{st}), t(\mathit{st}), E(\mathit{st})] \triangleq \mathbf{RR}(b(\mathit{st}) \wedge \mathit{tt} = t(\mathit{st}) \wedge (\forall\, e \in E(\mathit{st}) \bullet e \notin \mathit{ref}'))$$

$$\Phi[b(\mathit{st}), \sigma, t(\mathit{st})] \triangleq \mathbf{RR}(b(\mathit{st}) \wedge \mathit{st}' = \sigma(\mathit{st}) \wedge \mathit{tt} = t(\mathit{st}))$$

where $b : \Sigma \to \mathbb{B}$, $t : \Sigma \to \mathrm{seq}\, \mathcal{E}$, $E : \Sigma \to \mathbb{P}\, \mathcal{E}$, and $\sigma : \Sigma \to \Sigma$

- predicative expression of stateful failure traces and terminated traces
- b is a guard on the state, t a trace expression, and E a set of event

Definition (Basic Reactive Operators)

$$x := v \triangleq [\, \textbf{\textit{true}}_r \vdash \textbf{\textit{false}} \mid \Phi[true, \{x \mapsto v\}, \langle\rangle]\,]$$

$$\textbf{\textit{Skip}} \triangleq [\, \textbf{\textit{true}}_r \vdash \textbf{\textit{false}} \mid \Phi[true, id, \langle\rangle]\,]$$

$$c!v \triangleq [\, \textbf{\textit{true}}_r \vdash \mathcal{E}[true, \langle\rangle, \{c.v\}] \mid \Phi[true, id, \langle c.v\rangle]\,]$$

$$c?x \triangleq \left[\, \textbf{\textit{true}}_r \vdash \bigvee k \bullet \mathcal{E}[true, \langle\rangle, \{c.k\}] \;\middle|\; \bigvee k \bullet \Phi[true, id, \langle c.k\rangle]\,\right]$$

$$\textbf{\textit{Stop}} \triangleq [\, \textbf{\textit{true}}_r \vdash \mathcal{E}[true, \langle\rangle, \emptyset] \mid \textbf{\textit{false}}\,]$$

Example

$$inp?x \fatsemi buf := buf \frown \langle x\rangle \;=\; [\, \textbf{\textit{true}}_r \vdash peri \mid post\,]$$

where

$$peri \triangleq \mathcal{E}[\textit{true}, \langle\rangle, \{inp.v \mid v \in \mathbb{N}\}]$$

$$post \triangleq \bigvee v \in \mathbb{N} \bullet \Phi[\textit{true}, \langle inp.v\rangle, \{buf \mapsto buf \frown \langle x\rangle\}]$$

Equational Theory

Theorem (Reactive Relational Compositions)

$$[b] \fatsemi P = \Phi[b, id, \langle\rangle] \fatsemi P$$

$$\Phi[b_1, \sigma_1, t_1] \fatsemi \Phi[b_2, \sigma_2, t_2] = \Phi[b_1 \wedge \sigma_1 \dagger b_2, \sigma_2 \circ \sigma_1, t_1 \frown \sigma_1 \dagger t_2]$$

$$\Phi[b_1, \sigma_1, t_1] \fatsemi \mathcal{E}[b_2, t_2, E] = \mathcal{E}[b_1 \wedge \sigma_1 \dagger b_2, t_1 \frown \sigma_1 \dagger t_2, \sigma_1 \dagger E]$$

$$\Phi[b_1, \sigma_1, t_1] \lhd c \rhd \Phi[b_2, \sigma_2, t_2] = \Phi[b_1 \lhd c \rhd b_2, \sigma_1 \lhd c \rhd \sigma_2, t_1 \lhd c \rhd t_2]$$

$$\mathcal{E}[b_1, t_1, E_1] \lhd c \rhd \mathcal{E}[b_2, t_2, E_2] = \mathcal{E}[b_1 \lhd c \rhd b_2, t_1 \lhd c \rhd t_2, E_1 \lhd c \rhd E_2]$$

$$\left(\bigwedge_{i \in I} \mathcal{E}[b(i), t, E(i)]\right) = \mathcal{E}\left[\bigwedge_{i \in I} b(i), t, \bigcup_{i \in I} E(i)\right]$$

- allows automatic calculation of the peri- and postconditions
- use of Isabelle/HOL simplifier to apply these laws

Outline

Kleene Algebra Laws for Reactive Programs

Definition

A Kleene UTP theory $(\mathbf{H}, \mathit{II}_{\mathbf{H}})$ satisfies the following conditions:

