Isabelle/UTP: Mechanised Theory Engineering for Computer Scientists

Simon Foster

University of York

July 31, 2013

Outline

Outline

COMPASS Project

- EU FP7 research project looking at Systems of Systems
- SoS: complex, heterogeneous and distributed systems with diverse stakeholders
- e.g. smart cities, home ecosystems
- Q: how to model and verify such systems?
- A: COMPASS Modelling Language (CML)
- Circus process model, VDM specification language
- denotational/operational semantics based in the Unifying Theories of Programming

CML Bit Register

```
types
  Byte = int
  inv n == (n >= 0) and (n <= 255)

functions
 oflow : (Byte*Byte) -> bool
 oflow(i,j) == i+j > 255

 uflow : (Byte*Byte) -> bool
 uflow(i,j) == i-j < 0

channels
 init, overflow, underflow
 read, load, add, sub : Byte
```

CML Bit Register

```
process RegisterProc =
begin
  state
    reg : int

  operations
   INIT : () ==> ()
   INIT() == reg := 0

   LOAD : int ==> ()
   LOAD(i) == reg := i

   ADD: int ==> ()
   ADD(i) == reg := reg + i

   ...
```

CML Bit Register

```
REG =
 (load?i -> LOAD(i) ; REG)
 []
 (dcl j: int @ j := READ(); read!j -> REG)
 []
 (add?i -> (([oflow(reg,i)] & overflow -> INIT(); REG)
            []
            ([not oflow(reg,i)] & ADD(i); REG)))
 []
 (sub?i -> (([uflow(reg,i)] & underflow -> INIT(); REG)
            []
            ([not uflow(reg,i)] & SUB(i); REG)))
 @
   init -> INIT(); REG
```

Outline

Introduction – UTP

Unifying Theories of Programming (Hoare & He)

- a predicative relation algebra for formalising semantics
- programs as predicates – inputs related to outputs
- unified language for implementation and specification
- basically, everything is a relation
- emphasises denotational semantics: precise operator definition
- unifies many theories from across computer science
- e.g. relation algebra, WP semantics, Hoare logic, CSP, OOP

Introduction – UTP

What's in a UTP theory?

- alphabet – the obervations which can be made
- signature – functions on objects of the theory
- healthiness conditions – idempotent functions under which theory is closed

Example: Reactive Processes (CSP etc.)

- alphabet: variable tr, tr' to represent before/after traces
- signature: operators of CSP ($P \Box Q$, $a?x \rightarrow P$ etc.)
- healthiness conditions: $R1(P) = P \wedge tr \leq tr'$

Introduction

Isabelle/UTP

- an implementation of the UTP in Isabelle/HOL
- a hybrid embedding – techniques from deep / shallow
- formalises variable bindings, predicates, relations and theories
- dual aim:
  - mechanising programming language semantics (+ transcribe pen-and-paper proofs)
  - program verification / refinement
- **fully extensible**: add new operators by definition
- **fully customisable**: user specified values and types
- emphasis on usability – use by non Isabelle experts

Isabelle/HOL

- Isabelle – a generic proof assistant
  - proof checking (decidable)
  - proof automation (undecidable)
- HOL – Higher Order Logic
  - Functional Programming: $f = reverse \cdot map\ g$
  - Logic: $\forall xs.\ map\ f\ (map\ g\ xs) = map\ (f \cdot g)\ xs$
  - similar syntax to ML and Haskell
- LCF-style: proofs correct by construction wrt. a small logical core
- Large library of theories (Sets, Lists, Lattices, Automata etc.)
- Robust technology ($>$ 20 years in the making)

Encoding Overview

Predicates

- encoded as well-typed subsets of $\mathbb{P}$ (variable $\rightarrow$ value)
- set of variable bindings which make the predicate true
- e.g. $[\![x > 5]\!] = \{(x \mapsto 6), (x \mapsto 7), (x \mapsto 8) \cdots\}$
- bindings total – unconstrained variables possess all mappings
- $\emptyset$ represents false, UNIV represents true
- predicate operators generally map to set operators
- except quantifiers: $[\![\exists x.P]\!] = \{b1 \oplus b2 \ on \ \{x\} | b1 \in P\}$

Relations

- predicates which consist of primed and unprimed variables

What's included?

