INTO-CPS

Foundations for Simulink diagrams in UTP

Unifying Theories of Hybrid and Reactive Programming

Simon Foster Jeremy Jacob
Jim Woodcock Frank Zeyda

University of York, UK
(with thanks to University of Teesside, UK)

Tuesday 30^{th} May, 2017

INTO-CPS

into-cps.au.dk

Horizon 2020 Programme

Outline

Outline

Background

Reactive Systems

Unifying Reactive and Hybrid

Towards Simulink Block Semantics

Conclusion

Unifying Theories of Programming

- modelling languages have heterogeneous semantics
- UTP: framework for definition and study of formal semantics
- based on “programs-as-predicates”; enables unification
- look at different theoretical aspects in isolation
- formalise aspects as UTP theories
  - alphabet – observational variables
  - signature – operators of the language
  - healthiness conditions – define the theory’s domain
- build the theory supermarket – enable reuse
- study relationships between different modelling languages
- unify different semantic presentations
  - denotational, operational, algebraic, axiomatic ...

Programs as Predicates

- observable behaviour encoded in first-order predicate calculus
- relational predicates over input, output variables (x / x')
- alphabet gives the domain of possible observations
- we thus denote the core programming operators

$$\begin{aligned}
x := v &\triangleq x' = v \land y' = y \\
P \,;\, Q &\triangleq \exists\, x_0 \bullet P[x_0/x'] \land Q[x_0/x] \\
P \sqcap Q &\triangleq P \lor Q \\
P \lhd b \rhd Q &\triangleq (b \land P) \lor (\neg b \land Q) \\
P^{\omega} &\triangleq \mu\, X \bullet P \,;\, X
\end{aligned}$$

- definitions support proof of algebraic laws of programming
- natural notion of refinement: $P \sqsubseteq Q \Leftrightarrow P = P \sqcap Q$

UTP theory example

INTO-CPS

- simple language for real-time programs
- **alphabet:** what is observable in particular model
  - clock: $time, time' : \mathbb{N}$
  - program state: $st, st' : \Sigma$
- **healthiness conditions:** $time \leq time'$
  - encoded as idempotent and montonic functions
  - $\textbf{\textit{HT}}(P) = P \wedge time \leq time'$
  - theory elements are fixed-points: $\{P \mid \textbf{\textit{HT}}(P) = P\}$
  - give rise to complete lattices etc.
- **signature:** operators for building programs
  - $\textbf{\textit{Wait}}(n : \mathbb{N}) \mathrel{\widehat{=}} time' = time + n \wedge st' = st$
  - + relational operators ;, $\lhd - \rhd$, $x := v$ etc.
  - ***HT***-healthy relations are closed under these operators
  - e.g. $\textbf{\textit{HT}}(\textbf{\textit{Wait}}(n)) = \textbf{\textit{Wait}}(n)$

Verification

- unified language for both specification and programs
- apply automated theorem provers to verification
- Isabelle/UTP: putting UTP to work
- semantic embedding of UTP in Isabelle/HOL
- syntax to enable description of programs
- specify theories and mechanically prove algebraic laws
- tactics directly leveraging Isabelle's proof automation
- proven laws support verification calculi
- e.g. support for reasoning about CSP and Circus processes
- all laws presented herein have been mechanically verified

Isabelle/UTP time theory example

INTO-CPS

```
theory utp_simple_time imports "../theories/utp_theory" begin

alphabet 's st_time =
  time :: nat  st :: 's

type_synonym 's time_rel = "'s st_time hrel"

definition HT :: "'s time_rel ⇒ 's time_rel" where
[upred_defs]: "HT(P) = (P ∧ $time ≤ᵤ $time´)"

definition Wait :: "nat ⇒ 's time_rel" where
[upred_defs]: "Wait(n) = ($time´ =ᵤ $time + «n» ∧ $st´ =ᵤ $st)"

theorem HT_idem: "HT(HT(P)) = HT(P)" by rel_auto

theorem HT_mono: "P ⊑ Q ⟹ HT(P) ⊑ HT(Q)" by rel_auto

lemma HT_Wait: "HT(Wait(n)) = Wait(n)" by (rel_auto)

lemma HT_seqr_closed:
  "⟦ P is HT; Q is HT ⟧ ⟹ P ;; Q is HT"
  by (rel_auto, meson dual_order.trans) -- {* Sledgehammer required *}

theorem Wait_skip: "Wait(0) = II" by (rel_auto)

theorem Wait_Wait: "Wait(m) ;; Wait(n) = Wait (m + n)" by (rel_auto)
```

