Isabelle/UTP Syntax Reference

Simon Foster Frank Zeyda

June 21, 2017

1 Syntax Overview

This document describes the syntax used for writing expressions, predicates, and relations in Isabelle/UTP. On the whole the principle in creating the syntax is to be as faithful as possible to the UTP, whilst remaining conservative to Isabelle/HOL's existing syntax. Thus, where possible we reuse HOL syntax in UTP expressions and predicates, primarily using overloading, but sometimes this cannot be done without compromising HOL. Thus sometimes we need to add subscripts to our operators. Conservation is essential so that we can use HOL as a meta-language to manipulate UTP predicates.

With respect to expressions, on the whole we adopt the syntax for operators described in the Z reference manual[1], whilst making necessary adaptations to suit the type system of HOL. For example, a Z relation can become a function, relation, or finite map when mapped to HOL, and thus we present various overloaded versions of the Z relational operators, such as $\mathrm{dom}(f)$ and $A \rhd f$.

The Isabelle jEdit interface support a form of autocompletion. When typing mathematical syntax it will often present a list of suggested symbols. When this appears and the right symbol is in scope press the TAB key to insert it. This process will be necessary during most editing in Isabelle. On the whole, Isabelle provides LaTeX-like symbols for the majority of its operators.

[1] `http://spivey.oriel.ox.ac.uk/mike/zrm/zrm.pdf`

2 Expression Operators

2.1 Variables and Alphabets

Variables in Isabelle/UTP are modelled using lenses which are parametrised by the variable type τ and state-space σ. We use the Isabelle type-system to distinguish variables of a predicate, whose state-space is α, from variable of a relation, whose state-space is a product $\alpha \times \alpha$. This is why there is two ways of writing x. We can also, to some extent, collection variables in a set, as illustrated.

Isabelle/UTP	Math	Description	Isabelle code(s)
$\&x$	x	predicate variable	`&x`
$\$x$	x	relational before variable	`$x`
$\$x\acute$	x'	relational after variable	`$x\acute`
$x:y$	–	name qualification (lens composition)	`x:y`
$\{\$x, \$y, \$z\acute\}$	$\{x, y, z'\}$	variable set (lens summation)	`{$x,$y,$z\acute}`
Σ	–	all variables ("one" lens)	`\Sigma`
$\emptyset$	$\emptyset$	no variables ("zero" lens)	`\emptyset`
inα	inα	relational input variables	`in\alpha`
outα	outα	relational output variables	`out\alpha`

2.2 Arithmetic Operators

The arithmetic operators employ the Isabelle/HOL type classe hierarchy for groups, rings, fields, and orders. Consequently, UTP enjoys direct syntax for many of these operators. Sometimes the arithmetic operators can also be used for non-arithmetic types; for example 0 can be used to represent an empty sequence.

Math Operator	Description	Isabelle/UTP	Isabelle code(s)
$0, 1, 17, 3.147$	numerals	$0, 1, 17, 3.147$	`0, 1, 17, 3.147`
$x + y$	addition	$x + y$	`x + y`
$x - y$	subtraction	$x - y$	`x - y`
$x \cdot y$	multiplication	$x * y$	`x * y`
x/y	division	x/y	`x / y`
$x \,\mathbf{div}\, y$	integer division	$x \,\mathbf{div}\, y$	`x div y`
$x \,\mathbf{mod}\, y$	integer modulo	$x \,\mathbf{mod}\, y$	`x mod y`
$\lceil x \rceil$	numeric ceiling	$\lceil x \rceil_u$	`\lceil x \rceil \sub u`
$\lfloor x \rfloor$	numeric floor	$\lfloor x \rfloor_u$	`\lfloor x \rfloor \sub u`
$x \leq y$	less-than-or-equal	$x \leq_u y$	`x \le \sub u y`
$x < y$	less-than	$x <_u y$	`x < \sub u y`
$\min(x, y)$	minimum value	$\min_u(x, y)$	`min \sub u (x, y)`
$\max(x, y)$	maximum value	$\max_u(x, y)$	`max \sub u (x, y)`

2.3 Polymorphic Operators

The following operators are overloaded to various different expression types. Notably the functional operators, such as application, update, and domain, can be applied to a variety of HOL types including, functions, finite maps, and relations.

