Isabelle/UTP: Mechanised Theory Engineering for Computer Scientists

Simon Foster (with Frank Zeyda)

University of York

May 7, 2013
1. Overview of Isabelle/UTP
2. Automating Proof
3. Mechanising UTP Theories
Outline

1. Overview of Isabelle/UTP
2. Automating Proof
3. Mechanising UTP Theories
Unifying Theories of Programming

- a predicative relation algebra for formalising semantics
- programs as predicates – unified language for implementation and specification
- emphasises denotational semantics: precise operator definition
- theories are defined by healthiness conditions
- unifies many theories from across computer science
- examples: relation algebra, WP semantics, Hoare logic, CSP, OOP
Isabelle/UTP

- an implementation of the UTP in Isabelle/HOL
- builds on Marcel Oliveira’s original implementation in ProofPowerZ
- formalises variable bindings, predicates, relations and theories
- aim: mechanical program derivation and verification
- **fully extensible**: add new operators by definition
- **fully customisable**: user specified values and types
Isabelle/HOL

- **Isabelle** – a generic proof assistant
  - proof checking (decidable)
  - proof automation (undecidable)
- **HOL** – Higher Order Logic
  - Functional Programming: \( f = \text{reverse} \cdot \text{map} \ g \)
  - Logic: \( \forall \ xs. \ \text{map} \ f (\text{map} \ g \ xs) = \text{map} \ (f \cdot g) \ xs \)
  - 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)
Predicates
- encoded as well-typed subsets of $P$ (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, $\text{UNIV}$ represents true
- predicate operators generally map to set operators
- except quantifiers: $[\exists x. P] = \{b_1 \oplus b_2 \text{ on } \{x\} | b_1 \in P\}$

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 \( ; \)
  - assignment \( x := v \)
  - if-then-else \( P < c > Q \)
- complete lattice theory
  - weakest and strongest fixpoint \( \mu X. P, \nu X. P \)
  - finite iteration (Kleene Star) \( P^* \)
- weakest precondition semantics
- theory of Designs (imperative programs with total correctness)
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: ‘$p \Rightarrow q \land r$’
user can add grammar rules for new operators
Value Models

- needed to define concrete programs
- predicate values are user defined
- user supplies four things:
  1. a value sort type $\alpha$
  2. a typing sort type $\tau$ (must be countable)
  3. a typing relation $\vdash :: \alpha \Rightarrow \tau \Rightarrow \text{bool}$
  4. a definedness predicate $D :: \alpha \Rightarrow \text{bool}$
- main predicate type parametric: $\alpha$ WF-PREDICATE
- basic value model for VDM exists
- will be used as a basis for mechanising CML
**VDM Values**

```plaintext
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"
```

Foster and Zeyda (York, UK)  
Isabelle/UTP  
May 7, 2013
VDM type relation

**inductive** vbasic-type-rel :: vbasic ⇒ vdm ⇒ bool \((\text{infix} : b\ 50)\)

**Booll-type** : Booll \(x : b\) BoolT |

**Natl-type** : Natl \(x : b\) NatT |

**Intl-type** : Intl \(x : b\) IntT |

**Ratl-type** : Ratl \(x : b\) RatT |

**Reall-type** : Reall \(x : b\) RealT |

**Charl-type** : Charl \(x : b\) CharT |

**Tokenl-type** : Tokenl \(x : b\) TokenT |

**Quotel-type** : Quotel \(x : b\) QuoteT |

**Listl-type** : \(\forall x \in \text{set}\ \text{xs}.\ x : b\ a\) \(\rightarrow\) Listl xs : b ListT a
Expressions typed as \((\text{binding} \rightarrow \text{value})\)

- VDM/CML expressions currently shallowly embedded
- HOL values with a corresponding VDM type can be mapped
- e.g. numbers, strings, lists, finite sets etc.
- can be undefined – VDM proof obligations entail definedness
Outline

1. Overview of Isabelle/UTP

2. Automating Proof

3. Mechanising UTP Theories
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**)
- a natural proof language for Isabelle
- acts as an alternate syntax for proof scripts

<table>
<thead>
<tr>
<th>Isar</th>
<th>Isabelle</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>lemma</strong> <em>my_goal</em>: <strong>assumes</strong> <em>P</em></td>
<td><strong>lemma</strong> <em>my_goal</em>: <em>P</em> $\Rightarrow$ <em>A</em> = <em>B</em></td>
</tr>
<tr>
<td><strong>shows</strong> <em>A</em> = <em>B</em></td>
<td><strong>apply</strong> (<em>subgoal_tac</em> <em>Q</em>)</td>
</tr>
<tr>
<td>proof –</td>
<td><strong>apply</strong> (<em>force</em>)</td>
</tr>
<tr>
<td>from <em>assms</em> have <em>Q</em></td>
<td><strong>apply</strong> (<em>blast</em>)</td>
</tr>
<tr>
<td>by <em>blast</em></td>
<td><strong>qed</strong></td>
</tr>
<tr>
<td><strong>thus</strong> ?<em>thesis</em></td>
<td></td>
</tr>
<tr>
<td>by <em>force</em></td>
<td></td>
</tr>
<tr>
<td><strong>qed</strong></td>
<td></td>
</tr>
</tbody>
</table>
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 \( \land, \lor, \Rightarrow, \subseteq, \neg, \exists, \forall \)
  - **utp-rel-tac**: UTP predicates as HOL relations
    - classical binary relations: \( \mathbb{P}(A \times A) \)
    - handles most rel operators \( \cup, \cap, \emptyset, \mathbb{II}, \text{false}, \sim \)
  - **utp-rel-deep-tac**: well-formed UTP relations as HOL relations
    - adds \( \sim \) 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:** $\alpha$ WF_PREDICATE $\Rightarrow$ $\alpha$ WF_BINDING $\Rightarrow$ bool (\(\llbracket - \rrbracket\))