Circus Example

INTO-CPS

```
subsection {* Actions *}

text {* The Pay action describes the protocol when a payment of $n$ is requested between two cards,
  $i$ and $j$. It is slightly modified from the paper, as we firstly do not use operations but effect
  the transfer using indexed assignments directly, and secondly because before the transfer can proceed
  we need to check the balance is both sufficient, and that the transfer amount is greater than 0. It
  should also be noted that the indexed assignments give rise to preconditions that the list is
  defined at the given index. In other words, the given card records must be present. *}

definition Pay :: "index ⇒ index ⇒ money ⇒ action_mdx" where
"Pay i j n =
  pay.(«i»).(«j»).(«n») ⟶
   ((reject.(«i») ⟶ Skip)
     ◁ «i» =ᵤ «j» ∨ «n» ≤ᵤ 0 ∨ «n» >ᵤ &valueseq(«i»)ᵤ ▷R
   ({valueseq[«i»]} :=c (&valueseq(«i»)ᵤ - «n») ;;
    {valueseq[«j»]} :=c (&valueseq(«j»)ᵤ + «n») ;;
    accept.(«i») ⟶ Skip))"

definition PaySet :: "index ⇒ (index × index × money) set" where
[upred_defs]: "PaySet cardNum = {(i ,j,k). i < cardNum ∧ j < cardNum ∧ i ≠ j}"

definition AllPay :: "index ⇒ action_mdx" where
"AllPay cardNum = (⨅ (i, j, n) ∈ PaySet cardNum • Pay i j n)"

text {* The Cycle action just repeats the payments over and over for any extant and different card
  indices. In order to be well-formed we require that $cardNum \ge 2$. *}
```

Outline

Background

Reactive Systems

Unifying Reactive and Hybrid

Towards Simulink Block Semantics

Conclusion

Reactive Processes

- UTP theory for description of reactive programs
- basis of languages such as CSP and Circus
- imperative programs have initial and final states
- reactive programs additionally have intermediate states
- intermediate programs await interaction with the environment
- observational variables:
  - $wait, wait' : \mathbb{B}$ – distinguish intermediate and final states
  - $tr, tr' : \operatorname{seq} \mathrm{Event}$ – discrete history of interaction
- portion of trace contributed by present process: $\boldsymbol{tt} \mathrel{\widehat{=}} tr' - tr$
- **example**: $\boldsymbol{do}(a) \mathrel{\widehat{=}} I\!I \lhd wait \rhd (tr' = tr \frown \langle a \rangle \land \neg wait')$

Reactive Healthiness Conditions

INTO-CPS

$$\mathbf{R1}(P) \mathrel{\widehat=} P \land tr \le tr'$$

$$\mathbf{R2}_c(P) \mathrel{\widehat=} P[\epsilon, \mathbf{tt}/tr, tr'] \lhd tr \le tr' \rhd P$$

$$\mathbf{R3}(P) \mathrel{\widehat=} \mathit{I\!I} \lhd wait \rhd P$$

$$\mathbf{R} \mathrel{\widehat=} \mathbf{R3} \circ \mathbf{R2}_c \circ \mathbf{R1}$$

- ***R1***: trace is monotonically increasing
- ***R2***$_c$: no dependence on trace history
- ***R3***: if predecessor is waiting then do nothing
- ***R*** is closed under relational calculus operators (e.g. $;$ and $\sqcap$)

Reactive Design Contracts

INTO-CPS

- building on reactive processes we have reactive designs
- $ok, ok' : \mathbb{B}$ distinguish possible erroneous behaviour
- a reactive design is a triple: $[\, \mathit{pre} \vdash \mathit{peri} \mid \mathit{post} \,]$
  - precondition (***pre***) is a predicate encoding assumptions on state and environment required for correct execution
  - pericondition (***peri***) encodes commitments on trace that are satisfied by all intermediate states
  - postcondition (***post***) encodes commitments on trace and variables that are satisfied in all final states
- contractual specifications have a natural notion of refinement:

$$\frac{P_1 \Rightarrow Q_1 \qquad Q_2 \land P_1 \Rightarrow P_2 \qquad Q_3 \land P_1 \Rightarrow P_3}{[\, P_1 \vdash P_2 \mid P_3 \,] \sqsubseteq [\, Q_1 \vdash Q_2 \mid Q_3 \,]}$$

