$U{\cdot}(TP)^2$ Reference 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 Reference Manual for $U{\cdot}(TP)^2$. It describes the logic and the naming conventions that are used.

At present this document is a very rough draft, gathering together material mainly from the more literate parts of source code documentation.

Chapter 2

The Logic of $U{\cdot}(TP)^2$

2.1 Overview

The logic of UTP is a variant of the equational logic espoused by David Gries[GS93], with extensions as described below. In formal terms we take the work of Tourlakis [Tou01], and adapt this appropriately.

2.2 Syntax

Particularly in the implementation, we use the terminology "Expression" and "Predicate" where many logicians use "Term" and "Formula". We do not have a clean separation of the two, as we have expressions that define values with the aid of predicates (e.g. set comprehensions, and unique values). Other features to note are the presence of explicit substitutions in the object language—they are not simply part of the meta-language defining inference rules—and the use of explicit pattern-matching meta-variables in quantifier variable lists.

We use the following syntax notation, where ::= | * $^+$ [ ] ( ) ' ' have special meaning:

symbol	meaning
::=	is defined to be
|	separates alternatives
x^*	zero or more x
x^+	one or more x
$x_s^{(*\|+)}$	(zero|one) or more x separated by s
$[x]$	optional x
$(\ldots)$	grouping
'['	the symbol itself

Any other character/symbol denotes itself, and writing $x \in NonT ::= \ldots$ lets x

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

Figure 2.1: $U{\cdot}(\mathit{TP})^2$ Variables

stand for anything that satisfies the the definition of non-terminal *NonT* that occurs on the righthand side of the ::=.

2.2.1 Names

We have a number of variable namespaces (Fig. 2.1):

Observation Variables correspond to term (or "ordinary" variables) in predicate logic, and capture observations of the system behaviour being modelled.

Constants We take all constants (Numeric etc) as "names" for their values.

Function Names are used in applications, as we do not yet support full higher order function notation.

Type Variables are used to support a simple polymorphic type system, whose main purpose is to prevent spurious matches.

Expression Names are names that denote arbitrary expressions (known as term "schematic variables" in some circles).

Predicate Names are names that denote arbitrary predicates (also known as formula "schematic variables").

Variable-List Names are names that denote a list of variables, in contexts where such lists make sense (e.g., quantifier and substitution variable lists). These are distinguished from Observation Variables, Expression and Predicate Names, is that they have a dollar postfix, e.g, $x\$, E\$, P\$$. The only exceptions are the names *OBS*, *MDL* and *SCR* which are taken as Variable-list variables whose meaning is always a (sorted) list of known observation variables (all, model and script respectively).

The key distinction between observation variables and metavariables is what they represent:

Observations are variables that denote values in some kind of semantic domain, with some form of typing used to distinguish different parts of that domain. The meaning of an expression with free observation variables has to be defined w.r.t an environment mapping names to values.

$$\rho \in \mathit{Env} \quad = \quad \mathit{Name} \rightsquigarrow \mathit{Val}$$

Metavariables denote expression or formula syntax. The meaning of an expression or predicate with free metavariables has to be defined w.r.t. an meta-environment (interpretation) mapping names to expressions or predicates.

$$\Phi \in \mathit{EInt} \quad = \quad \mathit{Name} \rightsquigarrow \mathit{Expr}$$

$$\Psi \in \mathit{PInt} \quad = \quad \mathit{Name} \rightsquigarrow \mathit{Pred}$$

List Variables are variables that match lists of variables, where such lists make sense.

A key difference between observations x, y and metavariables X, Y is the outcome of asking for free variables:

$$\begin{aligned} \mathsf{fv}(x \oplus y) &= \{x, y\} \\ \mathsf{fv}(X \bigoplus Y) &= \mathsf{fv}.X \cup \mathsf{fv}.Y \end{aligned}$$

The free variables of a metavariable are dependent on the the meta-environment (interpretation) in place. The requirement to handle free-variables in this way leads to the need to have an explicit notation to describe free-variable sets.

$t \in \mathit{Type}$	$::=$	$?$	Arbitrary Type
	$\mid$	‘$\mathbb{B}$’	Booleans
	$\mid$	‘$\mathbb{Z}$’	Integers
	$\mid$	τ	Type Variable
	$\mid$	‘$\mathcal{P}$’ t	Sets
	$\mid$	$t^{+}_{‘\times’}$	Products
	$\mid$	$t^{‘*’}$	Sequences
	$\mid$	t‘$\to$’t	Functions
	$\mid$	‘`Env`’	Environments $\mathit{Name} \to \mathit{Type}$
	$\mid$	‘`free`’	Free Types (to be defined)
	$\mid$	‘μ’ τ ‘$\bullet$’ t	Recursive Types

Figure 2.2: $U \cdot (\mathit{TP})^2$ Type Syntax

2.2.2 Types (Sorts)

We have the type syntax shown in Figure 2.2. We do not have naturals or reals as basic types, although the former will probably be introduced shortly, to support probabilistic reasoning. The *Env* type is a placeholder for name environments, where the range type is often not expressible using the above type language, because it is some form of universal type.

2.2.3 Expressions (Terms)

The core expression syntax (Fig. 2.3) has constants, variables (for both values and expressions), application of (named) functions[1] , and abstractions and substitutions over observation, expression and predicate variables. Also provided are equality, and definite descriptions, which has predicate components (to be defined). Not shown are the explicit support for binary operators, or booleans, integers, sets, lists and maps. We note here, that unlike terms in [Tou01], our expressions here have quantifiers, and may contain predicates (well-founded formulas) as sub-components.

2.2.4 Predicates (Formulas)

The core logic is higher order, with quantification, abstraction and substitution for variables, expressions and predicates. In particular we have predicates that use predicate-sets (useful for recursion theory, in particular). We also have a simple polymorphic type-system, and a predicate asserting that an expression has a specified type—details to be added. This means that our logic is in fact many-sorted.

[1]We use a definition table to connect function names to abstractions —we should really do a proper higher-order expression model here, but it's not a high priority right now.

$e \in \mathit{Expr}$	$::=$	k	Constants
	$\mid$	x	(Obs.) Variables
	$\mid$	$f\ e$	Applications
	$\mid$	E	Expr Metavariable
	$\mid$	‘λ’ ℓ ‘$\bullet$’ e	Obs. Abstraction
	$\mid$	e‘[’$e^{+}_{,}$‘/’$x^{+}_{,}$‘]’	Explicit Obs. Substitution
	$\mid$	e‘[’$e^{+}_{,}$‘/’$E^{+}_{,}$‘]’	Explicit E-var. Substitution
	$\mid$	e‘[’$p^{+}_{,}$‘/’$P^{+}_{,}$‘]’	Explicit P-var. Substitution
	$\mid$	e‘=’e	Equality
	$\mid$	‘θ’ x [‘$\mid$’ p] ‘$\bullet$’ p	Definite Description

Figure 2.3: $U{\cdot}(TP)^2$ Expression Syntax

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

Figure 2.4: $U{\cdot}(TP)^2$ Predicates (and Sets)

2.3 Substitution

The meaning of $A[x := t]$ in Tourlakis is not given explicitly, but is discussed, particularly with regard to the side-condition designed to avoid variable capture (t substitutable in x (in A)). We shall define $P[x_1, \ldots, x_n := e_1, \ldots, e_n]$ as meta-notation describing the simultaneous syntactical substitution of each e_i for all free occurrences of the corresponding x_i in P, with α-renaming being used to avoid name-capture. We will write this in shorthand as $[\overrightarrow{x} := \overrightarrow{e}]$.

So we have explicit substitutions in the object language (written $[e/x]$), and syntactical-substitutions used to define axiom schemas and inference rules (written $[x := e]$). We use the notation $[\overrightarrow{x} := \overrightarrow{e}] \setminus \overrightarrow{y}$ to denote a substitution with all entries (x_i, e_i) removed where x_i is a member of $\overrightarrow{y}$. We also have a notion of substitution composition, denoted by juxtaposition.

There is also another complication, given that we can abstract over different classes of variables. Consider the following example (with X used as a predicate meta-variable, to make it stand out):

$$((\Lambda X \bullet Q \vee X)R)[e/x]$$

If we evaluation this by reducing the predicate application first we obtain:

$$\begin{array}{ll} & ((\Lambda X \bullet Q \vee X)R)[e/x] \\ \equiv & \quad \text{“ Predicate } \beta\text{-reduction ”} \\ & (Q \vee R)[e/x] \\ \equiv & \quad \text{“ Defn. of observation substitution ”} \\ & Q[e/x] \vee R[e/x] \end{array}$$

Now let's do the substitution first:

$$\begin{array}{ll} & ((\Lambda X \bullet Q \vee X)R)[e/x] \\ \equiv & \quad \text{“ Defn.of observation substitution ”} \\ & ((\Lambda X \bullet Q \vee X)[e/x])(R[e/x]) \end{array}$$