### Transfer Theorem

\[(P = Q) \iff (\forall b. \llbracket P \rrbracket b = \llbracket Q \rrbracket b)\]

### Congruence Rules (Selection)

- $\llbracket \text{true} \rrbracket b = \text{True}$
- $\llbracket \text{false} \rrbracket b = \text{False}$
- $\llbracket \neg \ p P \rrbracket b = \neg \llbracket P \rrbracket b$
- $\llbracket P \land \ p Q \rrbracket b = \llbracket P \rrbracket b \land \llbracket Q \rrbracket b$
- $\llbracket \exists p \ vs. \ p \rrbracket b = \exists b'. \llbracket P \rrbracket (b \oplus b' \ on \ vs)$
- $\llbracket \Box \ ps \rrbracket b = (\forall p \in ps. \llbracket p \rrbracket b)$
**Relation Tactic: upp-rel-tac**

### Interpretation Function

\[
\text{EvalR} :: \alpha \text{ WF}_\text{PREDICATE} \Rightarrow (\alpha \text{ WF}_\text{REL}_\text{BINDING}) \text{ rel } ([\_]\mathcal{R})
\]

### Transfer Theorems

\[
\begin{align*}
(P = Q) & \iff ([P]\mathcal{R} = [Q]\mathcal{R}) \\
(P \sqsubseteq Q) & \iff ([P]\mathcal{R} \supseteq [Q]\mathcal{R})
\end{align*}
\]

### Congruence Rules (Selection)

\[
\begin{align*}
[fake] \mathcal{R} & = \emptyset \\
[P \land_p Q] \mathcal{R} & = [P]\mathcal{R} \cap [Q]\mathcal{R} \\
[P \lor_p Q] \mathcal{R} & = [P]\mathcal{R} \cup [Q]\mathcal{R} \\
[\text{II}] \mathcal{R} & = \text{Id} \\
[P ; Q] \mathcal{R} & = [P]\mathcal{R} \circ [Q]\mathcal{R}
\end{align*}
\]
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
<table>
<thead>
<tr>
<th>Law</th>
<th>Equation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>SemiR-SkipR-left</strong></td>
<td>$\Pi \triangleright P = P$</td>
</tr>
<tr>
<td><strong>SemiR-assoc</strong></td>
<td>$P \triangleright (Q \triangleright R) = (P \triangleright Q) \triangleright R$</td>
</tr>
<tr>
<td><strong>SemiR-AndP-right-pre</strong></td>
<td>$P \triangleright (c \land Q) = (P \land c') \triangleright Q$</td>
</tr>
<tr>
<td><strong>SemiR-CondR-distr</strong></td>
<td>$(P \triangleright b \triangleright Q) \triangleright R = (P \triangleright R) \triangleright b \triangleright (Q \triangleright R)$</td>
</tr>
<tr>
<td><strong>SemiR-extract-var</strong></td>
<td>$P \triangleright Q = (\exists x''.P[x''/x]; Q[x''</td>
</tr>
<tr>
<td><strong>AssignR-SemiR-left</strong></td>
<td>$(x := e \triangleright p) = p[e/x]$</td>
</tr>
<tr>
<td><strong>AssertR-SemiR</strong></td>
<td>$b \perp \triangleright c \perp = (b \land c) \perp$</td>
</tr>
<tr>
<td><strong>IterP-false</strong></td>
<td>$\textbf{while } false \textbf{ do } P \textbf{ od } = \Pi$</td>
</tr>
</tbody>
</table>
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-rel-deep-tac`.
1. Overview of Isabelle/UTP

2. Automating Proof

3. Mechanising UTP Theories
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 \( Q \)
- healthiness conditions: H1 – H4
- most of Designs mechanised in Isabelle/UTP
- have tried to follow book proofs
- DEMO in Isabelle
Conclusions

- Isabelle/UTP nearly ready for theory engineers
- release coming soon
- will provide the basis for the CML theorem prover in the COMPASS project
- a little work needed for alphabets

TODO

- mechanise more theories (reactive process, OhCircus etc.)
- complete VDM/CML value model + unify POs with definedness
- mechanise refinement laws
- operational semantics
- precisely compare with related tech, e.g. IsabelleCircus