$U{\cdot}(TP)^2$ User Manual

Andrew Butterfield

November 21, 2013

Contents

Chapter 1

Introduction

1.1 What is $U{\cdot}(TP)^2$?

$U{\cdot}(TP)^2$ is a theorem-proving assistant for Hoare and He's Unifying Theories of Programming (UTP) [HH98]. It was developed as a tool to support foundational work in the UTP, that is, the development of UTP theories. A user-friendly graphical user-interface (GUI) has been designed into the tool from the start.

1.2 Structure of This Document

This is the User Guide for $U{\cdot}(TP)^2$.

1.3 Syntax Guide

1.3.1 Mathematical Syntax Summary

$$
\begin{array}{lcll}
x \in \mathit{Var} & & \text{(given)} & \text{Obs. Variables} \\
k \in \mathit{Const} & & \text{(given)} & \text{Constants} \\
f \in \mathit{FNames} & & \text{(given)} & \text{Function Names} \\
\tau \in \mathit{TVar} & & \text{(given)} & \text{Type Variables} \\
E \in \mathit{ENames} & & \text{(given)} & \text{Expression Metavariable Names} \\
P \in \mathit{PNames} & & \text{(given)} & \text{Predicate Metavariable Names} \\
x\$, E\$, P\$ \in \mathit{LNames} & & \text{(given)} & \text{List Variables} \\
\ell \in \mathit{QVars}_x & ::= & (x \mid x\$)^*_, & \\
\varepsilon \in \mathit{QVars}_E & ::= & (E \mid E\$)^*_, & \\
\varrho \in \mathit{QVars}_P & ::= & (P \mid P\$)^*_, & \text{Binding Lists}
\end{array}
$$

$$
\begin{array}{rcll}
t \in \mathit{Type} & ::= & ? & \text{Arbitrary Type} \\
 & \mid & `\mathbb{B}' & \text{Booleans} \\
 & \mid & `\mathbb{Z}' & \text{Integers} \\
 & \mid & \tau & \text{Type Variable} \\
 & \mid & `\mathcal{P}'\, t & \text{Sets} \\
 & \mid & t^+_{`\times'} & \text{Products} \\
 & \mid & t^{`*'} & \text{Sequences} \\
 & \mid & t`\!\rightarrow\!'t & \text{Functions} \\
 & \mid & `\mathtt{Env}' & \text{Environments } \mathit{Name} \rightarrow \mathit{Type} \\
 & \mid & `\mathtt{free}' & \text{Free Types (to be defined)} \\
 & \mid & `\mu'\, \tau\, `\bullet'\, t & \text{Recursive Types}
\end{array}
$$

$$
\begin{array}{rcll}
e \in \mathit{Expr} & ::= & k & \text{Constants} \\
 & \mid & x & \text{(Obs.) Variables} \\
 & \mid & f\ e & \text{Applications} \\
 & \mid & E & \text{Expr Metavariable} \\
 & \mid & `\lambda'\, \ell\, `\bullet'\, e & \text{Obs. Abstraction} \\
 & \mid & e`['e^+_, `/'x^+_, `]' & \text{Explicit Obs. Substitution} \\
 & \mid & e`['e^+_, `/'E^+_, `]' & \text{Explicit E-var. Substitution} \\
 & \mid & e`['p^+_, `/'P^+_, `]' & \text{Explicit P-var. Substitution} \\
 & \mid & e`\!=\!'e & \text{Equality} \\
 & \mid & `\theta'\, x\, [\, `|'\, p\,]\, `\bullet'\, p & \text{Definite Description}
\end{array}
$$