The question now concerns how we evaluate

$$(\Lambda X \bullet Q \vee X)[e/x].$$

If we say that the quantifier applies to predicate variables, but not to observation variables, so it can be ignored, then we obtain

$$\Lambda X \bullet Q[e/x] \vee X[e/x]$$

If we do this and then β-reduce, we obtain

$$Q[e/x] \vee (R[e/x])[e/x]$$

$$
\begin{array}{rcl}
k[\overrightarrow{x} := \overrightarrow{e}] & \widehat{=} & k \\
x[\overrightarrow{x} := \overrightarrow{e}] & \widehat{=} & \begin{cases} e_i, & i \text{ exists, s.t.: } x = x_i \\ x, & x \notin \overrightarrow{x} \end{cases} \\
(f\ e)[\overrightarrow{x} := \overrightarrow{e}] & \widehat{=} & f\,(e[\overrightarrow{x} := \overrightarrow{e}]) \\
E[\overrightarrow{x} := \overrightarrow{e}] & \widehat{=} & \begin{cases} E[\overrightarrow{e}/\overrightarrow{x}], & E \text{ not } \Lambda\text{-bound higher up} \\ E, & E \text{ is } \Lambda\text{-bound higher up} \end{cases} \\
(\lambda\,\overrightarrow{y}, \overrightarrow{q\$} \bullet e)[\overrightarrow{x} := \overrightarrow{e}] & \widehat{=} & \lambda\,\overrightarrow{y}, \overrightarrow{q\$} \bullet (e[\overrightarrow{x} := \overrightarrow{e}] \setminus \overrightarrow{y}) \\
(\Lambda \overrightarrow{E} \bullet e)[\overrightarrow{x} := \overrightarrow{e}] & \widehat{=} & \Lambda \overrightarrow{E} \bullet (e[\overrightarrow{x} := \overrightarrow{e}]) \\
(\Lambda \overrightarrow{P} \bullet e)[\overrightarrow{x} := \overrightarrow{e}] & \widehat{=} & \Lambda \overrightarrow{P} \bullet (e[\overrightarrow{x} := \overrightarrow{e}]) \\
(e[\overrightarrow{f}/\overrightarrow{y}])[\overrightarrow{x} := \overrightarrow{e}] & \widehat{=} & e([\overrightarrow{f}/\overrightarrow{y}][\overrightarrow{e}/\overrightarrow{x}]) \\
(e[\overrightarrow{e}/\overrightarrow{E}])[\overrightarrow{x} := \overrightarrow{e}] & \widehat{=} & \text{need to do inner subst first} \\
(e[\overrightarrow{p}/\overrightarrow{P}])[\overrightarrow{x} := \overrightarrow{e}] & \widehat{=} & \text{need to do inner subst first} \\
(e = e)[\overrightarrow{x} := \overrightarrow{e}] & \widehat{=} & e[\overrightarrow{x} := \overrightarrow{e}] = e[\overrightarrow{x} := \overrightarrow{e}] \\
(\theta x \mid P \bullet Q)[\overrightarrow{x} := \overrightarrow{e}] & \widehat{=} & \theta x \mid P[\overrightarrow{x} := \overrightarrow{e}] \setminus x \bullet Q[\overrightarrow{x} := \overrightarrow{e}] \setminus x
\end{array}
$$

Figure 2.5: Observation substitution for Expressions (modulo α-renaming to avoid variable capture)

This is not the correct outcome, as substitutions are not idempotent (consider $e = x + 1$ in this example).

So we see that the rule for substitution here is context dependent — we cannot apply an observation substitution to any Λ-bound predicate (or expression) meta-variable. This generalises to whenever we substitute for one type of variable (obs/expr/pred) under a lambda binding a different kind. If the variables are $\forall$ or $\exists$-bound, then the substitution is safe.

In Figure 2.5 we show the basic definition of syntactical substitution of expressions for observations, in expressions, without details of how variable capture is avoided. Following figures deal with predicates, and substitution for predicate and expression meta-variables

$$\begin{array}{rcl}
\mathit{True}[\overrightarrow{x} := \overrightarrow{e}] & \widehat{=} & \mathit{True} \\
\mathit{False}[\overrightarrow{x} := \overrightarrow{e}] & \widehat{=} & \mathit{False} \\
e[\overrightarrow{x} := \overrightarrow{e}] & \widehat{=} & \text{see Figure 2.5} \\
(e : t)[\overrightarrow{x} := \overrightarrow{e}] & \widehat{=} & (e[\overrightarrow{x} := \overrightarrow{e}] : t) \\
(\mathcal{D}\, e)[\overrightarrow{x} := \overrightarrow{e}] & \widehat{=} & \mathcal{D}\,(e[\overrightarrow{x} := \overrightarrow{e}]) \\
(\neg\, p)[\overrightarrow{x} := \overrightarrow{e}] & \widehat{=} & \neg\,(p[\overrightarrow{x} := \overrightarrow{e}]) \\
(p \maltese q)[\overrightarrow{x} := \overrightarrow{e}] & \widehat{=} & p[\overrightarrow{x} := \overrightarrow{e}] \maltese q[\overrightarrow{x} := \overrightarrow{e}] \\
P[\overrightarrow{x} := \overrightarrow{e}] & \widehat{=} & \begin{cases} P[\overrightarrow{e}/\overrightarrow{x}], & P \text{ not } \Lambda\text{-bound higher up} \\ P, & P \text{ is } \Lambda\text{-bound higher up} \end{cases} \\
(\yen\, \overrightarrow{y}, \overrightarrow{q\$} \mid p \bullet q)[\overrightarrow{x} := \overrightarrow{e}] & \widehat{=} & \yen\, \overrightarrow{y}, \overrightarrow{q\$} \mid (p[\overrightarrow{x} := \overrightarrow{e}] \setminus \overrightarrow{y}) \bullet (q[\overrightarrow{x} := \overrightarrow{e}] \setminus \overrightarrow{y})) \\
(\yen\, \overrightarrow{P} \bullet p)[\overrightarrow{x} := \overrightarrow{e}] & \widehat{=} & \yen\, \overrightarrow{P} \bullet (p[\overrightarrow{x} := \overrightarrow{e}]) \\
(\yen\, \overrightarrow{E} \bullet p)[\overrightarrow{x} := \overrightarrow{e}] & \widehat{=} & \yen\, \overrightarrow{E} \bullet (p[\overrightarrow{x} := \overrightarrow{e}]) \\
[p][\overrightarrow{x} := \overrightarrow{e}] & \widehat{=} & [p] \\
(\Lambda\, \overrightarrow{y}, \overrightarrow{q\$} \bullet p)[\overrightarrow{x} := \overrightarrow{e}] & \widehat{=} & \Lambda\, \overrightarrow{y}, \overrightarrow{q\$} \bullet (p[\overrightarrow{x} := \overrightarrow{e}] \setminus \overrightarrow{y}) \\
(\Lambda\, \overrightarrow{P} \bullet p)[\overrightarrow{x} := \overrightarrow{e}] & \widehat{=} & \Lambda\, \overrightarrow{P} \bullet (p[\overrightarrow{x} := \overrightarrow{e}]) \\
(p\ \overrightarrow{q\$})[\overrightarrow{x} := \overrightarrow{e}] & \widehat{=} & (p[\overrightarrow{x} := \overrightarrow{e}]\ \overrightarrow{q\$}[\overrightarrow{x} := \overrightarrow{e}]) \\
(p\ \overrightarrow{f})[\overrightarrow{x} := \overrightarrow{e}] & \widehat{=} & (p[\overrightarrow{x} := \overrightarrow{e}]\ \overrightarrow{f}[\overrightarrow{x} := \overrightarrow{e}]) \\
(p[\overrightarrow{f}/\overrightarrow{y}])[\overrightarrow{x} := \overrightarrow{e}] & \widehat{=} & p([\overrightarrow{f}/\overrightarrow{y}][\overrightarrow{e}/\overrightarrow{x}]) \\
(p[\overrightarrow{e}/\overrightarrow{E}])[\overrightarrow{x} := \overrightarrow{e}] & \widehat{=} & \text{need to do inner subst first} \\
(p[\overrightarrow{p}/\overrightarrow{P}])[\overrightarrow{x} := \overrightarrow{e}] & \widehat{=} & \text{need to do inner subst first} \\
(p\ S)[\overrightarrow{x} := \overrightarrow{e}] & \widehat{=} & p[\overrightarrow{x} := \overrightarrow{e}]\ S[\overrightarrow{x} := \overrightarrow{e}] \\
(p \in S)[\overrightarrow{x} := \overrightarrow{e}] & \widehat{=} & p[\overrightarrow{x} := \overrightarrow{e}] \in S[\overrightarrow{x} := \overrightarrow{e}] \\
 & & \\
S[\overrightarrow{x} := \overrightarrow{e}] & \widehat{=} & S[\overrightarrow{e}/\overrightarrow{x}] \\
\{\overrightarrow{p}\}[\overrightarrow{x} := \overrightarrow{e}] & \widehat{=} & \{\overrightarrow{p}[\overrightarrow{x} := \overrightarrow{e}]\} \\
\{\overrightarrow{P} \mid r \bullet p\}[\overrightarrow{x} := \overrightarrow{e}] & \widehat{=} & \{\overrightarrow{P} \mid r[\overrightarrow{x} := \overrightarrow{e}] \bullet p[\overrightarrow{x} := \overrightarrow{e}]\} \\
(ps_1 \bigcup ps_2)[\overrightarrow{x} := \overrightarrow{e}] & \widehat{=} & ps_1[\overrightarrow{x} := \overrightarrow{e}] \bigcup ps_2[\overrightarrow{x} := \overrightarrow{e}]
\end{array}$$