Math Operator	Description	Isabelle/UTP	Isabelle code(s)
$P = Q$	equals	$P =_u Q$	`P =\sub u Q`
$P \neq Q$	not equals	$P \neq_u Q$	`P \noteq\sub u Q`
$\lambda\, x \bullet P(x)$	λ-abstraction	$\lambda\, x \bullet P(x)$	`\lambda x \bullet P(x)`
$(x, y, \cdots, z)$	tuple	$(x, y, \cdots, z)_u$	`(x, y, ..., z)\sub u`
$\pi_1(x)$	tuple project first	$\pi_1(x)$	`\pi \sub 1 (x)`
$\pi_2(x)$	tuple project second	$\pi_2(x)$	`\pi \sub 2 (x)`
$f(x)$	functional application	$f(x)_a$	`f(x) \sub a`
$f \oplus \{k_1 \mapsto v_1, \cdots\}$	functional update	$f(k_1 \mapsto v_1, \cdots)_u$	`f(k1 \mapsto v1, ...)\sub u`
$\{k_1 \mapsto v_1, \cdots\}$	enumerated map	$[k_1 \mapsto v_1, \cdots]_u$	`[k1 \mapsto v1, ...]\sub u`
$\emptyset$	empty collection	$[]_u$	`[] \sub u`
$\#x$	size of collection	$\#_u(x)$	`# \sub u (x)`
$\mathrm{dom}(x)$	domain	$\mathrm{dom}_u(x)$	`dom \sub u (x)`
$\mathrm{ran}(x)$	range	$\mathrm{dom}_u(x)$	`ran \sub u (x)`
$f \lhd A$	domain restriction	$f \lhd_u A$	`f \lhd \sub u A`
$A \rhd f$	range restriction	$A \rhd_u f$	`A \rhd \sub u f`

2.4 Sequence Operators

Math Operator	Description	Isabelle/UTP	Isabelle code(s)
$\langle\rangle$	empty sequence	$\langle\rangle$	`\langle\rangle`
$\langle x, y, z\rangle$	enumerated sequence	$\langle x, y, z\rangle$	`\langle x, y, z \rangle`
$\langle m..n\rangle$	sequence interval $[m, n]$	$\langle m..n\rangle$	`\langle m .. n \rangle`
$xs \frown ys$	concatenation	$xs \frown_u ys$	`xs ^ \sub u y`
$\mathrm{head}(xs)$	head of sequence	$\mathrm{head}_u(xs)$	`head \sub u (xs)`
$\mathrm{tail}(xs)$	tail of sequence	$\mathrm{tail}_u(xs)$	`tail \sub u (xs)`
$\mathrm{last}(xs)$	last element	$\mathrm{last}_u(xs)$	`last \sub u (xs)`
$\mathrm{front}(xs)$	all but last element	$\mathrm{front}_u(xs)$	`front \sub u (xs)`
$\mathrm{ran}(xs)$	elements of a sequence	$\mathrm{elems}_u(xs)$	`elems \sub u (xs)`
$\mathrm{map}\, f\, xs$	map function over sequence	$\mathrm{map}_u\, f\, xs$	`map \sub u f xs`

2.5 Set Operators

Math Operator	Description	Isabelle/UTP	Isabelle code(s)
$\emptyset$	empty set	$\{\}_u$	`{} \sub u`
$\{x, y, z\}$	enumerated set	$\{x, y, z\}_u$	`{x, y, z} \sub u`
$[m, n]$	closed set interval	$\{m \mathinner{..} n\}_u$	`{ m .. n } \sub u`
$[m, n)$	open set interval	$\{m \mathinner{..<} n\}_u$	`{ m ..< n } \sub u`
$x \in A$	set membership	$x \in_u A$	`x \in \sub u A`
$x \notin A$	set non-membership	$x \notin_u A$	`x \notin \sub u A`
$A \cup B$	set union	$A \cup_u B$	`A \cup \sub u B`
$A \cap B$	set intersection	$A \cap_u B$	`A \cap \sub u B`
$A \setminus B$	set difference	$A - B$	`A - B`
$A \subseteq B$	subset	$A \subseteq_u B$	`A \subseteq \sub u B`
$A \subset B$	proper subset	$A \subset_u B$	`A \subset \sub u B`

3 Meta-logic Operators

The meta-logic of Isabelle/UTP is simply HOL. Thus we can use HOL to query and manipulate UTP predicates as objects. In particular we can express that an expression does not depend on a particular variable (unrestriction), and apply variable substitutions. We also support various alphabet manipulations which effectively allow the addition and deletion of variables for which we again employ the HOL type system.