- objective: use reactive designs to encode Simulink blocks

Reactive Design Examples

INTO-CPS

- CSP examples use $ref' : \mathbb{P}\,\mathrm{Event}$ to encode refusals

$$a \to \textbf{\textit{Skip}} = \left[\, \textbf{\textit{true}} \vdash a \notin ref' \wedge \textbf{\textit{tt}} = \langle\rangle \mid st' = st \wedge \textbf{\textit{tt}} = \langle a \rangle \,\right]$$

$$\textbf{\textit{Stop}} = \left[\, \textbf{\textit{true}} \vdash \textbf{\textit{tt}} = \langle\rangle \mid \textbf{\textit{false}} \,\right]$$

$$a \to \textbf{\textit{Chaos}} \,\square\, b \to \textbf{\textit{Skip}} =$$

$$\left[\, \neg(\langle a \rangle \leq \textbf{\textit{tt}}) \vdash \textbf{\textit{tt}} = \langle\rangle \wedge a \notin ref' \wedge b \notin ref' \mid \textbf{\textit{tt}} = \langle b \rangle \wedge st' = st \,\right]$$

$$\textbf{\textit{Chaos}} = \left[\, \textbf{\textit{false}} \vdash \textbf{\textit{false}} \mid \textbf{\textit{false}} \,\right]$$

$$I\!I_{\textbf{\textit{R}}} = \left[\, \textbf{\textit{true}} \vdash \textbf{\textit{false}} \mid I\!I \,\right]$$

Laws of reactive designs

$$[\, P_1 \vdash P_2 \mid P_3 \,] \sqcap [\, Q_1 \vdash Q_2 \mid Q_3 \,] = [\, P_1 \wedge Q_1 \vdash P_2 \vee Q_2 \mid P_3 \vee Q_3 \,]$$

$$\textbf{\textsf{Chaos}} \sqcap P = \textbf{\textsf{Chaos}}$$

$$\bigsqcap_{i \in I} [\, P_1(i) \vdash P_2(i) \mid P_3(i) \,] = \left[\bigwedge_{i \in I} P_1(i) \,\middle|\!- \bigvee_{i \in I} P_2(i) \,\middle|\, \bigvee_{i \in I} P_3(i) \right]$$

$$[\, P_1 \vdash P_2 \mid P_3 \,] ; [\, Q_1 \vdash Q_2 \mid Q_3 \,] = [\, P_1 \wedge P_3 \,\textbf{\textsf{wp}}\, Q_1 \vdash P_2 \vee P_3 ; Q_2 \mid P_3 ; Q_3 \,]$$

$$\mathit{II}_{\textbf{\textsf{R}}} ; P = P ; \mathit{II}_{\textbf{\textsf{R}}} = P$$

$$[\, P_1 \vdash P_2 \mid \textbf{\textsf{false}} \,] ; Q = [\, P_1 \vdash P_2 \mid \textbf{\textsf{false}} \,]$$

$$\textbf{\textsf{Chaos}} ; P = \textbf{\textsf{Chaos}}$$

$$[\, P \vdash Q \mid R \,]^{n+1} = \left[\bigwedge_{i \leq n} (R^i \,\textbf{\textsf{wp}}\, P) \,\middle|\!- \left(\bigvee_{i \leq n} R^i \right) ; Q \,\middle|\, R^{n+1} \right]$$

Parallel composition

INTO-CPS

- defined wrt. a merge predicate M
- describes how state and traces should be merged
- rather complicated...

$$[\, P_1 \vdash P_2 \mid P_3 \,] \parallel^{R}_{M} [\, Q_1 \vdash Q_2 \mid Q_3 \,] =$$

$$[\, P_2 \; \boldsymbol{wr}_M \; Q_1 \wedge P_3 \; \boldsymbol{wr}_M \; Q_1 \wedge Q_2 \; \boldsymbol{wr}_M \; P_1 \wedge Q_3 \; \boldsymbol{wr}_M \; P_1$$
$$\vdash P_2 \parallel^{E}_{M} Q_2 \;\wedge\; P_3 \parallel^{E}_{M} Q_2 \;\wedge\; P_2 \parallel^{E}_{M} Q_3$$
$$\mid P_3 \parallel_M Q_3 \,]$$

- $P \; \boldsymbol{wr}_M \; Q$ is the weakest assumption under which process Q will not violate condition P
- $\parallel^{E}_{M}$ merges only traces; $\parallel_M$ merges both states and traces
- parallel composition is always monotonic

Verification of reactive designs

INTO-CPS

- involves solving a conjecture: $[\, P_1 \vdash P_2 \mid P_3 \,] \sqsubseteq \mathrm{System}$
- $[\, P_1 \vdash P_2 \mid P_3 \,]$: contract with assumptions and commitments

(1) perform algebraic simplification of model
(2) calculate pre-, peri-, and postconditions of System
(3) apply refinement law to yield three proof obligations
(4) discharge (or refute) POs using Isabelle tactics

- we've implemented tactics to facilitate this
- rdes-calc – calculate a reactive design
- rel-auto – solve relational calculus
- + Isabelle provides access to ATPs with sledgehammer