1. $\mathbf{H}$ is idempotent and continuous
2. $\mathbf{H}$ is closed under sequential composition
3. identity $\mathit{II}_{\mathbf{H}}$ is $\mathbf{H}$-healthy
4. $\mathit{II}_{\mathbf{H}} ; P = P ; \mathit{II}_{\mathbf{H}} = P$, when P is $\mathbf{H}$-healthy
5. $\top_{\mathbf{H}} ; P = \top_{\mathbf{H}}$, when P is $\mathbf{H}$-healthy

Theorem

If $(\mathbf{H}, \mathit{II}_{\mathbf{H}})$ *is a Kleene UTP theory, then* $([\![\mathbf{H}]\!], \sqcap, \top_{\mathbf{H}}, ;, \mathit{II}_{\mathbf{H}}, ^*)$ *forms a weak Kleene algebra with* $P^* \triangleq (\bigsqcap_{i \in \mathbb{N}} P^{i+1}) \sqcap \mathit{II}_{\mathbf{H}}$.

Theorem

$(\mathbf{NCSP}, \mathbf{Skip})$ *and* $(\mathbf{RR}, \mathit{II}_{rea})$ *both form Kleene UTP theories*

Reactive While Loops

Theorem (Kleene Closed Contracts)

$$[\, P \vdash Q \mid R \,]^* \;=\; [\, R^* \;\textbf{wp}\; P \vdash R^* \fatsemi Q \mid R^* \,]$$

Definition (Reactive While)

$$b \circledast P \;\triangleq\; ([b] \fatsemi P)^* \fatsemi [\neg b]$$

Theorem (Contract Iteration)

If R is productive (produces at least one event) then

$$b \circledast [P \vdash Q \mid R] = [\, ([b] \fatsemi R)^* \;\textbf{wp}\; (b \Rightarrow P) \vdash ([b] \fatsemi R)^* \fatsemi [b] \fatsemi Q \mid ([b] \fatsemi R)^* \fatsemi [\neg b] \,]$$

- this characterises both quiescent and terminated loop observations

Refinement Introduction

Theorem

$[I_1 \vdash I_2 \mid I_3] \sqsubseteq b \circledast [Q_1 \vdash Q_2 \mid Q_3]$ *provided that:*

1. *the assumption is weakened* $(([b] \fatsemi Q_3)^*$ ***wp***$(b \Rightarrow Q_1) \sqsubseteq I_1)$*;*
2. *when* b *holds,* Q_2 *establishes the* I_2 *pericondition invariant* $(I_2 \sqsubseteq ([b] \fatsemi Q_2))$ *and,* Q_3 *maintains it* $(I_2 \sqsubseteq [b] \fatsemi Q_3 \fatsemi I_2)$*;*
3. *postcondition invariant* I_3 *is established when* b *is false* $(I_3 \sqsubseteq [\neg b])$ *and* Q_3 *establishes it when* b *is true* $(I_3 \sqsubseteq [b] \fatsemi Q_3 \fatsemi I_3)$*.*

- $I_{1\ldots3}$ are reactive invariant relations; they describe how the trace and state can evolve in each iteration

Outline

Strategy Overview

1. calculate the contract of a reactive program
2. use our equational theory to simplify the underlying reactive relations
3. identify suitable invariants for reactive while loops
4. prove refinements using relational calculus
5. discharge residual verification conditions using sledgehammer

Buffer Verification

Theorem (Deadlock Freedom)

$$\left[\textbf{\textit{true}}_r \mid\!\!\vdash \bigvee_{s,t,E,e} \mathcal{E}[s, t, \{e\} \cup E] \;\middle|\; \textbf{\textit{true}}_r \right] \sqsubseteq Buffer$$