Figure 2.6: Observation substitution for Predicates (modulo α-renaming to avoid variable capture)

$$
\begin{array}{rcl}
k[\overrightarrow{E} := \overrightarrow{e}] & \widehat{=} & k \\
x[\overrightarrow{E} := \overrightarrow{e}] & \widehat{=} & x \\
(f\ e)[\overrightarrow{E} := \overrightarrow{e}] & \widehat{=} & f\,(e[\overrightarrow{E} := \overrightarrow{e}]) \\
E[\overrightarrow{E} := \overrightarrow{e}] & \widehat{=} & \begin{cases} e_i, & i \text{ exists, s.t.: } E = E_i \\ E, & E \notin \overrightarrow{E} \end{cases} \\
(\lambda\,\overrightarrow{x}, \overrightarrow{q\$} \bullet e)[\overrightarrow{E} := \overrightarrow{e}] & \widehat{=} & \lambda\,\overrightarrow{x}, \overrightarrow{q\$} \bullet (e[\overrightarrow{E} := \overrightarrow{e}]) \\
(\Lambda \overrightarrow{F} \bullet e)[\overrightarrow{E} := \overrightarrow{e}] & \widehat{=} & \Lambda \overrightarrow{F} \bullet (e[\overrightarrow{E} := \overrightarrow{e}] \setminus \overrightarrow{F}) \\
(\Lambda \overrightarrow{P} \bullet e)[\overrightarrow{E} := \overrightarrow{e}] & \widehat{=} & \Lambda \overrightarrow{P} \bullet (e[\overrightarrow{E} := \overrightarrow{e}]) \\
(e[\overrightarrow{e}/\overrightarrow{x}])[\overrightarrow{E} := \overrightarrow{e}] & \widehat{=} & \text{need to do inner subst first} \\
(e[\overrightarrow{f}\,\overrightarrow{F}])[\overrightarrow{E} := \overrightarrow{e}] & \widehat{=} & e([\overrightarrow{f}/\overrightarrow{F}][\overrightarrow{e}/\overrightarrow{E}]) \\
(e[\overrightarrow{p}/\overrightarrow{P}])[\overrightarrow{E} := \overrightarrow{e}] & \widehat{=} & \text{need to do inner subst first} \\
(e = e)[\overrightarrow{E} := \overrightarrow{e}] & \widehat{=} & e[\overrightarrow{E} := \overrightarrow{e}] = e[\overrightarrow{E} := \overrightarrow{e}] \\
(\theta x \mid P \bullet Q)[\overrightarrow{E} := \overrightarrow{e}] & \widehat{=} & \theta x \mid P[\overrightarrow{E} := \overrightarrow{e}] \bullet Q[\overrightarrow{E} := \overrightarrow{e}]
\end{array}
$$

Figure 2.7: Expression substitution for Expressions (modulo α-renaming to avoid variable capture)

$$
\begin{array}{rcl}
k[\overrightarrow{P} := \overrightarrow{p}] & \widehat{=} & k \\
x[\overrightarrow{P} := \overrightarrow{p}] & \widehat{=} & x \\
(f\ e)[\overrightarrow{P} := \overrightarrow{p}] & \widehat{=} & f\,(e[\overrightarrow{P} := \overrightarrow{p}]) \\
E[\overrightarrow{P} := \overrightarrow{p}] & \widehat{=} & \begin{cases} E[\overrightarrow{p}/\overrightarrow{P}], & E \text{ not } \Lambda\text{-bound higher up} \\ E, & E \text{ is } \Lambda\text{-bound higher up} \end{cases} \\
(\lambda\,\overrightarrow{x}, \overrightarrow{q\$} \bullet e)[\overrightarrow{P} := \overrightarrow{p}] & \widehat{=} & \lambda\,\overrightarrow{x}, \overrightarrow{q\$} \bullet (e[\overrightarrow{P} := \overrightarrow{p}]) \\
(\Lambda \overrightarrow{F} \bullet e)[\overrightarrow{P} := \overrightarrow{p}] & \widehat{=} & \Lambda \overrightarrow{F} \bullet (e[\overrightarrow{P} := \overrightarrow{p}] \setminus \overrightarrow{F}) \\
(\Lambda \overrightarrow{P} \bullet e)[\overrightarrow{P} := \overrightarrow{p}] & \widehat{=} & \Lambda \overrightarrow{P} \bullet (e[\overrightarrow{P} := \overrightarrow{p}]) \\
(e[\overrightarrow{e}/\overrightarrow{x}])[\overrightarrow{P} := \overrightarrow{p}] & \widehat{=} & \text{need to do inner subst first} \\
(e[\overrightarrow{e}/\overrightarrow{E}])[\overrightarrow{P} := \overrightarrow{p}] & \widehat{=} & \text{need to do inner subst first} \\
(e[\overrightarrow{q\$}/\overrightarrow{Q}])[\overrightarrow{P} := \overrightarrow{p}] & \widehat{=} & e([\overrightarrow{q\$}/\overrightarrow{Q}][\overrightarrow{p}/\overrightarrow{P}]) \\
(e = e)[\overrightarrow{P} := \overrightarrow{p}] & \widehat{=} & e[\overrightarrow{P} := \overrightarrow{p}] = e[\overrightarrow{P} := \overrightarrow{p}] \\
(\theta x \mid P \bullet Q)[\overrightarrow{P} := \overrightarrow{p}] & \widehat{=} & \theta x \mid P[\overrightarrow{P} := \overrightarrow{p}] \bullet Q[\overrightarrow{P} := \overrightarrow{p}]
\end{array}
$$

Figure 2.8: Predicate substitution for Expressions (modulo α-renaming to avoid variable capture)

$$
\begin{array}{rcl}
\mathit{True}[\overrightarrow{E} := \overrightarrow{e}] & \widehat{=} & \mathit{True} \\
\mathit{False}[\overrightarrow{E} := \overrightarrow{e}] & \widehat{=} & \mathit{False} \\
e[\overrightarrow{E} := \overrightarrow{e}] & \widehat{=} & \text{see Figure 2.7} \\
(e : t)[\overrightarrow{E} := \overrightarrow{e}] & \widehat{=} & (e[\overrightarrow{E} := \overrightarrow{e}] : t) \\
(\mathcal{D}\, e)[\overrightarrow{E} := \overrightarrow{e}] & \widehat{=} & \mathcal{D}\,(e[\overrightarrow{E} := \overrightarrow{e}]) \\
(\neg\, p)[\overrightarrow{E} := \overrightarrow{e}] & \widehat{=} & \neg\,(p[\overrightarrow{E} := \overrightarrow{e}]) \\
(p \maltese q)[\overrightarrow{E} := \overrightarrow{e}] & \widehat{=} & p[\overrightarrow{E} := \overrightarrow{e}] \maltese q[\overrightarrow{E} := \overrightarrow{e}] \\
P[\overrightarrow{E} := \overrightarrow{e}] & \widehat{=} & \begin{cases} P[\overrightarrow{e}/\overrightarrow{x}], & P \text{ not } \Lambda\text{-bound higher up} \\ P, & P \text{ is } \Lambda\text{-bound higher up} \end{cases} \\
(\yen\,\overrightarrow{y}, \overrightarrow{q\$} \mid p \bullet q)[\overrightarrow{E} := \overrightarrow{e}] & \widehat{=} & \yen\,\overrightarrow{y}, \overrightarrow{q\$} \mid (p[\overrightarrow{E} := \overrightarrow{e}]) \bullet (q[\overrightarrow{E} := \overrightarrow{e}])) \\
(\yen\,\overrightarrow{P} \bullet p)[\overrightarrow{E} := \overrightarrow{e}] & \widehat{=} & \yen\,\overrightarrow{P} \bullet (p[\overrightarrow{E} := \overrightarrow{e}]) \\
(\yen\,\overrightarrow{F} \bullet p)[\overrightarrow{E} := \overrightarrow{e}] & \widehat{=} & \yen\,\overrightarrow{F} \bullet (p[\overrightarrow{E} := \overrightarrow{e}] \setminus \overrightarrow{F}) \\
[\,p\,][\overrightarrow{E} := \overrightarrow{e}] & \widehat{=} & [\,p[\overrightarrow{E} := \overrightarrow{e}]\,] \\
(\Lambda\,\overrightarrow{y}, \overrightarrow{q\$} \bullet p)[\overrightarrow{E} := \overrightarrow{e}] & \widehat{=} & \Lambda\,\overrightarrow{y}, \overrightarrow{q\$} \bullet (p[\overrightarrow{E} := \overrightarrow{e}]) \\
(\Lambda\,\overrightarrow{P} \bullet p)[\overrightarrow{E} := \overrightarrow{e}] & \widehat{=} & \Lambda\,\overrightarrow{P} \bullet (p[\overrightarrow{E} := \overrightarrow{e}]) \\
(p\ \overrightarrow{q\$})[\overrightarrow{E} := \overrightarrow{e}] & \widehat{=} & (p[\overrightarrow{E} := \overrightarrow{e}]\ \overrightarrow{q\$}[\overrightarrow{E} := \overrightarrow{e}]) \\
(p\ \overrightarrow{f}\,)[\overrightarrow{E} := \overrightarrow{e}] & \widehat{=} & (p[\overrightarrow{E} := \overrightarrow{e}]\ \overrightarrow{f}\,[\overrightarrow{E} := \overrightarrow{e}]) \\
(p[\overrightarrow{f}/\overrightarrow{y}])[\overrightarrow{E} := \overrightarrow{e}] & \widehat{=} & \text{need to do inner subst first} \\
(p[\overrightarrow{e}/\overrightarrow{E}])[\overrightarrow{E} := \overrightarrow{e}] & \widehat{=} & p([\overrightarrow{e}/\overrightarrow{E}][\overrightarrow{e}/\overrightarrow{E}] \\
(p[\overrightarrow{p}/\overrightarrow{P}])[\overrightarrow{E} := \overrightarrow{e}] & \widehat{=} & \text{need to do inner subst first} \\
(p\ S)[\overrightarrow{E} := \overrightarrow{e}] & \widehat{=} & p[\overrightarrow{E} := \overrightarrow{e}]\ S[\overrightarrow{E} := \overrightarrow{e}] \\
(p \in S)[\overrightarrow{E} := \overrightarrow{e}] & \widehat{=} & p[\overrightarrow{E} := \overrightarrow{e}] \in S[\overrightarrow{E} := \overrightarrow{e}] \\
 & & \\
S[\overrightarrow{E} := \overrightarrow{e}] & \widehat{=} & S[\overrightarrow{e}/\overrightarrow{E}] \\
\{\overrightarrow{p}\}[\overrightarrow{E} := \overrightarrow{e}] & \widehat{=} & \{\overrightarrow{p}[\overrightarrow{E} := \overrightarrow{e}]\} \\
\{\overrightarrow{P} \mid r \bullet p\}[\overrightarrow{E} := \overrightarrow{e}] & \widehat{=} & \{\overrightarrow{P} \mid r[\overrightarrow{E} := \overrightarrow{e}] \bullet p[\overrightarrow{E} := \overrightarrow{e}]\} \\
(ps_1 \bigcup ps_2)[\overrightarrow{E} := \overrightarrow{e}] & \widehat{=} & ps_1[\overrightarrow{E} := \overrightarrow{e}] \bigcup ps_2[\overrightarrow{E} := \overrightarrow{e}]
\end{array}
$$

