INTO-CPS

Isabelle/UTP

A Verification Toolbox for Unifying Theories

Simon Foster Jim Woodcock Kangfeng Ye

{firstname.lastname}@york.ac.uk

University of York, UK

Friday 22nd September, 2017

INTO-CPS

into-cps.au.dk

Horizon 2020 Programme

VeTSS Project

INTO-CPS

"Mechanised Assume-Guarantee Reasoning for Control Law Diagrams via Circus"

- AG proof support for discrete time Simulink diagrams
- Circus: stateful reactive language extending CSP
- use of reactive contracts to specify properties
- develop a library of examples and two case studies
- mechanised proof support for Simulink in Isabelle/UTP
- researcher: Dr. Kangfeng Ye (Randall)

Horizon 2020 Programme

Unifying Theories of Programming

- formal semantics framework from Tony Hoare and He Jifeng

Unifying Theories of Programming

- formal semantics framework from Tony Hoare and He Jifeng
- drives to find theories that unify computational paradigms
  - imperative and functional programming
  - sequential and concurrent computation
  - data structures and object orientation
  - real-time and hybrid systems

Unifying Theories of Programming

INTO-CPS

- formal semantics framework from Tony Hoare and He Jifeng
- drives to find theories that unify computational paradigms
  - imperative and functional programming
  - sequential and concurrent computation
  - data structures and object orientation
  - real-time and hybrid systems

> can we find fundamental laws that characterise their commonalities and highlight their differences?

Unifying Theories of Programming

INTO-CPS

- formal semantics framework from Tony Hoare and He Jifeng
- drives to find theories that unify computational paradigms
  - imperative and functional programming
  - sequential and concurrent computation
  - data structures and object orientation
  - real-time and hybrid systems

can we find fundamental laws that characterise their commonalities and highlight their differences?

- use alphabetised relational calculus as a *lingua franca*
- programs-as-predicates: specification + implementation
- link different semantic models (operational, axiomatic etc.)
- build verification tools for various paradigms

Example: Operational Semantics and Hoare Calculus

INTO-CPS

Definition (Transition Relation)

$$(\sigma_1, P_1) \to (\sigma_2, P_2) \;\triangleq\; \langle \sigma_1 \rangle \,;\, P_1 \sqsubseteq \langle \sigma_2 \rangle \,;\, P_2$$

Theorem (Operational Laws)

$$\frac{(\sigma, P) \to (\rho, Q)}{(\sigma, P \,;\, R) \to (\rho, Q \,;\, R)} \; \text{SEQ-STEP}$$

$$\frac{\sigma \models c}{(\sigma, \textbf{\textit{if}}\; c \;\textbf{\textit{then}}\; P \;\textbf{\textit{else}}\; Q) \to (\sigma, P)} \; \text{COND-TRUE}$$

$$\frac{\text{---}}{(\sigma, x := v) \to (\sigma(x := \sigma \dagger v), I\!I)} \; \text{ASSIGN}$$

$$\frac{\sigma \models c}{(\sigma, \textbf{\textit{while}}\; c \;\textbf{\textit{do}}\; P) \to (\sigma, P \,;\, \textbf{\textit{while}}\; c \;\textbf{\textit{do}}\; P)} \; \text{ITER-COPY}$$

Horizon 2020 Programme

Example: Operational Semantics and Hoare Calculus

INTO-CPS

Definition (Transition Relation)

$$(\sigma_1, P_1) \rightarrow (\sigma_2, P_2) \triangleq \langle \sigma_1 \rangle \,;\, P_1 \sqsubseteq \langle \sigma_2 \rangle \,;\, P_2$$

Definition (Hoare Calculus)

$$\{\, p \,\}\, Q \,\{\, r \,\} \triangleq (p \Rightarrow r') \sqsubseteq Q$$

Example: Operational Semantics and Hoare Calculus

INTO-CPS

Definition (Transition Relation)

$$(\sigma_1, P_1) \to (\sigma_2, P_2) \triangleq \langle \sigma_1 \rangle \,;\, P_1 \sqsubseteq \langle \sigma_2 \rangle \,;\, P_2$$

Definition (Hoare Calculus)

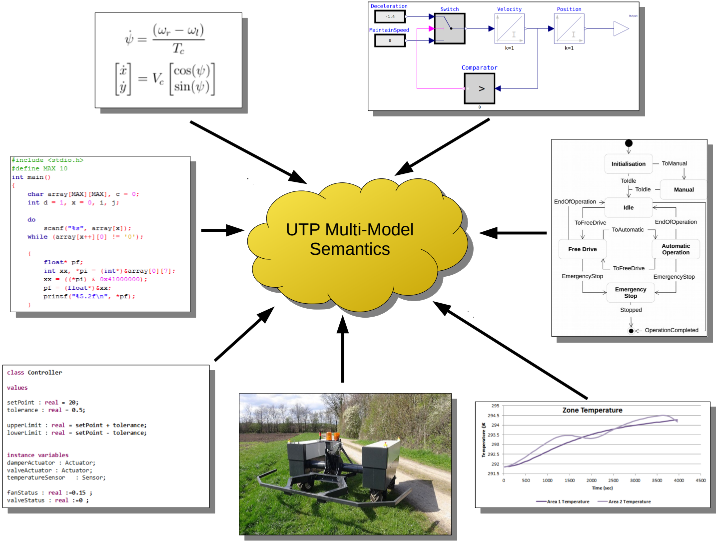
$$\{\, p \,\}\, Q \,\{\, r \,\} \triangleq (p \Rightarrow r') \sqsubseteq Q$$