Outline

Hybrid Computation

INTO-CPS

- how to use reactive contracts to specify hybrid systems?
- need to augment relational calculus with continuous variables
- i.e. piecewise continuous functions $\underline{x} : \mathbb{R}_{\geq 0} \to \mathbb{R}$
- desire to support specification in style of duration calculus
- additional operators for constructing ODEs and DAEs
- we achieve this by generalising the trace model (tr, tr')
- will enable a contractual approach for Simulink
- inspirations:
  - Hybrid CSP (He, Zhan et al.) — DAEs and pre-emption
  - HRML (He) — tri-partite alphabet
  - Duration Calculus (Zhou et al.) — interval operator
  - Timed Reactive Designs (Hayes et al.) — timed trace model

Trace algebra

INTO-CPS

- we first abstractly characterise the trace operators

Definition (Trace algebra)

A trace algebra $(\mathcal{T}, ^\frown, \epsilon)$ is a cancellative monoid satisfying:

$$x ^\frown (y ^\frown z) = (x ^\frown y) ^\frown z \tag{TA1}$$

$$\epsilon ^\frown x = x ^\frown \epsilon = x \tag{TA2}$$

$$x ^\frown y = x ^\frown z \;\Rightarrow\; y = z \tag{TA3}$$

$$x ^\frown z = y ^\frown z \;\Rightarrow\; x = y \tag{TA4}$$

$$x ^\frown y = \epsilon \;\Rightarrow\; x = \epsilon \tag{TA5}$$

- cancellation laws (TA3, TA4) allow us to decompose a trace

Horizon 2020 Programme

Trace operators

INTO-CPS

Definition (Trace prefix and subtraction)

$$x \le y \;\Leftrightarrow\; \exists\, z \bullet y = x \frown z$$

$$y - x \;\widehat{=}\; \begin{cases} \iota z \bullet y = x \frown z & \text{if } x \le y \\ \epsilon & \text{otherwise} \end{cases}$$

- trace prefix ($x \le y$): there is a z that extends x to yield y
- trace minus ($y - x$): if $y \le x$ then obtain the difference

Trace prefix laws

INTO-CPS

Theorem (Trace prefix laws)

$$(\mathcal{T}, \leq) \text{ is a partial order} \tag{TP1}$$

$$\epsilon \leq x \tag{TP2}$$

$$x \leq x \frown y \tag{TP3}$$

$$x \frown y \leq x \frown z \Leftrightarrow y \leq z \tag{TP4}$$

Theorem (Trace subtraction laws)

$$x - \epsilon = x \tag{TS1}$$

$$\epsilon - x = \epsilon \tag{TS2}$$

$$x - x = \epsilon \tag{TS3}$$

$$(x \frown y) - x = y \tag{TS4}$$

$$(x - y) - z = x - (y \frown z) \tag{TS5}$$

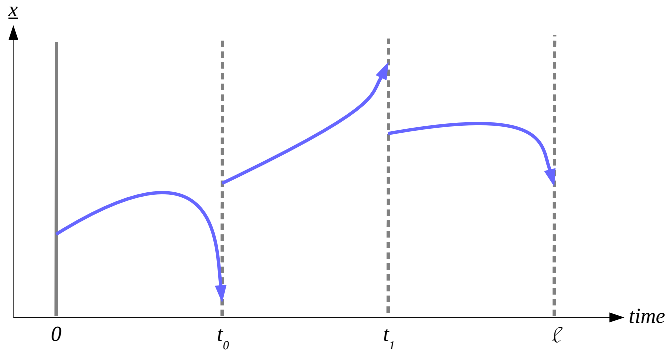
Generalised Reactive Designs

INTO-CPS

- redefine healthiness condition ***R*** using trace algebra
- define operators in terms of abstract trace behaviours
- reprove the laws of reactive designs in general context
- gives rise to generalised reactive design contracts
- different models support different reactive languages
- e.g. sequences form a trace algebra $\Rightarrow$ Circus
- allows import of law library into specialised theory

Continuous Time Traces

- model for hybrid systems is piecewise continuous functions
- finite number of left-closed/right-open continuous segments
- adapted from the work of (Hayes, 2006)