Figure 2.9: Expression substitution for Predicates (modulo α-renaming to avoid variable capture)

$$
\begin{array}{rcl}
\mathit{True}[\overrightarrow{P} := \overrightarrow{p}] & \widehat{=} & \mathit{True} \\
\mathit{False}[\overrightarrow{P} := \overrightarrow{p}] & \widehat{=} & \mathit{False} \\
e[\overrightarrow{P} := \overrightarrow{p}] & \widehat{=} & \text{see Figure 2.8} \\
(e : t)[\overrightarrow{P} := \overrightarrow{p}] & \widehat{=} & (e[\overrightarrow{P} := \overrightarrow{p}] : t) \\
(\mathcal{D}\, e)[\overrightarrow{P} := \overrightarrow{p}] & \widehat{=} & \mathcal{D}\,(e[\overrightarrow{P} := \overrightarrow{p}]) \\
(\neg\, p)[\overrightarrow{P} := \overrightarrow{p}] & \widehat{=} & \neg\,(p[\overrightarrow{P} := \overrightarrow{p}]) \\
(p \maltese q)[\overrightarrow{P} := \overrightarrow{p}] & \widehat{=} & p[\overrightarrow{P} := \overrightarrow{p}] \maltese q[\overrightarrow{P} := \overrightarrow{p}] \\
P[\overrightarrow{P} := \overrightarrow{p}] & \widehat{=} & \begin{cases} p_i, & P \text{ not bound higher} \\ & \text{and exists } i \text{ s.t.: } P = P_i \\ P, & \text{otherwise} \end{cases} \\
(\yen\, \overrightarrow{y}, \overrightarrow{q\$} \mid p \bullet q)[\overrightarrow{P} := \overrightarrow{p}] & \widehat{=} & \yen\, \overrightarrow{y}, \overrightarrow{q\$} \mid (p[\overrightarrow{P} := \overrightarrow{p}]) \bullet (q[\overrightarrow{P} := \overrightarrow{p}])) \\
(\yen\, \overrightarrow{Q} \bullet p)[\overrightarrow{P} := \overrightarrow{p}] & \widehat{=} & \yen\, \overrightarrow{Q} \bullet (p[\overrightarrow{P} := \overrightarrow{p}] \setminus \overrightarrow{Q}) \\
(\yen\, \overrightarrow{F} \bullet p)[\overrightarrow{P} := \overrightarrow{p}] & \widehat{=} & \yen\, \overrightarrow{F} \bullet (p[\overrightarrow{P} := \overrightarrow{p}]) \\
[\,p\,][\overrightarrow{P} := \overrightarrow{p}] & \widehat{=} & [\,p[\overrightarrow{P} := \overrightarrow{p}]\,] \\
(\Lambda\, \overrightarrow{y}, \overrightarrow{q\$} \bullet p)[\overrightarrow{P} := \overrightarrow{p}] & \widehat{=} & \Lambda\, \overrightarrow{y}, \overrightarrow{q\$} \bullet (p[\overrightarrow{P} := \overrightarrow{p}]) \\
(\Lambda\, \overrightarrow{Q} \bullet p)[\overrightarrow{P} := \overrightarrow{p}] & \widehat{=} & \Lambda\, \overrightarrow{P} \bullet (p[\overrightarrow{P} := \overrightarrow{p}] \setminus \overrightarrow{Q}) \\
(p\; \overrightarrow{q\$})[\overrightarrow{P} := \overrightarrow{p}] & \widehat{=} & (p[\overrightarrow{P} := \overrightarrow{p}]\; \overrightarrow{q\$}[\overrightarrow{P} := \overrightarrow{p}]) \\
(p\; \overrightarrow{f}\,)[\overrightarrow{P} := \overrightarrow{p}] & \widehat{=} & (p[\overrightarrow{P} := \overrightarrow{p}]\; \overrightarrow{f}\,[\overrightarrow{P} := \overrightarrow{p}]) \\
(p[\overrightarrow{f} / \overrightarrow{y}])[\overrightarrow{P} := \overrightarrow{p}] & \widehat{=} & \text{need to do inner subst first} \\
(p[\overrightarrow{e} / \overrightarrow{E}])[\overrightarrow{P} := \overrightarrow{p}] & \widehat{=} & \text{need to do inner subst first} \\
(p[\overrightarrow{q\$} / \overrightarrow{Q}])[\overrightarrow{P} := \overrightarrow{p}] & \widehat{=} & (p([\overrightarrow{q\$} / \overrightarrow{Q}][\overrightarrow{p} / \overrightarrow{P}]) \\
(p\; S)[\overrightarrow{P} := \overrightarrow{p}] & \widehat{=} & p[\overrightarrow{P} := \overrightarrow{p}]\; S[\overrightarrow{P} := \overrightarrow{p}] \\
(p \in S)[\overrightarrow{P} := \overrightarrow{p}] & \widehat{=} & p[\overrightarrow{P} := \overrightarrow{p}] \in S[\overrightarrow{P} := \overrightarrow{p}] \\
\\
S[\overrightarrow{P} := \overrightarrow{p}] & \widehat{=} & S[\overrightarrow{p} / \overrightarrow{P}] \\
\{\overrightarrow{p}\}[\overrightarrow{P} := \overrightarrow{p}] & \widehat{=} & \{\overrightarrow{p}[\overrightarrow{P} := \overrightarrow{p}]\} \\
\{\overrightarrow{Q} \mid r \bullet p\}[\overrightarrow{P} := \overrightarrow{p}] & \widehat{=} & \{\overrightarrow{P} \mid r[\overrightarrow{P} := \overrightarrow{p}] \setminus \overrightarrow{Q} \bullet p[\overrightarrow{P} := \overrightarrow{p}] \setminus \overrightarrow{Q}\} \\
(ps_1 \bigcup ps_2)[\overrightarrow{P} := \overrightarrow{p}] & \widehat{=} & ps_1[\overrightarrow{P} := \overrightarrow{p}] \bigcup ps_2[\overrightarrow{P} := \overrightarrow{p}]
\end{array}
$$

Figure 2.10: Predicate substitution for Predicates (modulo α-renaming to avoid variable capture)

2.4 Free Variable Set Notation

In determining free observation variables we have to deal with the presence of meta-variables denoting arbitrary predicates and expressions, as well as the presence of explicit substitutions, and quantifier list-variables that can denote lists of variables or expressions. This means that the free variables of a predicate or expression are contingent on not just the meta/list-variables, but whether or not certain observation variables are free/present in (any instantiation of) those meta/list-variables.

We shall use the lambda calculus to illustrate the consequences of having explicit substitution and quantifier-list matching in our logic.

We start with the untyped lamdba-calculus (L_0) where we have an unbounded set of variables ($v \in V$):

$$\begin{array}{rcl} v, w, x \in V & & \text{Given.} \\ e, f \in L_0 & ::= & v \mid e\, e \mid \lambda\, v \bullet e \end{array}$$

with $\lambda\, x_1, x_2, \ldots, x_n \bullet e$ as syntactic sugar for $\lambda\, x_1 \bullet \lambda\, x_2 \bullet \ldots \lambda\, x_n \bullet e$, itself often shortened to $\lambda\, \overrightarrow{x} \bullet e$.

Notation aside: We shall assume that $\overrightarrow{a}$ is shorthand for $a_1, \ldots, a_n$, for $n \geq 0$, and a_i will refer to the ith component, or indicate an iteration $i \in 1 \ldots n$, depending on context.

The free variables for L_0 are defined in the usual way:

$$\begin{array}{rcl} S_0 & = & \mathcal{P}\, V \\ \mathsf{fv}, \mathsf{fv}_0 & : & L_0 \to S_0 \\ \mathsf{fv}(v) & \widehat{=} & \{v\} \\ \mathsf{fv}(f\ e) & \widehat{=} & \mathsf{fv}(f) \cup \mathsf{fv}(e) \\ \mathsf{fv}(\lambda\, v \bullet e) & \widehat{=} & \mathsf{fv}(e) \setminus \{v\} \end{array}$$

So is substitution:

$$\begin{array}{rcl} \_[\_ := \_] & : & L_0 \to V \times L_0 \to L_0 \\ v[x := e] & \widehat{=} & e \lhd x = v \rhd v \\ (f\ e)[x := e] & \widehat{=} & (f[x := e])\,(e[x := e]) \\ (\lambda\, v \bullet f)[x := e] & \widehat{=} & \left\{ \begin{array}{ll} \lambda\, v \bullet f, & x = v \\ \lambda\, v \bullet f[x := e], & x \neq v \land v \notin \mathsf{fv}(e) \\ \lambda\, w \bullet f[v := w][x := e], & x \neq v \land v \in \mathsf{fv}(e) \land \text{new}\, w \end{array} \right. \end{array}$$

This extends to simultaneous substitutions as follows (here $\overrightarrow{x}$ has no duplicates):

$$\begin{array}{rcl}
\_[\_ := \_] & : & L_0 \to (V^n \times (L_0)^n) \to L_0 \\
e[\langle\rangle := \langle\rangle] & \widehat{=} & e, \text{ the empty case} \\
v[\overrightarrow{x} := \overrightarrow{e}] & \widehat{=} & e_i \lhd v = x_i \rhd v, \text{ for some } i \\
(f\ e)[\overrightarrow{x} := \overrightarrow{e}] & \widehat{=} & (f[\overrightarrow{x} := \overrightarrow{e}])\,(e[\overrightarrow{x} := \overrightarrow{e}]) \\
(\lambda\, v \bullet f)[\overrightarrow{x} := \overrightarrow{e}] & \widehat{=} & \left\{ \begin{array}{ll} \lambda\, w \bullet f[v := w][\overrightarrow{x} \setminus x_i := \overrightarrow{e} \setminus e_i], & v = x_i \\ \lambda\, w \bullet f[v := w][\overrightarrow{x} := \overrightarrow{e}], & v \notin \overrightarrow{x} \end{array} \right. \text{new } w
\end{array}$$

This then leads to key theorems regarding free variables and substitution:

$$\mathsf{fv}(e[x := f]) = \mathsf{fv}(e) \setminus \{x\} \cup (\mathsf{fv}(f) \lhd x \in \mathsf{fv}(e) \rhd \emptyset) \tag{2.1}$$

$$\mathsf{fv}(e[\overrightarrow{x} := \overrightarrow{e}]) = \mathsf{fv}(e) \setminus \overrightarrow{x} \cup \bigcup_i \{\mathsf{fv}(e_i) \lhd x_i \in \mathsf{fv}(e) \rhd \emptyset\} \tag{2.2}$$

The first is provable, with care, by induction on e (even the base-case is non-trivial !).