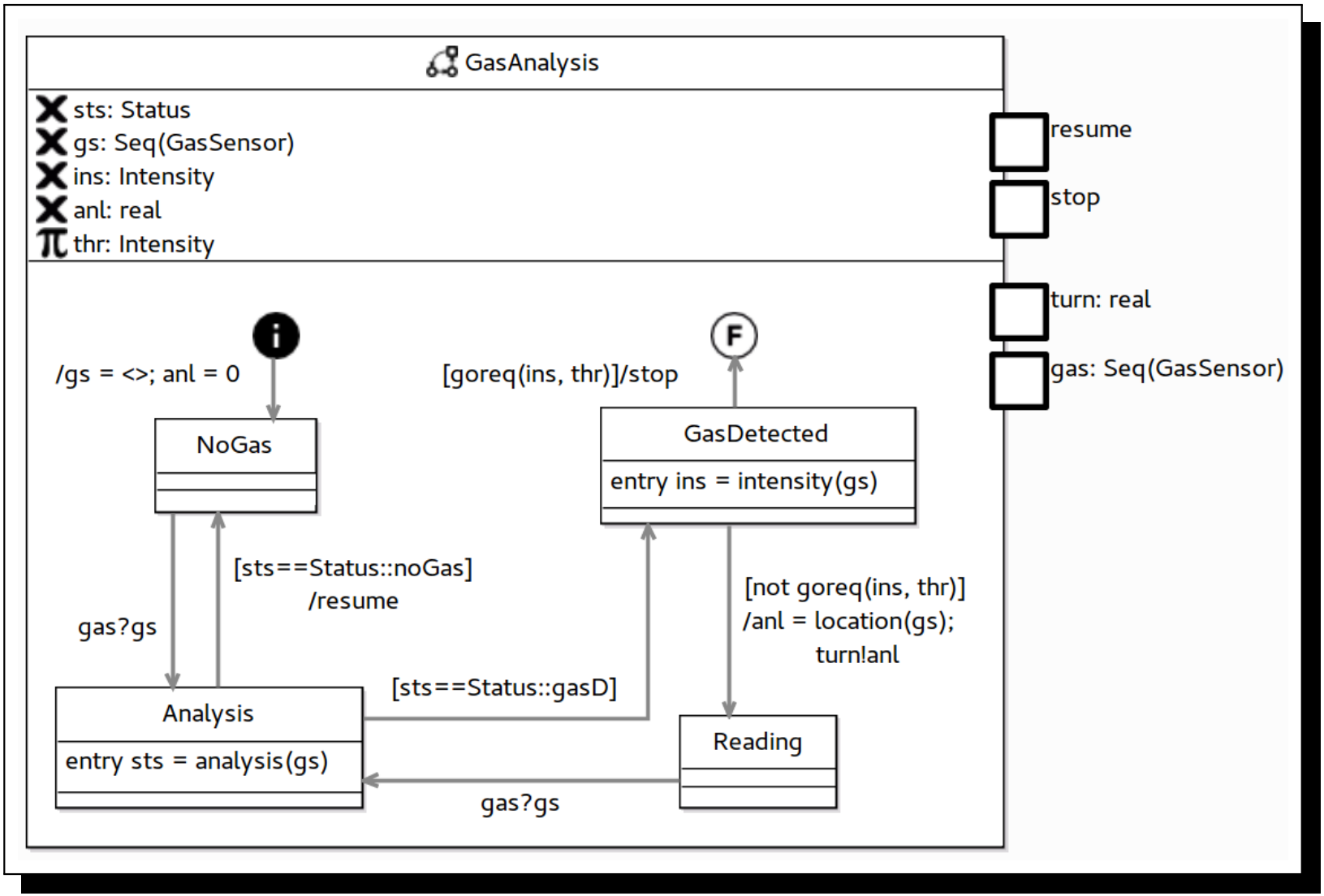
Theorem (Buffer Order Property)

The sequence of items output is a prefix of those that were previously input. This can be formally expressed as

$$[\, \textbf{\textit{true}}_r \mid\!\!\vdash outps(tt) \leq inps(tt) \mid \textbf{\textit{true}}_r \,] \sqsubseteq Buffer$$

where $inps(t), outps(t) : \mathrm{seq}\ \mathbb{N}$ *extract the sequence of input and output elements from the trace* t*, respectively. The postcondition is left unconstrained as* $Buffer$ *does not terminate.*

RoboChart (1) – State Machine Diagram



RoboChart (2) – Isabelle/HOL DSL

```
statemachine GasAnalysis [
  vars sts::Status  gs::"GasSensor list"  ins::Intensity  anl::real
  events resume  stop  turn::real  gas::"GasSensor list"
  states InitState NoGas Reading FinalState
         Analysis: "entry sts := «analysis»(&gs)ₐ"
         GasDetected: "entry ins := «intensity»(&gs)ₐ"
  initial InitState finals FinalState
  transitions
    t1: "from InitState to NoGas action gs := ⟨⟩; anl := 0"
    t2: "from NoGas to Analysis trigger gas?(gs)"
    t3: "from Analysis to NoGas condition &sts =ᵤ «noGas» action resume"
    t4: "from Analysis to GasDetected condition (&sts =ᵤ «gasD»)"
    t5: "from GasDetected to FinalState condition «goreq»(&ins,«thr»)ₐ action stop"
    t6: "from GasDetected to Reading condition ¬ «goreq»(&ins,«thr»)ₐ
         action anl := «location»(&gs)ₐ ; turn!(&anl)"
    t7: "from Reading to Analysis trigger gas?(gs)" ]
```