- basic predicate theory
  - logical operators, quantifiers, expressions, alpha renaming, substitution
- binary relations (inc. imperative programming operators)
  - sequential composition $\fatsemi$
  - assignment $x := v$
  - if-then-else $P \lhd c \rhd Q$
- complete lattice theory
  - weakest and strongest fixpoint $\mu X.P$, $\nu X.P$
  - finite iteration (Kleene Star) $P^\star$
- weakest precondition semantics
- theory of Designs (imperative programs with total correctness)

UTP Generic Parser

- Isabelle allows the construction of flexible parsers
- full mixfix syntax
- user specifies priority grammar rules
- fully modular – new syntax can be added
- we have built a basic parser for UTP syntax
- special quotes are used: $`p \Rightarrow q \wedge r`$
- user can add grammar rules for new operators

Value Models

- a way of injecting HOL types into UTP predicates
- needed to define concrete programs
- predicate values are user defined
- user supplies four things:
  1. a value sort type α
  2. a typing sort type τ (must be countable)
  3. a typing relation $\_:\_ :: \alpha \Rightarrow \tau \Rightarrow \text{bool}$
  4. a definedness predicate $\mathcal{D} :: \alpha \Rightarrow \text{bool}$
- main predicate type parametric: α WF-PREDICATE
- basic value model for VDM exists
- will be used as a basis for mechanising CML

VDM Values

```
datatype vbasic
  = PairI vbasic vbasic
  | NatI "nat"
  | IntI "int"
  | RatI "rat"
  | RealI "real"
  | CharI "char"
  | QuoteI "string"
  | TokenI vbasic
  | ListI "vbasic list"
  | FinI "vbasic list"
  | BoolI bool
  | RecI "vbasic list"
  | MapI "(vbasic * vbasic) list"
```

VDM type relation

inductive vbasic-type-rel :: $vbasic \Rightarrow vdmt \Rightarrow bool$ ($infix :_b 50$)

BoolI-type : BoolI $x :_b$ BoolT |
NatI-type : NatI $x :_b$ NatT |
IntI-type : IntI $x :_b$ IntT |
RatI-type : RatI $x :_b$ RatT |
RealI-type : RealI $x :_b$ RealT |
CharI-type : CharI $x :_b$ CharT |
TokenI-type : TokenI $x :_b$ TokenT |
QuoteI-type : QuoteI $x :_b$ QuoteT |
ListI-type : $[\![\forall x \in set\ xs.x :_b a]\!] \Longrightarrow$ ListI $xs :_b$ ListT a

Expressions

- Expressions typed as (binding $\rightarrow$ value)
- shallow expression model – existing Isabelle constructs lifted
- existing proof lemmas can be directly used
- variables given deep encoding
- e.g. numbers, strings, lists, finite sets etc.
- can be undefined – VDM proof obligations entail definedness

Outline

Proof in Isabelle

Isabelle Proof Tools

- powerful set of automated tools
  - simplifier – equational rewriting
  - blast – classical deduction (introduction / elimination)
  - auto – combination of several tools
  - Z3 – satisfiability modulo theorems
  - sledgehammer – call external automated theorem provers
- user-friendly proof scripting language (Isar)

Isar

- a natural proof language for Isabelle
- acts as an alternate syntax for proof scripts

Isar	Isabelle

lemma *my_goal* :
  assumes P
  shows $A = B$
proof $-$
  from *assms* **have** Q
  by *blast*
 	
  thus ?*thesis*
  by *force*
qed
	lemma *my_goal* : $P \Longrightarrow A = B$
	  apply(*subgoal_tac* Q)
	  apply(*force*)
	  apply(*blast*)
	qed

Proof in UTP

Question

- how to make use of Isabelle proof tools in the UTP?
- want to reach the same level of complexity as book proofs
- solution: transfer Isabelle proofs to UTP proofs

Proof Strategy – Tactics

Proof by Interpretation

- identify subtheories within UTP
- use HOL proof procedures to discharge goals
- fundamental idea in Isabelle

- tactics developed so far:
  - utp-pred-tac: UTP predicates as HOL predicates
    - handles std. logic operators $\wedge$, $\vee$, $\Rightarrow$, $\sqsubseteq$, $\neg$, $\exists$, $\forall$
  - utp-rel-tac: UTP predicates as HOL relations
    - classical binary relations: $\mathbb{P}\,(A \times A)$
    - handles most rel operators $\cup$, $\cap$, $;$, II, false, $^\smile$
  - utp-xrel-tac: well-formed UTP relations as HOL relations
    - adds $\neg$ and true: complete relation algebra

What's in a UTP tactic?