Mathematical Model

INTO-CPS

Definition (Timed traces)

$$\mathbb{TT} \mathrel{\widehat=} \left\{ \begin{array}{l} f : \mathbb{R}_{\geq 0} \nrightarrow \Sigma \\ \mid\ \exists\, t \bullet \mathrm{dom}(f) = [0, t) \\ \qquad \wedge\ t > 0 \Rightarrow \exists\, I : \mathbb{R}_{\mathsf{oseq}} \\ \quad \bullet \left(\begin{array}{l} \mathrm{ran}(I) \subseteq [0, t] \wedge \{0, t\} \subseteq \mathrm{ran}(I) \\ \wedge \left(\ \forall\, n < \#I - 1 \bullet f \ \textsf{cont-on}\, [I_n, I_{n+1})\ \right) \end{array} \right) \end{array} \right\}$$

$$\mathbb{R}_{\mathsf{oseq}} \mathrel{\widehat=} \{x : \textsf{seq}\,\mathbb{R} \mid \forall\, n < \#x - 1 \bullet x_n < x_{n+1}\}$$

$$f\ \textsf{cont-on}\,[m, n) \mathrel{\widehat=} \forall\, t \in [m, n) \bullet \lim_{x \to t} f(x) = f(t)$$

- I: a finite sequence of continuous intervals
- Σ: a topological space denoting the continuous state (e.g. $\mathbb{R}^n$)
- continuous variables are projections (lenses) on Σ

Operators

INTO-CPS

Definition (Timed-trace operators)

$$
\begin{aligned}
f \gg n &\mathrel{\widehat{=}} \lambda\, x \bullet f(x - n) \\
\mathsf{end}(f) &\mathrel{\widehat{=}} \min(\mathbb{R}_{\geq 0} \setminus \mathrm{dom}(f)) \\
\langle\rangle &\mathrel{\widehat{=}} \emptyset \\
f \frown g &\mathrel{\widehat{=}} f \cup (g \gg \mathsf{end}(f))
\end{aligned}
$$

- $f \gg n$ shifts $f : \mathbb{R} \nrightarrow A$ to the right by $n : \mathbb{R}$
- $\mathsf{end}(f)$ obtains the limit point of a trace
- theorem: $(\mathbb{TT}, \frown, \langle\rangle)$ forms a trace algebra
- key lemma: $\mathbb{TT}$ is closed under $\frown$
- we have now obtained a model of hybrid reactive processes

Hybrid relational calculus

- kernel language of relational hybrid programs
- augments relational calculus with continuous variables ($\underline{x}$)
- linked to discrete copies via coupling invariants
- continuous alphabet: $\mathrm{con}\alpha(P)$, discrete alphabet: $\mathrm{dis}\alpha(P)$
- operators denoted in terms of hybrid reactive processes

Hybrid relational calculus

- kernel language of relational hybrid programs
- augments relational calculus with continuous variables ($\underline{x}$)
- linked to discrete copies via coupling invariants
- continuous alphabet: $\mathrm{con}\alpha(P)$, discrete alphabet: $\mathrm{dis}\alpha(P)$
- operators denoted in terms of hybrid reactive processes
- retain standard relational operators
  - composition ($;$), assignment ($x := v$), if-then-else etc.

Hybrid relational calculus

- kernel language of relational hybrid programs
- augments relational calculus with continuous variables ($\underline{x}$)
- linked to discrete copies via coupling invariants
- continuous alphabet: $\mathrm{con}\alpha(P)$, discrete alphabet: $\mathrm{dis}\alpha(P)$
- operators denoted in terms of hybrid reactive processes
- retain standard relational operators
  - composition ($;$), assignment ($x := v$), if-then-else etc.
- add continuous evolution operators
  - differential algebraic equations — $\langle\, \underline{\dot{v}}_1 = f_1;\ \cdots\ ;\ \underline{\dot{v}}_n = f_n \mid B \,\rangle$
  - pre-emption — $P \,[\, B \,]\, Q$
  - interval (continuous specification) — $\lceil P \rceil$

Hybrid relational calculus

INTO-CPS

- kernel language of relational hybrid programs
- augments relational calculus with continuous variables ($\underline{x}$)
- linked to discrete copies via coupling invariants
- continuous alphabet: $\mathrm{con}\alpha(P)$, discrete alphabet: $\mathrm{dis}\alpha(P)$
- operators denoted in terms of hybrid reactive processes
- retain standard relational operators
  - composition ($;$), assignment ($x := v$), if-then-else etc.
- add continuous evolution operators
  - differential algebraic equations — $\langle\, \dot{\underline{v}}_1 = f_1;\ \cdots\ ;\ \dot{\underline{v}}_n = f_n \mid B \,\rangle$
  - pre-emption — $P\,[\,B\,]\,Q$
  - interval (continuous specification) — $\lceil P \rceil$

Example 1: Simple Bouncing Ball

INTO-CPS

Bouncing ball in Modelica

```
model BouncingBall
  Real p(start=2,fixed=true), v(start=0,fixed=true);
equation
  der(v) = -9.81;
  der(p) = v;
  when p <= 0 then
    reinit(v, -0.8*v);
  end when;
end BouncingBall;
```

