Isabelle/UTP Syntax Reference

Simon Foster Frank Zeyda

June 23, 2017

Contents

1 Syntax Overview

This document describes the syntax used for expressions, predicates, and relations in Isabelle/UTP [4]. On the whole the principle in creating the syntax is to be as faithful as possible to mathematical UTP [5, 1], and where necessary Z [7], whilst remaining conservative to Isabelle/HOL's existing syntax. This, of course, means we often need to compromise on both fronts. Where possible we reuse HOL syntax in UTP expressions and predicates, primarily using overloading, but sometimes this cannot be done without undermining fundamental operator of HOL. Thus sometimes we need to add subscripts to our UTP operators. Conservation is essential so that we can use HOL as a meta-language to manipulate UTP predicates.

On the whole, we present tables that show the mathematical UTP (or Z) syntax, the Isabelle/UTP version, and the codes that must be typed into Isabelle. These tables are neither functional nor injective, because one mathematical operator can become several different operators in the mechanisation. This is partly because UTP often has implicit handling of alphabets which is made explicit in Isabelle/UTP. Moreover, the UTP often overloads operator definitions and this sometimes must also be made explicit.

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. For more information, please also see the Isabelle/HOL documentation[2].

[1]`http://spivey.oriel.ox.ac.uk/mike/zrm/zrm.pdf`
[2]`http://isabelle.in.tum.de/documentation.html`

2 UTP Types

Type	Description
$'a \Longrightarrow '\alpha$	UTP variable / lens of type $'a$ and alphabet type $'\alpha$
$('a, '\alpha)\ uexpr$	expression with return type $'a$ and alphabet type $'\alpha$
$'\alpha\ usubst$	substitution on alphabet $'\alpha$
$'\alpha\ upred$	alphabetised predicate with alphabet type $'\alpha$
$('\alpha, '\beta)\ rel$	alphabetised relation with input alphabet $'\alpha$ and output alphabet $'\beta$
$'\alpha\ hrel$	homogeneous alphabetised relation over $'\alpha$
$('s, 'e)\ action$	*Circus* action with state-space type $'s$ and event alphabet type $'e$
$('d, 'c)\ hyrel$	hybrid relation with discrete state-space $'d$ and continuous state-space $'c$

3 Expression Operators

3.1 Variables and Alphabets

Variables in Isabelle/UTP are modelled using lenses [4, 2, 3] 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, depending on the context. Moreover, this particularly surfaces when dealing with programs expressed as relations or in *Circus* [6] as operators frequently use the type system to employ well-formedness of expressions. For instance, in an assignment $x := v$, v has state-space α not $\alpha \times \alpha$ and thus the predicate variable syntax should be used.

We can also use lenses to express alphabets [2] and, to some extent, collect variables in a set using lens operators [4], 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`
$x \bowtie y$	$x \neq y$	variables different (lens independence)	`x \bowtie y`
$x \approx_L y$	$x = y$	variables equal (lens equivalence)	`x \approx \sub L y`
$x \subseteq_L a$	$x \in a$	alphabet membership (sublens)	`x \subseteq \sub L a`

3.2 Arithmetic Operators

The arithmetic operators employ the Isabelle/HOL type class 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	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)`

3.3 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	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`

3.4 Sequence Operators

Sequences are modelled as HOL lists, and we lift several operators.

Math	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`

3.5 Set Operators

Sets are modelled as HOL sets, and we lift several operators.

Math	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	$\{m\,..\,n\}_u$	`{ m .. n } \sub u`
$[m, n)$	open set interval	$\{m\,..{<}\,n\}_u$	`{ 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`

4 Meta-logical Operators

The meta-logic of Isabelle/UTP is simply HOL. Thus, we can use HOL to query and manipulate UTP predicates as objects through application of predicate transformation. 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 [2]. Our meta-logical operators are entirely idiosyncratic to Isabelle/UTP and the only mathematical analogues are statements of the form "x is not mentioned in P", for example.

Isabelle/UTP	**Description**	**Isabelle code(s)**
$x \sharp P$	P does not depend on variable x	`x \sharp P`
$x \bowtie y$	variables different (lens independence)	`x \bowtie y`
$[x_1 \mapsto v_1, \cdots]_s$	construct substitution function with variable x_i being mapped to expression v_i	`[x1 \mapsto v1,...] \sub s`
$s -_s x$	remove variable x from substitution s	`s - \sub s x`
$s \dagger P$	apply substitution function s to P	`s \dagger P`
$\langle s \rangle_s x$	lookup expression associated with x in s	`\langle s \rangle \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`
$\lceil P \rceil_<$	lift predicate P to a precondition relation	`\lceil P \rceil \sub <`
$\lfloor P \rfloor_<$	drop precondition relation P to predicate	`\lfloor P \rfloor \sub <`
$\lceil P \rceil_>$	lift predicate P to a postcondition relation	`\lceil P \rceil \sub >`
$\lfloor P \rfloor_>$	drop postcondition relation P to predicate	`\lfloor P \rfloor \sub >`

5 Predicate Operators

The Isabelle/UTP predicate operators closely mimick the mathematical UTP syntax, and so we do not include a separate math column for space reasons. Operators on variables have two variants, one for HOL variables and one for UTP variables. Therefore, a translation from mathematical UTP to Isabelle/UTP must make a decision about the best option.