1. interpretation function – from the UTP to the target domain
2. transfer rules – transfer results from target into UTP domain
3. congruence rules – map operators of UTP to the target

Predicate Tactic: utp-pred-tac

Interpretation Function

EvalP :: α WF_PREDICATE $\Rightarrow$ α WF_BINDING $\Rightarrow$ bool $([\![\_]\!]\_)$

Transfer Theorem

$$(P = Q) \longleftrightarrow (\forall b.[\![P]\!]b = [\![Q]\!]b)$$

Congruence Rules (Selection)

$$\begin{aligned}
[\![\mathsf{true}]\!]b &= \mathsf{True} \\
[\![\mathsf{false}]\!]b &= \mathsf{False} \\
[\![\neg_{\mathsf{p}}\ P]\!]b &= \neg[\![P]\!]b \\
[\![P \wedge_p Q]\!]b &= [\![P]\!]b \wedge [\![Q]\!]b \\
[\![\exists_{\mathsf{p}}\ vs.\ p]\!]b &= \exists b'.[\![P]\!](b \oplus b' \textbf{ on } vs) \\
[\![\bigsqcup ps]\!]b &= (\forall p \in ps.[\![p]\!]b)
\end{aligned}$$

Relation Tactic: utp-rel-tac

Interpretation Function

EvalR :: α WF_PREDICATE $\Rightarrow$ (α WF_REL_BINDING) rel ($[\![\_]\!]\mathcal{R}$)

Transfer Theorems

$$(P = Q) \longleftrightarrow ([\![P]\!]\mathcal{R} = [\![Q]\!]\mathcal{R})$$
$$(P \sqsubseteq Q) \longleftrightarrow ([\![P]\!]\mathcal{R} \supseteq [\![Q]\!]\mathcal{R})$$

Congruence Rules (Selection)

$$\begin{aligned}
[\![\mathsf{false}]\!]\mathcal{R} &= \emptyset \\
[\![P \wedge_p Q]\!]\mathcal{R} &= [\![P]\!]\mathcal{R} \cap [\![Q]\!]\mathcal{R} \\
[\![P \vee_p Q]\!]\mathcal{R} &= [\![P]\!]\mathcal{R} \cup [\![Q]\!]\mathcal{R} \\
[\![\mathrm{II}]\!]\mathcal{R} &= \mathsf{Id} \\
[\![P \fatsemi Q]\!]\mathcal{R} &= [\![P]\!]\mathcal{R} \circ [\![Q]\!]\mathcal{R}
\end{aligned}$$

Proof Strategy – Algebraic Reasoning

- Isabelle's sledgehammer tool automates algebraic reasoning
- calls several automated theorems provers on problem
- user hits C-c C-a C-s and hopefully a solution comes back
- process:
  1. a relevance filter finds theorems which may be useful
  2. E, SPASS, Vampire, Waldmeister, Z3 called in parallel (some remotely!)
  3. internal ATPs, metis and Z3, used to reconstruct proof
- Significant library of algebraic laws under construction
- Several algebraic rules and theories formalised

Algebraic Laws Selection

SemiR-SkipR-left	$: \mathrm{II} \fatsemi P = P$
SemiR-assoc	$: P \fatsemi (Q \fatsemi R) = (P \fatsemi Q) \fatsemi R$
SemiR-AndP-right-pre	$: P \fatsemi (c \wedge Q) = (P \wedge c') \fatsemi Q$
SemiR-CondR-distr	$: (P \lhd b \rhd Q) \fatsemi R = (P \fatsemi R) \lhd b \rhd (Q \fatsemi R)$
SemiR-extract-var	$: P \fatsemi Q = (\exists x'''.P[x'''/x']; Q[x'''\|x])$
AssignR-SemiR-left	$: (x := e \fatsemi p) = p[e/x]$
AssertR-SemiR	$: b_\perp \fatsemi c_\perp = (b \wedge c)_\perp$
IterP-false	$:$ **while** *false* **do** P **od** $= \mathrm{II}$

