Isabelle/UTP
A Verification Toolbox for Unifying Theories

Simon Foster    Jim Woodcock    Kangfeng Ye
{firstname.lastname}@york.ac.uk

University of York, UK

Friday 22\textsuperscript{nd} September, 2017
VeTSS Project

“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)
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

▶ 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

- 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

**Definition (Transition Relation)**

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

**Theorem (Operational Laws)**

\[
\frac{(\sigma, P) \rightarrow (\rho, Q)}{(\sigma, P ; R) \rightarrow (\rho, Q ; R)} \quad \text{SEQ-STEP}
\]

\[
\frac{\sigma \models c}{(\sigma, \text{if } c \text{ then } P \text{ else } Q) \rightarrow (\sigma, P)} \quad \text{COND-TRUE}
\]

\[
\frac{(\sigma, x := v) \rightarrow (\sigma(x := \sigma \uparrow v), \bot)}{\text{ASSIGN}}
\]

\[
\frac{\sigma \models c}{(\sigma, \text{while } c \text{ do } P) \rightarrow (\sigma, P ; \text{while } c \text{ do } P)} \quad \text{ITER-COPY}
\]
Example: Operational Semantics and Hoare Calculus

### 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

**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)**

\[\{ p \} \ Q \{ r \} \triangleq (p \Rightarrow r') \sqsubseteq Q\]

**Theorem (Linking)**

\[\{ p \} \ Q \{ r \} \iff \sigma_1 \models p \quad (\sigma_1, Q) \rightarrow (\sigma_2, \bot) \quad \sigma_2 \models r\]

- operators are *denotations*, laws are *theorems*
- we apply this technique to more complex computational paradigms, such as *concurrent* and *hybrid systems*
$\dot{\psi} = \frac{\omega_r - \omega_i}{T_e}$

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

```
#include <stdio.h>
#define MAX 10

int main()
{
    char array[MAX];
    int d = 1, x = 0, i;
    do
        scanf("%s", array[i]);
        while (array[x] = 0, i++);
    float pf;
    int xx, "px = \{int\}*array[0][?];
    px = (*float)xx;
    printf("%s", "px);
}
```

```cpp
class Controller
{
    public:
        setpoint : real = 20;
        tolerance : real = 0.5;
        upperLimit : real = setPoint + tolerance;
        lowerLimit : real = setPoint - tolerance;

    private:
        instance variables
        dampenActuator : Actuator;
        valveActuator : Actuator;
        temperature : Sensor;
        setstatus : real = 0.11;
        valvestatus : real = 0;
}
```

UTP Multi-Model Semantics

INTO-CPS Multi-Modelling

Horizon 2020 Programme
INtegrated TOolchain for Cyber-Physical Systems

Vision: UTP CPS Verification Foundations

Graphical Notations | Object Orientation | Differential Equations | Concurrency | Contracts
---|---|---|---|---
Hybrid Systems | State | Real-time | Reactive Systems

Unifying Theories of Programming

Isabelle/HOL
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

- 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
lemma hoare_ex_1:
"\{true\}(z := &x) \land (\&x \geq_{u} \&y) \land (z := \&y) \land (\&z =_{u} \max_{u}(\&x, \&y))\}
by (hoare_auto)

lemma hoare_ex_2:
assumes "X > 0" "Y > 0"
shows "\{\&x =_{u} «X» \land \&y =_{u} «Y»\}
while ¬(\&x =_{u} \&y)
invr \&x >_{u} 0 \land \&y >_{u} 0 \land (gcd_{u}(\&x, \&y) =_{u} gcd_{u}(«X», «Y»))
do
(x := (\&x - \&y)) \land (\&x >_{u} \&y) \land (y := (\&y - &x))
od
\{\&x =_{u} gcd_{u}(«X», «Y»)\}
using assm (hoare_auto, (metis gcd.commute gcd_diff1)+)
Examples