Theorem (Linking)

$$\{\, p \,\}\, Q \,\{\, r \,\} \Leftrightarrow \frac{\sigma_1 \models p \quad (\sigma_1, Q) \to (\sigma_2, \mathit{II})}{\sigma_2 \models r}$$

- operators are denotations, laws are theorems
- we apply this technique to more complex computational paradigms, such as concurrent and hybrid systems

INTO-CPS Multi-Modelling



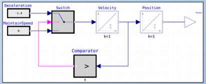
Horizon 2020 Programme

Vision: UTP CPS Verification Foundations

INTO-CPS

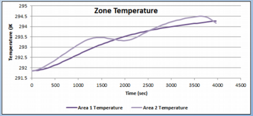
```c
#include <stdio.h>
#define MAX 10
int main()
{
    char array[MAX][MAX], c = 0;
    int d = 1, x = 0, i, j;

    do
        scanf("%s", array[x]);
    while (array[x++][0] != '0');

    {
        float* pf;
        int xx, *pi = (int*)&array[0][7];
        xx = ((*pi) & 0x41000000);
        pf = (float*)&xx;
        printf("%5.2f\n", *pf);
    }
```

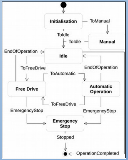
```
class Controller

values

setPoint : real = 20;
tolerance : real = 0.5;

upperLimit : real = setPoint + tolerance;
lowerLimit : real = setPoint - tolerance;

instance variables
damperActuator : Actuator;
valveActuator : Actuator;
temperatureSensor   : Sensor;

fanStatus : real :=0.15 ;
valveStatus : real :=0 ;
```



$$\dot{\psi} = \frac{(\omega_r - \omega_l)}{T_c}$$

$$\begin{bmatrix} \dot{x} \\ \dot{y} \end{bmatrix} = V_c \begin{bmatrix} \cos(\psi) \\ \sin(\psi) \end{bmatrix}$$





Graphical Notations	Object Orientation	Differential Equations	Concurrency	Contracts

Hybrid Systems	State	Real-time	Reactive Systems

Unifying Theories of Programming

Isabelle/HOL

Horizon 2020 Programme

Isabelle/UTP

- a verification toolbox for the UTP based on Isabelle/HOL
- relational calculus, proof tactics, and algebraic laws

Isabelle/UTP

- a verification toolbox for the UTP based on Isabelle/HOL
- relational calculus, proof tactics, and algebraic laws
- define syntax for programs and create verification calculi
- via formalisation of semantic “building blocks” (UTP theories)
- utilise Isabelle’s powerful proof automation for verification

Isabelle/UTP

INTO-CPS

- a verification toolbox for the UTP based on Isabelle/HOL
- relational calculus, proof tactics, and algebraic laws
- define syntax for programs and create verification calculi
- via formalisation of semantic “building blocks” (UTP theories)
- utilise Isabelle’s powerful proof automation for verification
- formalise links between domains using Galois connection
- large library of formalised algebraic laws of programming

Examples

INTO-CPS

```
lemma hoare_ex_1:
  "{|true|}(z := &x) ◁ (&x ≥u &y) ▷r (z := &y){|&z =u maxu(&x, &y)|}u"
  by (hoare_auto)

lemma hoare_ex_2:
  assumes "X > 0" "Y > 0"
  shows
  "{|&x =u «X» ∧ &y =u «Y»|}
    while ¬(&x =u &y)
    invr &x >u 0 ∧ &y >u 0 ∧ (gcdu(&x,&y) =u gcdu(«X»,«Y»))
    do
        (x := (&x - &y)) ◁ (&x >u &y) ▷r (y := (&y - &x))
    od
    {|&x =u gcdu(«X», «Y»)|}u"
  using assms by (hoare_auto, (metis gcd.commute gcd_diff1)+)
```

Examples

INTO-CPS

```
definition Pay :: "index ⇒ index ⇒ money ⇒ action_mdx" where
"Pay i j n =
  pay.((«i»,«j»,«n»)u) →
    ((reject.(«i») → Skip)
      ◁ «i» =u «j» ∨ «i» ∉u domu(&accts) ∨ «n» ≤u 0 ∨ «n» >u &accts(«i»)a ▷R
    ({accts[«i»]} :=C (&accts(«i»)a - «n») ;;
     {accts[«j»]} :=C (&accts(«j»)a + «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)"
```