Isabelle/UTP	**Description**	**Isabelle code(s)**
$x \sharp P$	P does not depend on variable x	`x \sharp P`
$s \dagger P$	apply substitution s to P	`s \dagger P`
$[x_1 \mapsto v_1, \cdots]_s$	construct substitution with variable x_i being mapped to expresssion v_i	`[x1 \mapsto v1,...] \sub s`
$P[\![v/x]\!]$	apply singleton substitution	`P[|v/x|]`
$P[\![v_1, \cdots / x_1, \cdots]\!]$	apply multiple substitutions	`P[|v1,.../x1,...|]`
$P \oplus_p a$	alphabet extrusion / extension (by lens a)	`P \oplus \sub p a`
$P \restriction_p a$	alphabet restriction (by lens a)	`P \restrict \sub p a`

4 Predicate Operators

Math Operator	Description	Isabelle code(s)
true	logical true / universal relation	`true`
false	logical false / empty relation	`false`
$\neg P$	negation / complement	`~` or `\not`
$P \wedge Q$	conjunction	`/\` or `\and`
$P \vee Q$	disjunction	`\/` or `\or`
$P \Rightarrow Q$	implication	`=>` or `\Rightarrow`
$P \lhd b \rhd Q$	infix if-then-else conditional	`P \triangleleft b \triangleright Q`
$P \sqsubseteq Q$	refinement	`[=` or `\sqsubseteq`
x	predicate variable	`&x`
x	relational before variable	`$x`
x'	relational after variable	`$x\acute`
$\ll v \gg$	HOL term / variable quotation	`<<x>>`
$\forall x \bullet P$	universal quantifier (UTP variable)	`! x \bullet P`
$\exists x \bullet P$	existential quantifier (UTP variable)	`? x \bullet P`
$\forall x \bullet P$	universal quantifier (HOL variable)	`\bold ! x \bullet P`
$\exists x \bullet P$	existential quantifier (HOL variable)	`\bold ? x \bullet P`
$P \sqcap Q$	binary infimum / internal choice	`P \sqcap Q`
$P \sqcup Q$	binary supremum	`P \sqcup Q`
$\bigsqcap i \in I \bullet P(i)$	indexed infimum / internal choice	`\Sqcap i \in I \bullet P(i)`
$\bigsqcup i \in I \bullet P(i)$	indexed supremum	`\Sqcup i \in I \bullet P(i)`
$\mu X \bullet P(X)$	weakest fixed-point	`\mu X \bullet P(X)`
$\nu X \bullet P(X)$	strongest fixed-point	`\nu X \bullet P(X)`
P	UTP predicate tautology	`‘P‘`

5 Relational Operators

Math	Isabelle/UTP	Description	Isabelle code(s)
$P\ ;\ Q$	$P\ ;;\ Q$	sequential composition	`P ;; Q`
$P\ ;\ Q$	$P\ ;;_h\ Q$	homogeneous sequential composition	`P ;; \sub s Q`
II	II	relational identity / skip	`II`
true	$\mathbf{true}_h$	homogeneous universal relation	`true \sub h`
false	$\mathbf{false}_h$	homogeneous empty relation	`false \sub h`
P^{-1}	P^-	relational converse / inverse	`P \sup -`
$P \lhd b \rhd Q$	$P \lhd b \rhd Q$	infix if-then-else	`P \triangleleft b \triangleright Q`
$P \lhd b \rhd Q$	$P \lhd b \rhd_r Q$	if-then-else where b is a condition: a predicate with no after variables	`P \triangleleft b \triangleright \sub r Q`
–	$\langle s \rangle_a$	assignment of substitution s	`\langle s \rangle \sub a`
$x := v$	$x := v$	singleton assignment; v is an expression with no after variables	`x := v`
$b^\top$	$b^\top$	relational assumption	`b \sup \top`
$b_\bot$	$b_\bot$	relational assertion	`b \sub \bottom`
$b^* P$	**while** b **do** P **od**	while loop (strongest fixed-point)	`while b do P od`