Bouncing ball in hybrid relational calculus

$$p, v := 2, 0\ ;\ (\langle\, \underline{\dot{p}} = \underline{v};\ \underline{\dot{v}} = -9.81\,\rangle\, [\,\underline{p} \leq 0\,]\, v := -v * .8))^{\omega}$$

Hybrid denotational semantics

$$\begin{aligned}
\textbf{\textit{tt}} &\triangleq tr' - tr \\
\underline{x}(t) &\triangleq \textbf{\textit{tt}}(t).x \\
\ell &\triangleq \mathsf{end}(\textbf{\textit{tt}}) \\
P \mathop{@} \tau &\triangleq \{\underline{x} \mapsto \textbf{\textit{tt}}(\tau).x \mid \underline{x} \in \mathrm{con}\alpha(P) \setminus \{\underline{t}\}\} \dagger P
\end{aligned}$$

- $\sigma \dagger P$ applies substitution function σ to P
- $P \mathop{@} \tau$ lifts continuous variables to instant τ
- e.g. $(\underline{x} > 1 \wedge \underline{y} = \underline{x} \cdot 3) \mathop{@} \tau$ = $(\underline{x}(\tau) > 1 \wedge \underline{y}(\tau) = \underline{x}(\tau) \cdot 3)$
- borrowed from timed refinement calculus

Interval operator

INTO-CPS

$$\lceil P(\tau) \rceil \triangleq tr' \geq tr \wedge (\forall t \in [0, \ell) \bullet P(t) @ t)$$

- cf. Duration Calculus (Zhou, Ravn, and Hanzen, 1993)
- continuous spec: states that P holds on the interval $[0, \ell)$
- e.g. $\lceil 15 < temp \wedge temp \leq 30 \rceil$

Theorem (Interval laws)

$$\lceil P(\tau) \wedge Q(\tau) \rceil = \lceil P(\tau) \rceil \wedge \lceil Q(\tau) \rceil$$
$$\lceil P(\tau) \vee Q(\tau) \rceil \sqsubseteq \lceil P(\tau) \rceil \vee \lceil Q(\tau) \rceil$$
$$\lceil true \rceil = \textbf{\textit{R1}}(\textbf{\textit{true}})$$
$$\lceil false \rceil = (tr' = tr)$$
$$(\forall \tau \bullet P(\tau) \Rightarrow Q(\tau)) \Rightarrow \lceil Q(\tau) \rceil \sqsubseteq \lceil P(\tau) \rceil$$

Hybrid specification

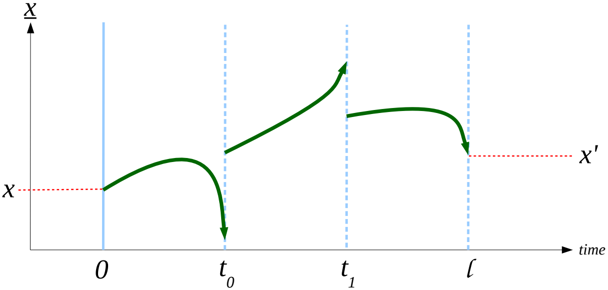
$$[\![P]\!] \triangleq \lceil P \rceil \wedge \ell > 0 \wedge \bigwedge_{\underline{v} \in \mathrm{con}\alpha(P)} \left(v = \underline{v}(0) \wedge v' = \lim_{t \to \ell} (\underline{v}(t)) \right) \wedge \mathit{II}_{\mathrm{dis}\alpha(P)}$$

- continuous state evolves according to P
- must make non-zero progress to avoid certain Zeno effects
- initial condition for v taken from discrete copy
- final condition for v taken from limit construction

Hybrid specification

$$[\![P]\!] \triangleq \lceil P \rceil \wedge \ell > 0 \wedge \bigwedge_{\underline{v} \in \mathrm{con}\alpha(P)} \left(v = \underline{v}(0) \wedge v' = \lim_{t \to \ell} (\underline{v}(t)) \right) \wedge \mathit{\Pi}_{\mathrm{dis}\alpha(P)}$$

- continuous state evolves according to P
- must make non-zero progress to avoid certain Zeno effects
- initial condition for v taken from discrete copy
- final condition for v taken from limit construction