RoboChart (3) – Reactive Program

$$
\begin{array}{l}
actv := InitState ⨟ \\
\textbf{do} \\
\quad actv = InitState \rightarrow \epsilon ⨟ \mathbf{r{:}}gs := \langle\rangle ⨟ \mathbf{r{:}}anl := 0 ⨟ actv := NoGas \\
\mid actv = NoGas \rightarrow gas?\mathbf{r{:}}gs ⨟ actv := Analysis \\
\mid actv = Analysis \rightarrow \\
\quad \mathbf{r{:}}sts := analysis(\mathbf{r{:}}gs) ⨟ \left(\begin{array}{l} \mathbf{r{:}}sts = noGas \;\&\; \epsilon ⨟ resume ⨟ actv := NoGas \\ \Box\ \mathbf{r{:}}sts = gasD \;\&\; \epsilon ⨟ actv := GasDetected \end{array} \right) \\
\mid actv = GasDetected \rightarrow \mathbf{r{:}}ins := intensity(\mathbf{r{:}}gs) ⨟ \\
\qquad \left(\begin{array}{l} goreq(ins, thr) \;\&\; \epsilon ⨟ stop ⨟ actv := FinalState \\ \Box\ (\neg goreq(ins, thr)) \;\&\; \epsilon ⨟ \begin{array}{l} \mathbf{r{:}}anl := location(\mathbf{r{:}}gs) ⨟ \\ turn!(\mathbf{r{:}}anl) ⨟ actv := Reading \end{array} \end{array} \right) \\
\mid actv = Reading \rightarrow gas?\mathbf{r{:}}gs ⨟ actv := Analysis \\
\textbf{od}
\end{array}
$$

RoboChart (4) – Verification

```
goal (6 subgoals):
 1. dlockf ⊑ ε ; ⟨[&r:gs ↦s ⟨⟩, &r:anl ↦s 0, &rc_ctrl ↦s «''NoGas''»]⟩a
 2. dlockf ⊑
    ((«analysis»(&r:gs)a =u «gasD») &
     ε ; (r:sts), rc_ctrl := «analysis»(&r:gs)a, «''GasDetected''») □
    ((«analysis»(&r:gs)a =u «noGas») &
     ε ; resume ; (r:sts), rc_ctrl := «analysis»(&r:gs)a, «''NoGas''»)
 3. dlockf ⊑
    ((¬ «goreq»((«intensity»(&r:gs)a, «thr»)u)a) &
     ε ; turn!(«location»(&r:gs)a) ;
          ⟨[&r:ins ↦s «intensity»(&r:gs)a, &r:anl ↦s «location»(&r:gs)a, &rc_ctrl ↦s
            «''Reading''»]⟩a) □
    («goreq»((«intensity»(&r:gs)a, «thr»)u)a &
     ε ; stop ; (r:ins), rc_ctrl := «intensity»(&r:gs)a, «''FinalState''»)
```

Outline

Conclusions

- verification technique for reactive programs utilising reactive contracts
- equational theory for simplifying stateful-failure reactive relations
- integration of Kleene algebra reasoning in reactive contracts
- verification strategy in Isabelle/UTP
- enables checking of properties in infinite state reactive programs

Future Work

- extend our calculation strategy to parallel composition
- apply it to more substantial examples
- infinite trace models
- integration with real-time, probabilistic, and hybrid computational behaviours

Resources and Publications

- Isabelle/UTP: https://www.cs.york.ac.uk/circus/isabelle-utp/
- CyPhyAssure: https://www.cs.york.ac.uk/circus/CyPhyAssure/
- RoboCalc: https://www.cs.york.ac.uk/circus/RoboCalc/
- S. Foster, J. Baxter, A. Cavalcanti, A. Miyazawa, J. Woodcock. Automating Verification of State Machines with Reactive Designs and Isabelle/UTP. FACS 2018. LNCS 11222
- S. Foster, A. Cavalcanti, J. Woodcock, F. Zeyda. Unifying Theories of Time with Generalised Reactive Processes. Information Processing Letters, Volume 135. 2018.
- S. Foster, A. Cavalcanti, S. Canham, J. Woodcock, F. Zeyda. Unifying Theories of Reactive Design Contracts. Submitted to Theoretical Computer Science Journal, December 2017. Preprint: https://arxiv.org/abs/1712.10233
- S. Foster, F. Zeyda, J. Woodcock. Unifying Heterogeneous State-Spaces with Lenses. Proc. 13th Intl. Colloq. on Theoretical Aspects of Computing (ICTAC 2016).