$p \in \mathit{Pred}$	$::=$	‘`True`’ | ‘`False`’	Constant Predicates
	|	e	Atomic Predicate (Boolean-valued Expr.)
	|	e ‘:’ t	Type Assertion
	|	‘$\mathcal{D}$’ e	Definedness Assertion.
	|	‘$\neg$ ’p	Negation
	|	$p \maltese q$	Composites $\maltese \in \{\wedge, \vee, \Rightarrow, \equiv, \sqcap, \sqsubseteq\}$
	|	P	Explicit Metavariable
	|	¥ℓ [‘|’ p] ‘$\bullet$’p	Observation Quantifiers, ¥ $\in \{\forall, \exists, \exists!\}$
	|	¥ϱ‘$\bullet$’p	Predicate Quantifiers,¥ $\in \{\forall, \exists\}$
	|	¥ε‘$\bullet$’p	Expression Quantifiers, ¥ $\in \{\forall, \exists\}$
	|	‘ ’[p ‘]’	Observation Universal Closure
	|	‘Λ’ ε‘$\bullet$’p	Expression Abstraction
	|	‘Λ’ ϱ‘$\bullet$’p	Predicate Abstraction
	|	$p\ p^+$	Pred-Pred Application
	|	$p\ e^+$	Pred-Expr Application
	|	p‘[’$e^+_{,}/x^+_{,}$‘]’	Explicit Obs. Substitution
	|	p‘[’$e^+_{,}/E^+_{,}$‘]’	Explicit Expr. Substitution
	|	p‘[’$p^+_{,}/P^+_{,}$‘]’	Explicit Pred. Substitution
	|	$p\ ps$	Pred-PredSet Application
	|	p ‘$\in$’ ps	Predicate Set membership
$ps \in \mathit{PredSet}$	$::=$	S	Set Name
	|	‘{’ $p^*_{,}$ ‘}’	Enumeration
	|	‘{’ $P^+_{,}$ [‘|’ p] ‘$\bullet$’ p ‘}’	Comprehension
	|	ps ‘$\bigcup$’ ps	Union

1.3.2 Symbol Conversion Guide

The following table shows how various mathematical symbols are rendered using the ASCII syntax:

Math.	`ASCII`
Variables	
x	`x`
x'	`x'`
x_s	`x_s`
$x\$$	`x$`
Obs, *Mdl*, *Scr*	`O`, `M`, `S`
$Obs\backslash_{x,y}$	`O\x:y`
Types	
?	`?`
$\mathbb{B}$	`B`
$\mathbb{Z}$	`Z`
τ	`t`
$\mathcal{P}$	`P`
$\times$	`x`
*	`*`
$\rightarrow$	`->`
'Env'	`ENV`
Expressions	
λ	`\\`
$\bullet$	`@`
/	`//` (*Expr*)
$=$	`=`
θ	`the`

<table>
<tr><td>Math.</td><td>ASCII</td></tr>
<tr><td colspan="2">Predicates</td></tr>
<tr><td>True</td><td>TRUE</td></tr>
<tr><td>False</td><td>FALSE</td></tr>
<tr><td>$:$</td><td>|:_:|</td></tr>
<tr><td>$\mathcal{D}$</td><td>DEFD</td></tr>
<tr><td>$\neg$</td><td>~</td></tr>
<tr><td>$\wedge$</td><td>/\</td></tr>
<tr><td>$\vee$</td><td>\/</td></tr>
<tr><td>$\Rightarrow$</td><td>=></td></tr>
<tr><td>$\equiv$</td><td>==</td></tr>
<tr><td>$\sqcap$</td><td>|~|</td></tr>
<tr><td>$\sqsubseteq$</td><td>|=</td></tr>
<tr><td>$\forall$</td><td>forall</td></tr>
<tr><td>$\exists$</td><td>exists</td></tr>
<tr><td>$\exists!$</td><td>exists1</td></tr>
<tr><td>$[$</td><td>[</td></tr>
<tr><td>$]$</td><td>]</td></tr>
<tr><td>Λ</td><td>\! (Expr)</td></tr>
<tr><td>Λ</td><td>!! (Pred)</td></tr>
<tr><td>$\in$</td><td>IN PredSet</td></tr>
<tr><td>$\{$</td><td>{} PredSet</td></tr>
<tr><td>$\}$</td><td>}} PredSet</td></tr>
<tr><td>$|$</td><td>|</td></tr>
<tr><td>$\bigcup$</td><td>U</td></tr>
<tr><td>/</td><td>///(Pred)</td></tr>
</table>