Isabelle/UTP	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`
$\ll v \gg$	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`
$\boldsymbol{\forall} x \bullet P$	universal quantifier (HOL variable)	`\bold ! x \bullet P`
$\boldsymbol{\exists} x \bullet P$	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‘

6 Relational Operators

Math	Isabelle/UTP	Description	Isabelle code(s)
x	$\$x$	relational before variable	`$x`
x'	$\$x\acute{}$	relational after variable	`$x\acute`
$P \mathbin{;} Q$	$P\ ;;\ Q$	sequential composition	`P ;; Q`
$P \mathbin{;} Q$	$P\ ;;_h\ Q$	homogeneous sequential composition	`P ;; \sub s Q`
P^n	$P\ \hat{}\ n$	repeated sequential composition	`P \bold ^ n`
$;\ i : xs \bullet P(i)$	$;;\ i : xs \bullet P(i)$	replicated sequential composition	`;; i : xs \bullet P(i)`
P^*	P^*	Kleene star	`P \sup \star n`
II	II	relational identity / skip	`II`
$\mathbf{true}$	$\mathbf{true}_h$	homogeneous universal relation	`true \sub h`
$\mathbf{false}$	$\mathbf{false}_h$	homogeneous empty relation	`false \sub h`
P^{-1}	P^-	relational converse / inverse	`P \sup -`
$P \lhd b \rhd Q$	$P \triangleleft b \triangleright Q$	infix if-then-else	`P \triangleleft b \triangleright Q`
$P \lhd b \rhd Q$	$P \triangleleft b \triangleright_r Q$	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`
$x_1, x_2 := e_1, e_2$	$(x_1, x_2) := (e_1, e_2)$	multiple assignment	`(x1,x2) := (e1,e2)`
$b^\top$	$b^\top$	relational assumption	`b \sup \top`
$b_\bot$	$b_\bot$	relational assertion	`b \sub \bottom`
$b^* P$	$\mathbf{while}\ b\ \mathbf{do}\ P\ \mathbf{od}$	while loop (strongest fixed-point)	`while b do P od`

7 CSP and ***Circus*** Operators

Math	Isabelle/UTP	Description	Isabelle code(s)
Chaos	**Chaos**	chaotic / aborting process	`Chaos`
Miracle	**Miracle**	most deterministic process	`Miracle`
Skip	**Skip**	do nothing and terminate	`Skip`
Stop	**Stop**	deadlocked process	`Stop`
$P \lhd b \rhd Q$	$P \triangleleft b \triangleright_R Q$	reactive if-then-else	`P \triangleleft b \triangleright \sub R Q`
$x := v$	$x :=_C v$	*Circus* assignment	`x := \sub C v`
$x_1, x_2 := e_1, e_2$	$(x_1, x_2) :=_C (e_1, e_2)$	*Circus* multiple assignment	`(x1,x2) := \sub C (e1,e2)`
$\mu X \bullet P(X)$	$\mu_C X \bullet P(X)$	recursive fixed-point	`\mu \sub C X \bullet P(X)`
$a \to P$	$a \longrightarrow P$	simple event prefix	`a \bold\rightarrow P`
$a?x \to P(x)$	$a?x \longrightarrow P(x)$	input event prefix	`a?x \bold\rightarrow P(x)`
$a!v \to P$	$a!v \longrightarrow P$	output event prefix	`a!x \bold\rightarrow P`
$a?x!v \to P(x)$	$a?x!v \longrightarrow P(x)$	mixed event prefix	`a?x!v \bold\rightarrow P(x)`
$b \;\&\; P$	$b \;\&_u\; P$	process guarded by condition b	`b & \sub u P`
$P \Box Q$	$P \Box Q$	binary external choice	`P \box Q`
$\Box\, i : I \bullet P(i)$	$\Box\, i \in I \bullet P(i)$	indexed external choice	`\box i \in I \bullet P(i)`
$P \vert\vert\vert Q$	$P \vert\vert\vert Q$	interleaving	`P ||| Q`
$P \llbracket A \rrbracket Q$	$P [\vert A \vert] Q$	parallel composition	`P [|A|] Q`
$[P \vdash Q \mid R]$	$[P \vdash Q \mid R]_C$	reactive contract / specification	`[ P |- Q | R ]\sub C`
–	$\lceil P \rceil_S$	lift relation on state to process	`\lceil P \rceil \sub s`
–	$\lceil P \rceil_{S<}$	lift predicate on state to process	`\lceil P \rceil \sub s`
–	$(c \cdot v)_u$	construct event over channel c	`(c \cdot v) \sub u`

8 References

[1] A. Cavalcanti and J. Woodcock. A tutorial introduction to CSP in unifying theories of programming. In *Refinement Techniques in Software Engineering*, volume 3167 of *LNCS*, pages 220–268. Springer, 2006.

[2] S. Foster and J. Woodcock. Towards verification of cyber-physical systems with UTP and Isabelle/HOL. In *Concurrency, Security, and Puzzles*, volume 10160 of *LNCS*. Springer, January 2017 2017.

[3] S. Foster and F. Zeyda. Optics in isabelle/hol. *Archive of Formal Proofs*, 2017. `https://www.isa-afp.org/entries/Optics.shtml`.

[4] S. Foster, F. Zeyda, and J. Woodcock. Unifying heterogeneous state-spaces with lenses. In *Proc. 13th Intl. Conf. on Theoretical Aspects of Computing (ICTAC)*, volume 9965 of *LNCS*. Springer, 2016.

[5] T. Hoare and J. He. *Unifying Theories of Programming*. Prentice-Hall, 1998.

[6] M. Oliveira, A. Cavalcanti, and J. Woodcock. A UTP semantics for Circus. *Formal Aspects of Computing*, 21:3–32, 2009.

[7] J. Woodcock and J. Davies. *Using Z: Specification, Refinement, and Proof.* Prentice Hall Series in Computer Science. Prentice-Hall, Apr. 1996.