We then extend our language to include explicit expression meta-variables ($E \in M$) and explicit simultaneous substitutions:

$$\begin{array}{rcl}
E \in M & & \text{Given.} \\
e, f \in L_1 & ::+ & E \mid e[\overrightarrow{f}/\overrightarrow{x}]
\end{array}$$

Here it is understood that vectors $\overrightarrow{f}$ and $\overrightarrow{x}$ are of the same length and the latter has no duplicates.

We now extend the notion of free variables, but note that we cannot "expand" the application of fv to E. The effect of this is that we no longer can return a set of variables, but must instead return an expression ($s \in S_1$) denoting such a set as a function of its explicit meta-variables:

$$s \in S_1 \quad ::= \quad \{\overrightarrow{v}\} \mid E \mid s \setminus s \mid \bigcup\{\overrightarrow{s}\} \mid v \in s \Rrightarrow s$$

The last construction is a conditional: for $v \in s_0 \Rrightarrow s_1$, if the condition holds, then it denotes s_1, otherwise it denotes the empty set.

There are obvious injections L^{01} and S^{01} that embed $e_0 : L_0$ and $s_0 : S_0$ into L_1 and S_1 respectively.

Given an environment (ρ_M) mapping E to values from L_0, we can then map both L_1 to L_0 and S_1 to S_0 as follows (where we drop $_M$ in most cases as it is obvious from context):

$$\begin{array}{rcl}
\rho_M & : & M \to L_0 \\
L^{10} & : & (M \to L_0) \to L_1 \to L_0 \\
L^{10}_\rho(v) & \widehat{=} & v \\
L^{10}_\rho(f\ e) & \widehat{=} & (L^{10}_\rho(f))\,(L^{10}_\rho(e)) \\
L^{10}_\rho(\lambda\, v \bullet e) & \widehat{=} & \lambda\, v \bullet (L^{10}_\rho(e)) \\
L^{10}_\rho(E) & \widehat{=} & \rho(E) \\
L^{10}_\rho(e[\overrightarrow{f}/\overrightarrow{x}]) & \widehat{=} & (L^{10}_\rho(e))[\overrightarrow{x} := \overrightarrow{f}\,]
\end{array}$$

$$\begin{array}{rcl}
S^{10} & : & (M \to L_0) \to S_1 \to S_0 \\
S^{10}_\rho(\{\overrightarrow{v}\}) & \widehat{=} & \{\overrightarrow{v}\} \\
S^{10}_\rho(E) & \widehat{=} & \mathsf{fv}_0(\rho(E)) \\
S^{10}_\rho(s_1 \setminus s_2) & \widehat{=} & S^{10}_\rho(s_1) \setminus S^{10}_\rho(s_2) \\
S^{10}_\rho(\bigcup\{\overrightarrow{s}\}) & \widehat{=} & \bigcup\{\overrightarrow{S^{10}_\rho(s)}\} \\
S^{10}_\rho(v \in s_0 \Rrightarrow s_1) & \widehat{=} & S^{10}_\rho(s_1) \lhd v \in S^{10}_\rho(s_0) \rhd \emptyset
\end{array}$$

We can extend fv to L_1 as follows:

$$\begin{array}{rcl}
\mathsf{fv}, \mathsf{fv}_1 & : & L_1 \to S_1 \\
\mathsf{fv}(x) & \widehat{=} & \{x\} \\
\mathsf{fv}(e_1\ e_2) & \widehat{=} & \bigcup(\mathsf{fv}\, e_1, \mathsf{fv}\, e_2) \\
\mathsf{fv}(\lambda\, x \bullet e) & \widehat{=} & (\mathsf{fv}\, e) \setminus \{x\} \\
\mathsf{fv}(E) & \widehat{=} & E \\
\mathsf{fv}(e[\overrightarrow{f}/\overrightarrow{x}]) & \widehat{=} & \mathsf{fv}(e) \setminus \{\overrightarrow{x}\} \cup \bigcup_i \{x_i \in \mathsf{fv}(e) \Rrightarrow \mathsf{fv}(f_i)\}
\end{array}$$

For a substitution, we see that the presence of the free variables of a replacement expression (e_i) is contingent on the presence of the corresponding *target* variable (x_i) in the free variables of the *base* expression (e). If the base expression is a meta-variable, then we get the following instantiation of the last law:

$$\mathsf{fv}(E)[e_1, \ldots, e_n/x_1, \ldots, x_n] \quad = \quad E \setminus \{x_1, \ldots, x_n\} \cup \bigcup\{x_i \in E \bullet\!\Rrightarrow e_i\}$$

We cannot either perform the set difference operation, nor evaluate any of the conditionals. In effect, in order to give an accurate description of the free variables of this language we need to return a (variable-set valued) expression that describes how the resulting set of variables is contingent upon the (yet to be determined) free variables of the meta-variables. This is the motivation for the $\Rrightarrow$ construct in S_1.

We now find that we can construct the following diagram relating the L_i and S_i:

$$\begin{tikzcd}
L_1 \arrow[r, "\mathsf{fv}_1"] \arrow[d, bend left, "L^{10}_\rho"] & S_1 \arrow[d, bend left, "S^{10}_\rho"] \\
L_0 \arrow[r, "\mathsf{fv}_0"'] \arrow[u, bend left, "L^{01}"] & S_0 \arrow[u, bend left, "S^{01}"]
\end{tikzcd}$$

From this we can immediately suggest a few lemmas/theorems:

$$
\begin{aligned}
L^{10}_\rho(L^{01}(e_0)) &= e_0, \text{ for all } \rho & (2.3)\\
S^{10}_\rho(S^{01}(s_0)) &= s_0, \text{ for all } \rho & (2.4)\\
\mathsf{fv}_1(L^{01}(e_0)) &= S^{01}(\mathsf{fv}_0(e_0)) & (2.5)\\
S^{10}_\rho(\mathsf{fv}_1(e_1)) &= \mathsf{fv}_0(L^{10}_\rho(e_1)), \text{ for all } \rho & (2.6)
\end{aligned}
$$

Proofs of these are by induction over the leftmost e_i in each case, and are best done in the order given above. The following is an easy consequence of the above:

$$
\begin{aligned}
& S^{10}_\rho(\mathsf{fv}_1(L^{01}(e_0)))\\
= \;& S^{10}_\rho(S^{01}(\mathsf{fv}_0(e_0)))\\
= \;& \mathsf{fv}_0(e_0)
\end{aligned}
$$

At this point we need to extend the language further to have explicit quantifier meta-variables that stand for lists of ordinary variables or corresponding lists of expressions:

$$q\$, r\$ \in Q \qquad \text{Given.}$$

We then extend our language again (where $\overrightarrow{q\$}$, like $\overrightarrow{x}$, has no duplicates):

$$e, f \in L_2 \quad ::+ \quad \lambda\, \overrightarrow{v}, \overrightarrow{q\$} \bullet e \mid e[\overrightarrow{f}, \overrightarrow{r\$} / \overrightarrow{x}, \overrightarrow{q\$}]$$

As before, there is an obvious injection $L^{12} : L_1 \to L_2$, as well as $L^{02} : L_0 \to L_2$, and the syntactic sugar $\lambda\, \overrightarrow{x} \bullet e$ is now a proper part of L_2. In effect the L_2 extensions subsume the lambda and substitution constructs of L_1.