Algebraic Theories

Theorem

UTP predicates form a **Boolean Algebra**. *By utp-pred-tac.* □

Theorem

UTP predicates form a **Complete Lattice**. *By utp-pred-tac.* □

Theorem

UTP predicates form a **Kleene Algebra**. *By utp-rel-tac.* □

Theorem

Well-formed UTP relations form a **Relation Algebra**.
By utp-xrel-tac. □

Proofs in Isabelle/UTP

```
theorem RefineP_to_CondR:
  "‘P ⊑ (Q ◁ b ▷ R)‘ = ‘(P ⊑ b ∧ Q) ∧ (P ⊑ ¬ b ∧ R)‘"
proof -
  have "‘P ⊑ (Q ◁ b ▷ R)‘ = ‘[(Q ◁ b ▷ R) ⇒ P]‘"
    by (metis RefP_def)
  also have "... = ‘[(b ∧ Q) ∨ (¬ b ∧ R) ⇒ P]‘"
    by (metis CondR_def)
  also have "... = ‘[b ∧ Q ⇒ P] ∧ [¬ b ∧ R ⇒ P]‘"
    by (utp_pred_auto_tac)
  also have "... = ‘(P ⊑ b ∧ Q) ∧ (P ⊑ ¬ b ∧ R)‘"
    by (metis RefP_def)
  finally show ?thesis .
qed
```

Proof Exercise

See if you can complete this proof:

```
theorem CondR_unreachable_branch:
  "‘(P ◁ b ▷ (Q ◁ b ▷ R))‘ = ‘P ◁ b ▷ R‘" (is "?lhs = ?rhs")
proof -
  have "‘(P ◁ b ▷ (Q ◁ b ▷ R))‘ = ‘((Q ◁ b ▷ R) ◁ ¬b ▷ P)‘"
  also have "... = ‘(Q ◁ b ∧ ¬ b ▷ (R ◁ ¬ b ▷ P))‘"
  also have "... = ‘(Q ◁ false ▷ (R ◁ ¬ b ▷ P))‘"
  also have "... = ‘(R ◁ ¬ b ▷ P)‘"
  also have "... = ?rhs"
end
```

Proof Solution

```
theorem CondR_unreachable_branch:
  "‘(P ◁ b ▷ (Q ◁ b ▷ R))‘ = ‘P ◁ b ▷ R‘" (is "?lhs = ?rhs")
proof -
  have "?lhs      = ‘((Q ◁ b ▷ R) ◁ ¬b ▷ P)‘"
    by (metis CondR_sym)
  also have "... = ‘(Q ◁ b ∧ ¬ b ▷ (R ◁ ¬ b ▷ P))‘"
    by (metis CondR_assoc)
  also have "... = ‘(Q ◁ false ▷ (R ◁ ¬ b ▷ P))‘"
    by (utp_pred_tac)
  also have "... = ‘(R ◁ ¬ b ▷ P)‘"
    by (metis CondR_false)
  also have "... = ?rhs"
    by (metis CondR_sym)
  finally show ?thesis .
qed
```

Outline

Case Study: Theory of Designs

- a subclass of relations
- boolean variables ok and ok' used to observe program starting / terminating
- $P \vdash Q$: a program with precondition P, postcondition Q
- e.g. $x \neq 0 \vdash y' = y/x$
- healthiness conditions: H1 – H4
- most of Designs mechanised in Isabelle/UTP
- have tried to follow book proofs

Designs basic definitions

```
abbreviation "okay ≡ MkPlainP ''okay'' True TYPE(bool)
TYPE('m :: BOOL_SORT)"
abbreviation "OKAY ≡ {okay↓,okay↓´}"

definition DesignD ::
"'VALUE WF_PREDICATE ⇒
 'VALUE WF_PREDICATE ⇒
 'VALUE WF_PREDICATE" (infixr "⊢" 60) where
"p ⊢ q = `ok ∧ p ⇒ ok' ∧ q`"

definition SkipD :: "'VALUE WF_PREDICATE" where
"SkipD = true ⊢ II_REL_VAR - OKAY"

notation SkipD ("II_D")
```

Healthiness Conditions

definition *J_pred :: "'VALUE WF_PREDICATE" ("J")* **where**
"J ≡ (ok ⇒$_p$ ok') ∧$_p$ II$_{REL_VAR - OKAY}$"

abbreviation *ok_true ::*
"'VALUE WF_PREDICATE ⇒ 'VALUE WF_PREDICATE" ("_^{t}"
[150]) **where**
"p^t ≡ ‘p[true/okay´]‘"

definition *H1 :: "'a WF_FUNCTION"* **where** *"H1(P) = ‘ok ⇒ P‘"*
definition *H2 :: "'a WF_FUNCTION"* **where** *"H2(P) = ‘P ; J‘"*
definition *H3 :: "'a WF_FUNCTION"* **where** *"H3(P) = ‘P ; II$_D$‘"*
definition *"isH4(P) ≡ ‘P ; true‘ = ‘true‘"*