1.3.3 Variables

Conceptually, variables have a root and decoration, and if list-variables, may also have a list of 'subtracted' roots

$$(r, d, rs), v \in \mathit{Var} \quad \widehat{=} \quad \mathit{Root} \times \mathit{Decor} \times \mathit{Root}^*$$

The root is a simple name:

$$r \in \mathit{Root} \quad \widehat{=} \quad \mathit{Name}$$

A decoration is either a pre-marking, a post-marking, or a subscript:

$$d \in \mathit{Decor} \quad \widehat{=} \quad \mathit{Pre} \mid \mathit{Post} \mid \mathit{PrePost} \mid \mathit{Subscript}\ \mathit{Name}$$

We use the notation (r, d) when the subtracted-list is empty or irrelevant.

What has just been presented is the *abstract* form of a variable.

Current concrete rendering:

Abstract	Concrete
(r, Pre)	`r`
(r, Post)	`r'`
$(r, \mathit{PrePost})$	`r?`
$(r, \mathit{Subscript}\ s)$	`r_s`

We have defined a sub-class of names as matching observation list-variables, namely those names with decor that starts with `chrLIST`.

We further classify as follows:

Reserved: *Obs*, *Mdl*, *Scr*

Generic: v, e, $\ldots$ (lowercase)

1.3.4 Side Conditions

A table showing side-condition abstract, mathematical and concrete syntaxes:

`SideCond`	Math	ASCII
`SCtrue`	True	`true`
`SCisCond PredM "Q"`	Q a condition	`CND Q`
`SCisCond ExprM "e"`	e un-dashed	`cnd e`
`SCnotFreeIn PredM ["x","y"] "Q"`	$x, y \notin Q$	`Q ## x,y`
`SCnotFreeIn ExprM ["x","y"] "e"`	$x, y \notin e$	`e # x,y`
`SCareTheFreeOf PredM ["x","y"] "Q"`	$x, y = Q$	`Q == x,y`
`SCareTheFreeOf ExprM ["x","y"] "E"`	$x, y = e$	`e = x,y`
`SCcoverTheFreeOf PredM ["x","y"] "Q"`	$x, y \supseteq Q$	`Q << x,y`
`SCcoverTheFreeOf ExprM ["x","y"] "e"`	$x, y \supseteq e$	`e < x,y`
`SCfresh PredM ["Q","R"]`	Q, R fresh	`FRSH Q,R`
`SCfresh ExprM ["x","y"]`	x, y fresh	`frsh x,y`
`SCAnd [sc1,sc2]`	$sc_1 \wedge sc_2$	`sc1 ; sc2`

1.3.5 Language Specifications

The textual form of the language specification is one that matches how the construct would be written, as an interleaving of language elements (basic and list) with tokens, which may be absent. The following character have special roles:

Basic Elements letters `V`, `T`, `E` and `P`

List Element letter `*` or `#` immediately after a basic element.

Whitespace ignored/skipped

Anything else is interpreted as a token, even if it contains the above special characters. The only error is if two tokens occur one after another, with everything else being interpreted as a valid language specification.

For illustration, here are some specifications that correspond to well-known language constructs:

Construct	Specifier	Example
Logical-And	`P*/\`	`P /\Q /\R`
Logical-Or	`P*\/`	`P \/ Q \/ R`
Pred. Forall	`Forall V,* @ P`	`Forall P,Q @ P => Q`
Assignment	`V := E`	`x := y + z`
Sim.–Assignment	`V#, := E#,`	`x,y := y + z,y-1`

Bibliography

[GS93] David Gries and Fred B. Schneider. *A Logical Approach to Discrete Math*. Texts and Monographs in Computer Science. Berlin: Springer Verlag, 1993.

[HH98] C. A. R. Hoare and Jifeng He. *Unifying Theories of Programming*. Prentice-Hall, 1998.

[Tou01] George Tourlakis. On the soundness and completeness of equational predicate logics. *J. Log. Comput.*, 11(4):623–653, 2001.

Appendix A

Appendices