Hybrid Evolution

INTO-CPS

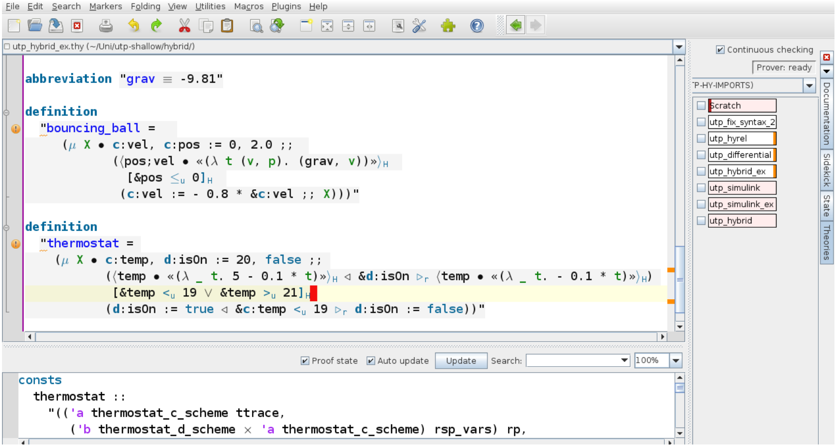
$$\langle\, \dot{\underline{x}} = \mathcal{F}'(\underline{x}, \dot{\underline{x}}) \,|\, B(\underline{x}) \,\rangle \;\triangleq\; \exists \mathcal{F} \bullet \llbracket \mathcal{F}' \text{ has-deriv } \mathcal{F} \text{ at } \tau \wedge \underline{x} = \mathcal{F}(\tau) \wedge B(\underline{x}) \rrbracket$$

$$P \,[\, B \,]\, Q \;\triangleq\; (Q \lhd B \,@\, 0 \rhd (P \wedge \lceil \neg B \rceil)) \vee ((P \wedge \lceil \neg B \rceil \wedge B') \,;\, Q)$$

- differential-algebraic equation: $\langle\, \dot{\underline{x}} = \mathcal{F}'(\underline{x}, \dot{\underline{x}}) \,|\, B(\underline{x}) \,\rangle$
- there exists a solution ($\mathcal{F}$) satisfying the ODE and constraint
- pre-emption: $P \,[\, B \,]\, Q$
- P evolves until B becomes true, then Q is enabled
- remember: all constructs boil down to constraints on the continuous trace

Mechanisation in Isabelle/UTP

- based on Multivariate Analysis and HOL-ODE packages
- support for limits, ODEs, DAEs, and their solutions
- solutions can be verified but not generated



Outline

INTO-CPS

Background

Reactive Systems

Unifying Reactive and Hybrid

Towards Simulink Block Semantics

Conclusion

Horizon 2020 Programme

Recap

INTO-CPS

- we have presented reactive design contracts
- with an intuitive notion of refinement
- we then generalised the underlying trace semantics
- new model based on piecewise-continuous functions
- and constructed a hybrid relational calculus
- we can now combine these to describe Simulink blocks

Overview

- each block is a parametric reactive design
- parameters are constants and shared wires
- wires are modelled as continuous variables
- blocks constrain the behaviours of the wires
- preconditions prevent erroneous behaviour (e.g. div by zero)
- parallel compose instantiated blocks to give overall diagram
- yields a system of differential and algebraic equations

Simulink blocks (tentative!)

INTO-CPS

$$\mathrm{Source}(\underline{x}, v) \mathrel{\widehat=} [\, \textbf{\textit{true}} \vdash \lceil x = v \rceil \mid \textbf{\textit{false}} \,]$$

$$\mathrm{Term}(\underline{x}) \mathrel{\widehat=} [\, \textbf{\textit{true}} \vdash \textbf{\textit{true}} \mid \textbf{\textit{false}} \,]$$

$$\mathrm{Add}(\underline{x}, \underline{y}, \underline{z}) \mathrel{\widehat=} [\, \textbf{\textit{true}} \vdash \lceil z = x + y \rceil \mid \textbf{\textit{false}} \,]$$

$$\mathrm{Gain}(\underline{x}, \underline{y}, v) \mathrel{\widehat=} [\, \textbf{\textit{true}} \vdash \lceil y = v \cdot x \rceil \mid \textbf{\textit{false}} \,]$$

$$\mathrm{Divide}(\underline{x}, \underline{y}, \underline{z}) \mathrel{\widehat=} [\, \lceil y \neq 0 \rceil \vdash \lceil z = x / y \rceil \mid \textbf{\textit{false}} \,]$$