Examples

INTO-CPS

```
theorem money_constant:
  assumes "finite cards" "i ∈ cards" "j ∈ cards" "i ≠ j"
  shows "[domᵤ(&accts) =ᵤ «cards» ⊢ true | sumᵤ($accts) =ᵤ sumᵤ($accts´)]꜀ ⊑ Pay i j n"
-- {* We first calculate the reactive design contract and apply refinement introduction *}
proof (simp add: assms Pay_contract, rule CRD_refine_rdes)

  -- {* Three proof obligations result for the pre/peri/postconditions. The first requires us to
    show that the contract's precondition is weakened by the implementation precondition.
    It is because the implementation's precondition is under the assumption of receiving an
    input and the money amount constraints. We discharge by first calculating the precondition,
    as done above, and then using the relational calculus tactic. *}

  from assms
  show "`[domᵤ(&accts) =ᵤ «cards»]S< ⇒
          ℐ(true,⟨(pay·(«i», «j», «n»)ᵤ)ᵤ⟩) ⇒ᵣ
          [(«i» ∉ᵤ domᵤ(&accts) ∨ «n» ≤ᵤ 0 ∨ &accts(«i»)ₐ <ᵤ «n») ∨
            «i» ∈ᵤ domᵤ(&accts) ∧ «j» ∈ᵤ domᵤ(&accts)]S<`"
    by (rel_auto)
```

Examples

INTO-CPS

```
theorem extChoice_commute:
  assumes "P is NCSP" "Q is NCSP"
  shows "P □ Q = Q □ P"
  by (rdes_eq cls: assms)

theorem extChoice_assign:
  assumes "P is NCSP" "Q is NCSP"
  shows "x :=c v ;; (P □ Q) = (x :=c v ;; P) □ (x :=c v ;; Q)"
  by (rdes_eq cls: assms)

theorem stop_seq:
  assumes "P is NCSP"
  shows "Stop ;; P = Stop"
  by (rdes_eq cls: assms)
```

Examples

INTO-CPS

```
definition
"BrakingTrain =
   c:accel, c:vel, c:pos := «normal_deceleration», «max_speed», «0» ;;
   ⟨{&accel,&vel,&pos} • «train_ode»⟩ₕ untilₕ ($vel´ ≤ᵤ 0) ;; c:accel := 0"

theorem braking_train_pos_le:
 "($c:accel´ =ᵤ 0 ∧ ⌈$pos´ <ᵤ 44⌉ₕ) ⊑ BrakingTrain" (is "?lhs ⊑ ?rhs")
proof -
  -- {* Solve ODE, replacing it with an explicit solution: @{term train_sol}. *}
  have "?rhs =
    c:accel, c:vel, c:pos := «-1.4», «4.16», «0» ;;
    {&accel,&vel,&pos} ←ₕ «train_sol»(&accel,&vel,&pos)ₐ(«time»)ₐ untilₕ ($vel´ ≤ᵤ 0) ;;
    c:accel := 0"
  by (simp only: BrakingTrain_def train_sol)
  -- {* Set up initial values for the ODE solution using assigned variables. *}
  also have "... =
    {&accel,&vel,&pos} ←ₕ «train_sol(-1.4,4.16,0)(time)» untilₕ ($vel´ ≤ᵤ 0) ;; c:accel := 0"
  by (simp add: assigns_r_comp usubst unrest alpha, literalise, simp)
```

Conclusion

- UTP enables a holistic approach to formal semantics
- Isabelle/UTP: computational theories $\rightarrow$ verification tools
- wide spectrum of paradigms supported

Conclusion

INTO-CPS

- UTP enables a holistic approach to formal semantics
- Isabelle/UTP: computational theories $\rightarrow$ verification tools
- wide spectrum of paradigms supported
- still much more work to be done
- more UTP theories to mechanise (objects, real-time, etc.)
- performance and scalability
- VeTSS: reasoning about discrete-time Simulink diagrams

Conclusion

INTO-CPS

- UTP enables a holistic approach to formal semantics
- Isabelle/UTP: computational theories $\rightarrow$ verification tools
- wide spectrum of paradigms supported
- still much more work to be done
- more UTP theories to mechanise (objects, real-time, etc.)
- performance and scalability
- VeTSS: reasoning about discrete-time Simulink diagrams

- Isabelle/UTP: http://www.cs.york.ac.uk/~simonf/utp-isabelle
- GitHub: https://github.com/isabelle-utp/utp-main
- Email: simon.foster@york.ac.uk

References

INTO-CPS

- C. A. R. Hoare and J. He. Unifying Theories of Programming. Prentice Hall, 1998
- 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.
- A. Cavalcanti, P. Clayton, and C. O'Halloran. From control law diagrams to Ada via Circus. Formal Aspects of Computing, 23(4):465-512, Jul 2011.
- Isabelle/UTP. https://github.com/isabelle-utp/utp-main