The $q\$$ and $r\$$ are intended to denote lists (possibly empty) of ordinary variables ($\overrightarrow{v}$) and expressions ($\overrightarrow{f}$) respectively. To this end we introduce two new environments:

$$\begin{array}{lcl}\rho_V & : & Q \rightarrow V^* \\ \rho_E & : & Q \rightarrow L_1^*\end{array}$$

We can use these to define the conversion

$$L^{21}_{(\rho_V,\rho_E)} : L_2 \rightarrow L_1,$$

and coupled with ρ_M, we can then get

$$L^{20}_{(\rho_V,\rho_E,\rho_M)} : L_2 \rightarrow L_0.$$

In the sequel we shall often use ρ to denote one of the above when it is obvious from context, or to denote the entire tuple-parameter of a conversion: i.e. L^{20}_ρ. We can now define the conversion:

$$\begin{array}{rcl} L^{21} & : & (Q \rightarrow V^*) \times (Q \rightarrow L_1^*) \rightarrow L_2 \rightarrow L_1 \\ L^{21}_\rho(v) & \widehat{=} & v \\ L^{21}_\rho(f\ e) & \widehat{=} & (L^{21}_\rho(f))\,(L^{21}_\rho(e)) \\ L^{21}_\rho(E) & \widehat{=} & E \\ L^{21}_\rho(\lambda\, \overrightarrow{v}, \overrightarrow{q\$} \bullet e) & \widehat{=} & \lambda\, \overrightarrow{v} \frown \overrightarrow{\rho_V(q\$)} \bullet (L^{21}_\rho(e)) \\ L^{21}_\rho(e[\overrightarrow{f}, \overrightarrow{r\$}/\overrightarrow{x}, \overrightarrow{q\$}]) & \widehat{=} & (L^{21}_\rho(e))[\overrightarrow{L^{21}(f)} \frown \overrightarrow{\rho_E(r\$)}/\overrightarrow{x} \frown \overrightarrow{\rho_V(q\$)}] \end{array}$$

Inbuilt here, are assumptions about the lengths of various lists matching up in $[\overrightarrow{f}, \overrightarrow{r\$}/\overrightarrow{x}, \overrightarrow{q\$}]$:

$$\begin{array}{rcl} \mathsf{len}\, \overrightarrow{f} & = & \mathsf{len}\, \overrightarrow{x} \\ \mathsf{len}\, \overrightarrow{r\$} & = & \mathsf{len}\, \overrightarrow{q\$} \\ \mathsf{len}(\rho_E(r\$_i)) & = & \mathsf{len}(\rho_V(q\$_i)) \end{array}$$

We now need to extend the variable set language to cope with the Q extensions (again we have S_2 subsuming one S_1 component, and adding a new one:

$$s \in S_2 \quad ::+ \quad \{\overrightarrow{v}, \overrightarrow{q\$}\} \mid q\$ \in s \Rrightarrow r\$$$

Here we introduce the $\Rrightarrow$ symbol, similar to $\Rrightarrow$, but note that the contents and meaning are different: $q\$$ will denote a list of variables ($\overrightarrow{v}$), and $r\$$ will denote a list of sets ($\overrightarrow{s}$), of the same length. The overall value will be merging all s_i where v_i is in s. Again,the embedding $S^{12} : S_1 \rightarrow S_2$ should be obvious.

Now, the opposite conversion:

$$\begin{array}{rcl}
S^{21} & : & (Q \to V^*) \times (Q \to L_1^*) \to S_2 \to S_1 \\
S^{21}_\rho(\{\overrightarrow{v}, \overrightarrow{q\$}\}) & \widehat{=} & \{\overrightarrow{v} \frown \overrightarrow{\rho_V(q\$)}\} \\
S^{21}_\rho(E) & \widehat{=} & E \\
S^{21}_\rho(s_1 \setminus s_2) & \widehat{=} & S^{21}_\rho(s_1) \setminus S^{21}_\rho(s_2) \\
S^{21}_\rho(\bigcup\{\overrightarrow{s}\}) & \widehat{=} & \bigcup\{\overrightarrow{S^{21}_\rho(s)}\} \\
S^{21}_\rho(v \in s_0 \Rrightarrow s_1) & \widehat{=} & v \in S^{21}_\rho(s_0) \Rrightarrow S^{21}_\rho(s_1) \\
S^{21}_\rho(q\$ \in s \mathrel{\Rrightarrow\!\!\!\!\Rrightarrow} r\$) & \widehat{=} & \bigcup\limits_i\{\rho_V(q\$)_i \in S^{21}_\rho(s) \Rrightarrow \mathsf{fv}_1(\rho_E(r\$)_i)\}
\end{array}$$

We are now in a position to define free variables over L_2:

$$\begin{array}{rcl}
\mathsf{fv}, \mathsf{fv}_2 & : & L_2 \to S_2 \\
\mathsf{fv}(x) & \widehat{=} & \{x\} \\
\mathsf{fv}(e_1\ e_2) & \widehat{=} & \bigcup(\mathsf{fv}\,e_1, \mathsf{fv}\,e_2) \\
\mathsf{fv}(\lambda\ \overrightarrow{x}, \overrightarrow{q\$} \bullet e) & \widehat{=} & (\mathsf{fv}\,e) \setminus \{\overrightarrow{x}, \overrightarrow{q\$}\} \\
\mathsf{fv}(E) & \widehat{=} & E \\
\mathsf{fv}(e[\overrightarrow{f}, \overrightarrow{r\$}/\overrightarrow{x}, \overrightarrow{q\$}]) & \widehat{=} & \mathsf{fv}(e) \setminus \{\overrightarrow{x}, \overrightarrow{q\$}\} \\
& & \cup \bigcup\limits_i\{x_i \in \mathsf{fv}(e) \Rrightarrow \mathsf{fv}(f_i)\} \\
& & \cup \bigcup\limits_j\{q\$_j \in \mathsf{fv}(e) \mathrel{\Rrightarrow\!\!\!\!\Rrightarrow} r\$_j\}
\end{array}$$

Once more, we have a commuting square:

$$\begin{tikzcd}
L_2 \arrow[r, "\mathsf{fv}_2"] \arrow[d, bend left, "L^{21}_\rho"] & S_2 \arrow[d, bend left, "S^{21}_\rho"] \\
L_1 \arrow[r, "\mathsf{fv}_1"'] \arrow[u, bend left, "L^{12}"] & S_1 \arrow[u, bend left, "S^{12}"]
\end{tikzcd}$$

and corresponding theorems:

$$
\begin{aligned}
L^{21}_\rho(L^{12}(e_1)) &= e_1,\ \text{for all } \rho &\quad (2.7)\\
S^{21}_\rho(S^{12}(s_1)) &= s_1,\ \text{for all } \rho &\quad (2.8)\\
\mathsf{fv}_2(L^{12}(e_1)) &= S^{12}(\mathsf{fv}_1(e_1)) &\quad (2.9)\\
S^{21}_\rho(\mathsf{fv}_2(e_2)) &= \mathsf{fv}_1(L^{21}_\rho(e_2)),\ \text{for all } \rho &\quad (2.10)
\end{aligned}
$$

At this point we consider normal forms for free-variable set expressions, which require three further language extensions: we extend membership to include conjunction, and introduce a shorthand for the empty conjunction (true). We also allow explicit subtractions on the rhs of $\Rrightarrow$:

$$
\begin{array}{rcll}
m \in \mathit{Member} & ::= & v \in s \mid q\$ \in s \mid q\$ \in r\$ & \text{Element Membership}\\
& \mid & \bigwedge(m_1, \ldots, m_n) & \text{Conjunction}\\
\top & \mathrel{\widehat=} & \bigwedge() & \\
s \in S_3 & ::+ & m \Rrightarrow r\$ \setminus \{\overrightarrow{v}, \overrightarrow{q\$}\} &
\end{array}
$$

Note that we also allow a membership predicate of the form $q\$ \in s$ to appear in $\Rrightarrow$, with a different semantics to the same thing in $\Rrightarrow$:

$$
S^{21}_\rho(q\$ \in s_1 \Rrightarrow s_2) \quad \mathrel{\widehat=} \quad \bigcup_i \{\rho_V(q\$)_i \in S^{21}_\rho(s_1) \Rrightarrow s_2\}
$$

In addition we allow $q\$ \in r\$$ to appear in $\Rrightarrow$, where we break the need for corresponding indices of $q\$$ and $r\$_1$ to match, but retain it between $q\$_i$ and $(r\$_2)_i$:

$$
S^{21}_\rho(q\$ \in r\$_1 \Rrightarrow r\$_2) \quad \mathrel{\widehat=} \quad \bigcup_i \{\rho_V(q\$)_i \in \mathsf{fv}_1(\rho_E(r\$_1)) \Rrightarrow \mathsf{fv}_1(\rho_E(r\$_2)_i)\}
$$

We also introduce a form of conjunction for membership predicates in $\Rrightarrow$ and the notion of a naked Q variable $r\$$ denoting a list of expressions:

$$
s \in S_3 \quad ::+ \quad m \Rrightarrow s \mid r\$
$$

Embedding $S^{23} : S_2 \to S_3$ is obvious, whilst the conversion in the opposite direction is less so:

$$
\begin{array}{rcl}
S^{32} & : & S_3 \to S_2\\
S^{32}(\bigwedge() \Rrightarrow s) & \mathrel{\widehat=} & s\\
S^{32}(\bigwedge(m_1, \ldots, m_n) \Rrightarrow s) & \mathrel{\widehat=} & m_1 \Rrightarrow (m_2 \Rrightarrow \ldots (m_n \Rrightarrow s) \ldots)\\
S^{32}(r\$) & \mathrel{\widehat=} & q\$ \in \{\}\$q \Rrightarrow r\$
\end{array}
$$

So, we can use fv_2 to get the desired free-variable set for any instance of L_2, as a member of S_2, and then embed into S_3 for normalisation and reasoning.