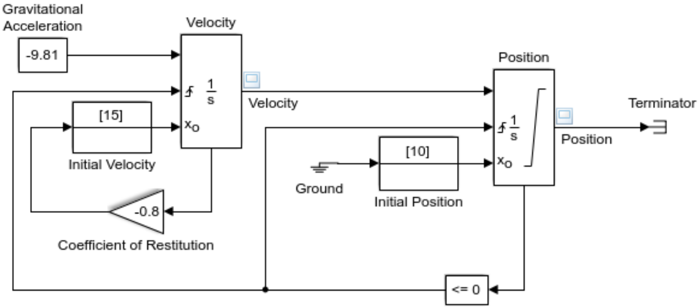
$$\mathrm{Integrate}(\underline{x}, \underline{y}, \underline{y_0}, \underline{r}) \mathrel{\widehat=} x := x_0 \,;\, (\langle\, \dot{y} = x \,\rangle\, [\, r \,]\, x := x_0)^{\omega}$$

$$\mathrm{InitVal}(\underline{x}, \underline{y}, v) \mathrel{\widehat=} [\, \textbf{\textit{true}} \vdash \lceil y = v \lhd \tau = 0 \rhd y = x \rceil \mid \textbf{\textit{false}} \,]$$

$$\mathrm{Cond}(\underline{x}, \underline{y}, f) \mathrel{\widehat=} [\, \textbf{\textit{true}} \vdash \lceil y = f(x) \rceil \mid \textbf{\textit{false}} \,]$$

- all instances of general pattern: Init ; (Cont [Cond] Disc)$^{\omega}$

Bouncing Ball Example



- from `https://uk.mathworks.com/help/simulink/examples/simulation-of-a-bouncing-ball.html`

Translation

INTO-CPS

- we need to give each wire an identifier in the alphabet
- alphabet: $\underline{g}, \underline{v}, \underline{v}_0, \underline{v}_1, \underline{p}, \underline{p}_0, \underline{p}_1 : \mathbb{R}$ and $r : \mathbb{B}$
- diagram described by following parallel composition of blocks:

$$
\begin{aligned}
&\mathsf{Source}(\underline{g}, -9.81) \parallel \mathsf{Integrate}(\underline{g}, \underline{v}_0, \underline{v}, \underline{r}) \parallel \mathsf{Gain}(\underline{v}, \underline{v}_1, -0.8) \\
&\parallel \mathsf{InitVal}(\underline{v}_1, \underline{v}_0, 15) \parallel \mathsf{Integrate}(\underline{v}, \underline{p}, \underline{p}_0, \underline{r}) \parallel \mathsf{Source}(\underline{p}_1, 0) \\
&\parallel \mathsf{InitVal}(\underline{p}_1, \underline{p}_0, 10) \parallel \mathsf{Cond}(\underline{p}, \underline{r}, \lambda\, x \bullet x \leq 0) \parallel \mathsf{Term}(\underline{p})
\end{aligned}
$$

- verification requires calculation of pre- and periconditions
- this involves flattening and solving the system of equations

Outline

Background

Reactive Systems

Unifying Reactive and Hybrid

Towards Simulink Block Semantics

Conclusion

Horizon 2020 Programme

Conclusion

INTO-CPS

- reactive designs facilitate contractual specifications
- Isabelle/UTP provides verification infrastructure
- generalised trace model facilitates hybrid models
- combining these provides denotational foundations for Simulink
- UTP enables combination of models from different languages
  - programming languages (for controller implementation)
  - modelling languages (Modelica, SysML etc.)
- enable a multi-disciplinary approach to system design
- could apply to justify optimisations and transformations

Future work

- investigate correct merge predicate for Simulink
- lots more algebraic laws to prove
- alternative trace models (superdense-time?)
- automated flattening of Simulink diagrams
- reasoning about ODEs in Isabelle/HOL (incl. approximations)
  - cf. work of Fabian Immler and Johannes Hölzl
- integration of CAS with Isabelle
- static analysis of Simulink diagrams
- combination with existing work with Circus
- unifying with our Modelica semantics

References

INTO-CPS

- A. Cavalcanti and J. Woodcock. A Tutorial Introduction to CSP in Unifying Theories of Programming. PSSE 2004. LNCS 3167.
- S. Foster, B. Thiele, A. Cavalcanti, and J. Woodcock. Towards a UTP semantics for Modelica. UTP 2016. LNCS 10134.
- S. Foster, F. Zeyda, and J. Woodcock. Unifying heterogeneous state-spaces with lenses. ICTAC 2016. LNCS 9965.
- Isabelle/UTP. https://github.com/isabelle-utp/utp-main
- F. Immler and J. Hölzl. HOL-ODE. https://www.isa-afp.org/entries/Ordinary_Differential_Equations.shtml
- C. Zhou, A. P. Ravn, M. R. Hansen. An extended Duration Calculus for hybrid real-time systems. Hybrid Systems. LNCS 736. 1993.
- I. J. Hayes, S. E. Dunne, and L. Meinicke. Unifying theories of programming that distinguish nontermination and abort. MPC 2010. LNCS 6120. pp. 178–194.
- C. Zhou, J. Wang, A. P. Ravn. A formal description of hybrid systems. Hybrid Systems III: Verification and Control. LNCS 1066.
- J. He. HRML: a hybrid relational modelling language. QRS 2015.