**definition** Pay :: "index \(\Rightarrow\) index \(\Rightarrow\) money \(\Rightarrow\) action\_mdx" \(\textit{where}\)

\[
\text{Pay } i \ j \ n = \begin{cases}
pay.((i, j, n)) & \text{if } (i, j, n) \in \text{dom}_u \text{(accts)} \\
\text{reject.}(i) & \text{otherwise}
\end{cases}
\]

\[
\begin{align*}
\text{\{accts[\langle i \rangle]\}} & \leftarrow C (\text{accts}(i) - n) \\
\text{\{accts[\langle j \rangle]\}} & \leftarrow C (\text{accts}(j) + n) \\
\text{accept.}(i) & \rightarrow \text{Skip}
\end{align*}
\]

**definition** PaySet :: "index \(\Rightarrow\) (index \times index \times money) set" \(\textit{where}\)

\[
\text{PaySet cardNum} = \{(i, j, k). i < \text{cardNum} \land j < \text{cardNum} \land i \neq j\}
\]

**definition** AllPay :: "index \(\Rightarrow\) action\_mdx" \(\textit{where}\)

\[
\text{AllPay cardNum} = (\prod (i, j, n) \in \text{PaySet cardNum} \cdot \text{Pay } i \ j \ n)
\]
Examples

\begin{quote}
\texttt{theorem money_constant:}
\begin{quote}
\texttt{\hspace{2em} assumes "finite cards" "i \in cards" "j \in cards" "i \neq j"}
\begin{quote}
\texttt{\hspace{2em} shows "[dom_u(&accts) =_u \langle cards \rangle \vdash true \mid sum_u($accts) =_u sum_u($accts')] \subseteq \text{Pay i j n}"}
\end{quote}
\end{quote}
\begin{quote}
\texttt{\hspace{2em} -- \{\* We first calculate the reactive design contract and apply refinement introduction \*\}}
\end{quote}
\begin{quote}
\texttt{\hspace{2em} proof (simp add: assms Pay_contract, rule CRD_refine_rdes)}
\end{quote}
\begin{quote}
\texttt{\hspace{2em} -- \{\* Three proof obligations result for the pre/per/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. \*\}}
\end{quote}
\end{quote}

\begin{quote}
\texttt{\hspace{2em} from assms}
\begin{quote}
\texttt{\hspace{2em} show "[\langle dom_u(&accts) =_u \langle cards \rangle \rangle_{S_\leq} \Rightarrow}
\begin{quote}
\texttt{\hspace{2em} I(true,{(pay(\langle i \rangle, \langle j \rangle, \langle n \rangle)_{U_\leq})}) \Rightarrow_r}
\begin{quote}
\texttt{\hspace{2em} [\langle i \rangle \notin dom_u(&accts) \lor \langle n \rangle \leq_0 \theta \lor &accts(\langle i \rangle)_{n_\leq \langle n \rangle) \lor}
\begin{quote}
\texttt{\hspace{2em} \langle i \rangle \notin dom_u(&accts) \land \langle j \rangle \in_0 dom_u(&accts)]_{S_\leq}"}
\texttt{by (rel_auto)}}
\end{quote}
\end{quote}
\end{quote}
\end{quote}
\end{quote}
Examples

```latex
\begin{verbatim}
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)
\end{verbatim}
```
Examples

**definition**
```
"BrakingTrain =
  c:accel, c:vel, c:pos := «normal_deceleration», «max_speed», «0» ;;
  {{accel, &vel, &pos} • «train_ode»} until_h ($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 until_h ($vel ≤_u 0) ;; c:accel := 0"
  by (simp only: BrakingTrain_def train_sol)
-- {* Set up initial values for the ODE solution using assigned variables. *}
also have "... =
  {{accel, &vel, &pos} ←_h «train_sol(-1.4,4.16,0)(time)» until_h ($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 → verification tools
- wide spectrum of paradigms supported
Conclusion

- UTP enables a holistic approach to formal semantics
- Isabelle/UTP: computational theories → 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

- UTP enables a holistic approach to formal semantics
- Isabelle/UTP: computational theories → 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

- Isabelle/UTP. https://github.com/isabelle-utp/utp-main