At this point it is worth writing out S_3 explicitly (as S), and fine-tuning the syntax of membership:

$$\begin{array}{rcl}
s \in S & ::= & \{\overrightarrow{v}, \overrightarrow{q\$}\} \\
& | & E \\
& | & s \setminus s \\
& | & \bigcup\{\overrightarrow{s}\} \\
& | & m \Rrightarrow s \\
& | & q\$ \in s \mathrel{\Rrightarrow\!\!\!\!\Rrightarrow} r\$ \setminus \{\overrightarrow{v}, \overrightarrow{q\$}\} \\
& | & r\$ \\
m \in \mathit{Mmbr} & ::= & v \in s \mid q\$ \in s \mid \bigwedge(m_1, \ldots, m_n)
\end{array}$$

It is worth stressing how the constructs $r\$$, $\Rrightarrow$ and $\mathrel{\Rrightarrow\!\!\!\!\Rrightarrow}$ should be interpreted:

- As a standalone, $r\$$ denotes the union of the free-variables of the list of expressions given by ρ_E.

$$[\![r\$]\!] = \bigcup_i \{\mathsf{fv}(\rho_E(r\$)_i)\}$$

- The construct $(v \mid q\$) \in s \Rrightarrow t$ equals t if all of v or $q\$$ is contained in s. If either s or t is an instance of $r\$$ then it is viewed as per the previous bullet-point.

$$\begin{array}{rcl}
[\![v \in s \Rrightarrow t]\!] & = & [\![t]\!] \lhd v \in [\![s]\!] \rhd \emptyset \\
[\![q\$ \in s \Rrightarrow t]\!] & = & [\![t]\!] \lhd q\$ \subseteq [\![s]\!] \rhd \emptyset
\end{array}$$

- The construct $q\$ \in s \mathrel{\Rrightarrow\!\!\!\!\Rrightarrow} r\$ \setminus \{\overrightarrow{v}, \overrightarrow{q\$}\}$ is well-formed only if $\mathsf{len}\,\rho_V(q\$) = \mathsf{len}\,\rho_E(r\$)$, and basically determines its result on a component-wise examination of the elements of $q\$$.

$$\begin{array}{rcl}
[\![q\$ \in s \mathrel{\Rrightarrow\!\!\!\!\Rrightarrow} r\$ \setminus \{\overrightarrow{v}, \overrightarrow{q\$}\}]\!] & = & \displaystyle\bigcup_{i=1}^{\mathsf{len}\ q\$} \{\mathsf{fv}(\rho_E(r\$)_i) \lhd \rho_V(q\$)_i \in [\![s]\!] \rhd \emptyset\} \setminus \{\overrightarrow{v}, \overrightarrow{q\$}\} \\
& = & \displaystyle\bigcup_{i=1}^{\mathsf{len}\ q\$} \{\rho_V(q\$)_i \in s \Rrightarrow \mathsf{fv}(\rho_E(r\$)_i)\} \setminus \{\overrightarrow{v}, \overrightarrow{q\$}\}
\end{array}$$

  If s is an instance of $r\$$ then we use the interpretation in the first bullet point. The case $v \in s \mathrel{\Rrightarrow\!\!\!\!\Rrightarrow} r\$$ is only valid if $\mathsf{len}\,\rho_E(r\$) = 1$.

In effect $\Rrightarrow$ is a global conditional, while $\mathrel{\Rrightarrow\!\!\!\!\Rrightarrow}$ works pointwise on corresponding members of $q\$$ and $r\$$.

Note that if $q\$$ and $r\$$ have length one, then $q\$ \in s \Rrightarrow r\$$ and $q\$ \in s \mathrel{\Rrightarrow\!\!\!\!\Rrightarrow} r\$$ are the same.

A conditional expression $m \Rrightarrow s$ is *upfront* if s does not itself contain any conditionals, noting that $m \mathrel{\Rrightarrow\!\!\!\Rrightarrow} r\$$ are always up-front by construction. A set-expression is *atomic* if it has one of the following forms (a is v or $q\$$):

$$\{a_1, \ldots, a_n\} \qquad\qquad E \setminus \{a_1, \ldots, a_n\},\ n \geq 0 \qquad\qquad r\$ \setminus \{a_1, \ldots, a_n\},\ n \geq 0$$

We define our normal form to be a union of upfront conditional set-expressions whose set components are atomic. We convert to normal-form by repeatedly applying the following equivalences left-to-right, designed to bring union and conditional to the top

$$\begin{aligned}
m_1 \Rrightarrow (m_2 \Rrightarrow s) &= \bigwedge(m_1, m_2) \Rrightarrow s \\
(m \Rrightarrow s) \setminus \{a_i\} &= m \Rrightarrow (s \setminus \{a_i\}) \\
m \Rrightarrow \bigcup(s_i) &= \bigcup(m \Rrightarrow s_i) \\
m \Rrightarrow \{\} &= \{\} \\
q\$ \in (v \in s_0 \Rrightarrow s_1) \mathrel{\Rrightarrow\!\!\!\Rrightarrow} r\$ &= v \in s_0 \Rrightarrow (q\$ \in s_1 \mathrel{\Rrightarrow\!\!\!\Rrightarrow} r\$) \\
q\$_1 \in (q\$_2 \in s_2 \mathrel{\Rrightarrow\!\!\!\Rrightarrow} r\$_2) \mathrel{\Rrightarrow\!\!\!\Rrightarrow} r\$_1 &= \bigwedge(q\$_1 \in r\$_2, q\$_2 \in s_2) \mathrel{\Rrightarrow\!\!\!\Rrightarrow} r\$_1 \\
\bigwedge(m_i, a \in (m \Rrightarrow s_1)) \Rrightarrow s_2 &= \bigwedge(m_i, a \in s_1, m) \Rrightarrow s_2 \\
\bigwedge(m_i, a \in \{a_i\}, a \in \{a_j\}) \Rrightarrow s_3 &= \bigwedge(m_i, a \in (\{a_i\} \cap \{a_j\}), m) \Rrightarrow s_3 \\
\bigwedge(m_i, a \in \{\}) \Rrightarrow s &= \{\} \\
a \in N \setminus \{a_i\} &= a \in N, \qquad a \notin \{a_i\} \\
&= a \in \{\}, \qquad a \in \{a_i\} \\
\bigwedge(m_i, a \in \bigcup(s_i)) \Rrightarrow s &= \bigcup(\bigwedge(m_i, a \in s_i) \Rrightarrow s) \\
\bigcup(s_i) \setminus \{a_i\} &= \bigcup(s_i \setminus \{a_i\}) \\
(s \setminus \{a_i\}) \setminus \{a_j\} &= s \setminus \{a_i, a_j\} \\
s &= \bigcup(\top \Rrightarrow s)
\end{aligned}$$

We assume that nested unions and conjunctions are flattened on-the-fly.

$((P \equiv Q) \equiv R) \equiv (P \equiv (Q \equiv R))$	⌞Ax-$\equiv$-assoc⌝
$P \equiv Q \equiv Q \equiv P$	⌞Ax-$\equiv$-symm⌝
$\mathit{true} \equiv Q \equiv Q$	⌞Ax-$\equiv$-id⌝
$\mathit{false} \equiv \neg \mathit{true}$	⌞Ax-*false*-def⌝
$\neg(P \equiv Q) \equiv \neg P \equiv Q$	⌞Ax-$\neg$-$\equiv$-distr⌝
$P \vee Q \equiv Q \vee P$	⌞Ax-$\vee$-symm⌝
$(P \vee Q) \vee R \equiv P \vee (Q \vee R)$	⌞Ax-$\vee$-assoc⌝
$P \vee P \equiv P$	⌞Ax-$\vee$-idem⌝
$P \vee (Q \equiv R) \equiv P \vee Q \equiv P \vee R$	⌞Ax-$\vee$-$\equiv$-distr⌝
$P \vee \neg P$	⌞Ax-Excl-Mdl⌝
$P \wedge Q \equiv P \equiv Q \equiv P \vee Q$	⌞Ax-Golden-Rule⌝
$P \Rightarrow Q \equiv P \vee Q \equiv Q$	⌞Ax-$\Rightarrow$-def⌝

Figure 2.11: $U{\cdot}(TP)^2$ Propositional Axioms

2.5 Axioms

We take our inspiration from [Tou01]:

Ax1. All propositional axioms from [GS93].

Ax2. $A \vee (\forall x \bullet B) \equiv (\forall x \bullet A \vee B), x \notin A$

Ax3. $(\forall x \bullet A) \Rightarrow A[x := t]$ (t substitutable in x (in A))

Ax4. $x = x$

Ax5. **(Liebniz)** $x = t \Rightarrow (A \equiv A[x := t])$
(t substitutable in x (in A))

The meaning of $A[x := t]$ in Tourlakis is not given explicitly, but is discussed, particularly with regard to the side-condition designed to avoid variable capture (t substitutable in x (in A)). We shall define $A[x := t]$ as meta-notation describing the substitution of t for all free x in A, with α-renaming being used to avoid name-capture (so the side-conditions on Ax3 and 5 above can be dropped). Our take on the propositional axioms is shown in Figure 2.11. The remaining axioms are shown in Figure 2.12. The substitution axioms (⌞Ax-XXX-Subst⌝) are experimental. The reflexivity axiom, and that for θ are in fact schemas, indexed by all possible types (sorts), as we have a many-sorted logic.

Also worth noting are the β-reduction axioms, e.g.

$$(\lambda\, x, x\$ \bullet e) f = (\lambda, x\$ \bullet e)[f/x]\,.$$

The lhs can match a lambda-expression with only one variable ($(\lambda\, x \bullet e) f$) which results in the rhs being $(\lambda\, ; \bullet e)[f/x]$, which is just $e[f/x]$. So, to maintain consistency, we should be able to match a rhs of the form $e[f/x]$ and then view it as a zero-argument lambda abstraction, and succeed, returning the lhs as $(\lambda\, x \bullet e) f$.