H1 has left zero

```
lemma H1_left_zero:
  assumes "P ∈ WF_RELATION" "P is H1"
  shows "true ; P = true"
proof -
  from assms have "`true ; P` = `true ; (ok ⇒ P)`"
    by (simp add:is_healthy_def H1_def)
  also have "... = `true ; (¬ ok ∨ P)`"
    by (simp add:ImpliesP_def)
  also have "... = `(true ; ¬ ok) ∨ (true ; P)`"
    by (simp add:SemiR_OrP_distl)
  also from assms have "... = `true ∨ (true ; P)`"
    by (simp add:SemiR_precond_left_zero closure)
  finally show ?thesis by simp
qed
```

Conclusions

- Isabelle/UTP nearly ready for theory engineers
- 4 theories mechanised: designs, undefinedness, CSP and ACP
- release coming soon
- will provide the basis for the CML theorem prover in the COMPASS project

TODO

- mechanise more theories (reactive process, OhCircus etc.)
- complete VDM/CML value model + unify POs with definedness
- mechanise refinement laws
- operational semantics

Values and Variables

```
class DEFINED =
  fixes Defined   :: "'a ⇒ bool" ("𝒟")

class VALUE = DEFINED +
  fixes   utype_rel :: "'a ⇒ nat ⇒ bool" (infix ":ᵤ" 50)
  assumes utype_nonempty: "∃ t v. v :ᵤ t ∧ 𝒟 v"

typedef 'a UTYPE = "{t. ∃ v::'a. v :ᵤ t ∧ 𝒟 v}"
  by (smt mem_Collect_eq utype_nonempty)

datatype SUBSCRIPT = Sub "nat" | NoSub
datatype NAME = MkName string nat SUBSCRIPT

type_synonym 'VALUE VAR = "NAME × 'VALUE UTYPE × bool"
```

Bindings and Predicates

definition
var_compat :: "'a ⇒ 'a VAR ⇒ bool" (infix "▷" 50) where
"v ▷ x ≡ v : vtype x ∧ (aux x ⟶ 𝒟 v)"

typedef *'a WF_BINDING = "{b . ∀ v::'a VAR. b v ▷ v}"*
— proof: there exists a binding

lift_definition *binding_override_on ::*
"'a WF_BINDING ⇒ 'a WF_BINDING ⇒ 'a VAR set ⇒
'a WF_BINDING" ("_ ⊕$_b$ _ on _" [56, 56, 0] 55) **is**
"override_on" — proof: bindings are closed under override

typedef *'a WF_PREDICATE = "UNIV :: 'a WF_BINDING set set"*
..

Predicate Operators

```
lift_definition TrueP :: "'a WF_PREDICATE"
  is "UNIV :: 'a WF_BINDING set" .

lift_definition FalseP :: "'a WF_PREDICATE"
  is "{} :: 'a WF_BINDING set" .

lift_definition NotP :: "'a WF_PREDICATE ⇒ 'a WF_PREDICATE"
  is "uminus" .

lift_definition AndP ::
  "'a WF_PREDICATE ⇒ 'a WF_PREDICATE ⇒ 'a WF_PREDICATE"
  is "λ x y. x ∩ y :: 'a WF_BINDING set" .

lift_definition ExistsP ::
  "'a VAR set ⇒ 'a WF_PREDICATE ⇒ 'a WF_PREDICATE"
  is "λ vs p. {b1 ⊕b b2 on vs | b1 b2. b1 ∈ p}" .
```

Unrestricted Variables

```
definition UNREST ::
  "'a VAR set ⇒ 'a WF_PREDICATE ⇒ bool" where
"UNREST vs p ⟷
   (∀ b1∈destPRED p . ∀ b2. b1 ⊕b b2 on vs∈destPRED p)"

theorem UNREST_TrueP [unrest]:
"UNREST vs true"
  by (simp add: UNREST_def TrueP_def)

theorem UNREST_AndP [unrest]:
"⟦UNREST vs p1; UNREST vs p2⟧ ⟹
 UNREST vs (p1 ∧p p2)"
  by (simp add: UNREST_def AndP_def)
```

- unrest theory attribute