$p \vee (\forall x\$, y\$ \bullet q)$	⌞Ax-$\vee$-$\forall x$-scope⌝
$\equiv (\forall x\$ \bullet p \vee (\forall y\$ \bullet q)), \quad x\$ \notin p$	
$p \vee (\forall E\$, F\$ \bullet q)$	⌞Ax-$\vee$-$\forall E$-scope⌝
$\equiv (\forall E\$ \bullet p \vee (\forall F\$ \bullet q)), \quad E\$ \notin p$	
$p \vee (\forall P\$, Q\$ \bullet q)$	⌞Ax-$\vee$-$\forall P$-scope⌝
$\equiv (\forall P\$ \bullet p \vee (\forall Q\$ \bullet q)), \quad P\$ \notin p$	
$(\forall x\$ \bullet p \wedge q) \equiv (\forall x\$ \bullet p) \wedge (\forall x\$ \bullet q)$	⌞Ax-$\forall x$-distr⌝
$(\forall E\$ \bullet p \wedge q) \equiv (\forall E\$ \bullet p) \wedge (\forall E\$ \bullet q)$	⌞Ax-$\forall E$-distr⌝
$(\forall P\$ \bullet p \wedge q) \equiv (\forall P\$ \bullet p) \wedge (\forall P\$ \bullet q)$	⌞Ax-$\forall P$-distr⌝
$(\forall x, x\$ \bullet p) \Rightarrow (\forall x\$ \bullet p[e/x])$	⌞Ax-$\forall x$-inst⌝
$(\forall E, E\$ \bullet p) \Rightarrow (\forall E\$ \bullet p[e/E])$	⌞Ax-$\forall E$-inst⌝
$(\forall P, P\$ \bullet q) \Rightarrow (\forall P\$ \bullet q[r/P])$	⌞Ax-$\forall P$-inst⌝
$(\exists x\$ \bullet p) \equiv \neg (\forall x\$ \bullet \neg p)$	⌞Ax-$\exists x$-def⌝
$(\exists E\$ \bullet p) \equiv \neg (\forall E\$ \bullet \neg p)$	⌞Ax-$\exists E$-def⌝
$(\exists P\$ \bullet p) \equiv \neg (\forall P\$ \bullet \neg p)$	⌞Ax-$\exists P$-def⌝
$\exists! x\$ \bullet p$	⌞Ax-$\exists! x$-def⌝
$\equiv (\exists x\$ \bullet p) \wedge \exists y\$ \bullet p[y\$/, x\$] \Rightarrow y\$ = x\$$	
$e = e$	⌞Ax-$=$-refl⌝
$(e = \theta x \bullet p)$	⌞Ax-θ-Def⌝
$\equiv p[e/x] \wedge (\forall y \bullet p[y/x] \Rightarrow y = e), \; x \notin e$	
$(\lambda x, x\$ \bullet e)f = (\lambda, x\$ \bullet e)[f/x]$	⌞Ax-β-OReduce⌝
$(\Lambda E, E\$ \bullet q)e \equiv (\Lambda, E\$ \bullet q)[e/E]$	⌞Ax-β-EReduce⌝
$(\Lambda P, P\$ \bullet q)r \equiv (\Lambda, P\$ \bullet q)[r/P]$	⌞Ax-β-PReduce⌝
$(\bigwedge_{i=1}^{n} x_i = e_i) \Rightarrow (p \equiv p[\overrightarrow{e}/\overrightarrow{x}]), \quad x_i$ distinct,	⌞Ax-Leibniz⌝
x_i distinct	
$p[\overrightarrow{e}/\overrightarrow{x}] \equiv p[\overrightarrow{x} := \overrightarrow{e}]$	⌞Ax-OSubst⌝
$p[\overrightarrow{e}/\overrightarrow{E}] \equiv p[\overrightarrow{E} := \overrightarrow{e}]$	⌞Ax-ESubst⌝
$p[\overrightarrow{q}/\overrightarrow{P}] \equiv p[\overrightarrow{P} := \overrightarrow{q}]$	⌞Ax-PSubst⌝
$true[e\$/x\$] \equiv true$	⌞Ax-$true$-OSubst⌝
$true[e\$/E\$] \equiv true$	⌞Ax-$true$-ESubst⌝
$true[q\$/P\$] \equiv true$	⌞Ax-$true$-PSubst⌝
$false[e\$/x\$] \equiv false$	⌞Ax-$false$-OSubst⌝
$false[e\$/E\$] \equiv false$	⌞Ax-$false$-ESubst⌝
$false[q\$/P\$] \equiv false$	⌞Ax-$false$-PSubst⌝

Figure 2.12: $U\cdot(TP)^2$ Non-propositional Axioms

$$\frac{P}{P[Q := R]} \qquad \text{(Substitution)}$$

$$\frac{P \equiv Q}{R[S := P] \equiv R[S := Q]} \qquad \text{(Leibniz)}$$

$$\frac{P, P \equiv Q}{Q} \qquad \text{(Equanimity)}$$

Figure 2.13: $U{\cdot}(TP)^2$ Inference Rules

2.6 Inference

From [Tou01]:

Inf1. (Substitution) (no capture)

Inf2. (Leibniz) $\dfrac{A \equiv B}{C[p := A] \equiv C[p := B]}$

Inf3. (Equanimity) $\dfrac{A, A \equiv B}{B}$

Inf4. (Transitivity) $\dfrac{A \equiv B, B \equiv C}{A \equiv C}$

We note that Inf4 is derivable from Inf2 and Inf3, so we treat it as derivable. Our inference rules are shown in Figure 2.13.

2.7 Proof/Theorems

We adopt the definitions from [Tou01] with minor changes in notation:

The set of Γ-theorems, $\mathbf{Thm}_\Gamma$, is the ⊆-smallest subset of *Pred* that satisfies:

Th1 $\mathbf{Thm}_\Gamma$ contains as subset the closure under (Substitution) of all the propositional axioms (Fig. 2.11) and all the instances of the axiom schemata for the non-propositional part (Fig. 2.12)—the so-called logical axioms.

Th2 $\Gamma \subseteq \mathbf{Thm}_\Gamma$—the non-logical axioms.

Th3 $\mathbf{Thm}_\Gamma$ is closed under (Leibniz) and (Equanimity).

We write $p \in \mathbf{Thm}_\Gamma$ as $\Gamma \vdash p$, and use $\vdash p$ to denote $\emptyset \vdash p$.

A finite sequence $p_1, \ldots, p_n$ of *Pred* is a Γ-proof iff every p_i, for $i = 1, \ldots, n$ is one of

Pr1 A logical axiom

Pr2 A member of Γ

Pr3 The result of either (Leibniz) or (Equanimity) applied to some p_j, with $j < i$.

2.8 Meta-Theorems

We shall need to identify the derived rules that are embodied, either in the matcher or in the provided proof strategies, as these form part of the prover "core" (e.g. the deduction theorem and the `Assume` strategy, or the all the derived variants of $\llcorner$AX-$\vee$-$\forall$-SCOPE$\urcorner$, which are implemented by `completeMatch`, in module `Manipulation`). Some of these rules are presented in Figure 2.14. Post's Tautology Theorem requires more background. Of considerable importance is the Herbrand Deduction Theorem and its variants, shown in Figure 2.15. The Flexible Deduction rule is implemented, as suggested in [Tou01], not by doing the substitution, but by making $\overrightarrow{x}$ behave like constants, by marking them as "known", to use the parlance in module `Matching`.

2.9 Undefinedness

We have yet to make a firm decision how to handle undefinedness, apart from noting that the technical details of how this is done have implications for the validity of the reflexivity of equals, and of the Deduction Theorem, among others.

Type matching can only return type-bindings, so it is simpler than the others.

For the formal presentation we assume the following mathematical type syntax:

$$
\begin{array}{rcl}
\rho, \tau \in \mathit{Type} & ::= & B \text{ — Boolean} \\
& | & Z \text{ — Integer} \\
& | & t \text{ — Type variable} \\
& | & ? \text{ — Arbitrary Type} \\
& | & P\,\tau \text{ — Set Type Constructor} \\
& | & \tau^* \text{ — List Type Constructor} \\
& | & \tau \times \tau \text{ — Product Type Constructor} \\
& | & \tau \to \tau \text{ — Function Type Constructor} \\
& | & !s \text{ — Error Type}
\end{array}
$$

$\dfrac{P \equiv Q, Q \equiv R}{P \equiv R}$	(Transitivity)
$\Gamma \vdash p \quad \text{iff} \quad \Gamma \vdash p \equiv \mathit{True}$	(Truth-is-True)
$\vdash (\forall \overrightarrow{x}, \overrightarrow{q\$} \bullet \mathit{True}) \equiv \mathit{True}$	(Always-True)
$\vdash (\forall E\$ \bullet \mathit{True}) \equiv \mathit{True}$	
$\vdash (\forall P\$ \bullet \mathit{True}) \equiv \mathit{True}$	
$\Gamma \vdash p \quad \text{iff} \quad \Gamma \vdash \forall \overrightarrow{x}, \overrightarrow{q\$} \bullet p$	($\forall$ Intro/Elim)
$\Gamma \vdash p \quad \text{iff} \quad \Gamma \vdash \forall E\$ \bullet p$	
$\Gamma \vdash p \quad \text{iff} \quad \Gamma \vdash \forall P\$ \bullet p$	
$p \vdash \forall \overrightarrow{x}, \overrightarrow{q\$} \bullet p$	(Generalization)
$p \vdash \forall E\$ \bullet p$	$p \vdash \forall P\$ \bullet p$
$p \vdash p[\overrightarrow{x} := \overrightarrow{e}]$	(Corr. Generalization)
$p \vdash p[E\$:= \overrightarrow{e}]$	$p \vdash p[P\$:= \overrightarrow{q\$}]$
$\vdash p \Rightarrow (\forall \overrightarrow{x}, \overrightarrow{y}, \overrightarrow{q\$} \bullet r)$	(Universal Consequent 1)
$\quad \equiv (\forall \overrightarrow{x} \bullet p \Rightarrow \forall \overrightarrow{y}, \overrightarrow{q\$} \bullet r), \quad \overrightarrow{x} \notin p$	
$\vdash p \Rightarrow (\forall E\$, E\$ \bullet r)$	
$\quad \equiv (\forall E\$ \bullet p \Rightarrow \forall E\$ \bullet r), \quad E\$ \notin p$	
$\vdash p \Rightarrow (\forall P\$, Q\$ \bullet r)$	
$\quad \equiv (\forall P\$ \bullet p \Rightarrow \forall Q\$ \bullet r), \quad P\$ \notin p$	
$\vdash p \equiv \forall \overrightarrow{x} \bullet p, \quad \overrightarrow{x} \notin p$	(Corr. Univ. Conseq. 1)
$\vdash p \equiv \forall E\$ \bullet p, \quad E\$ \notin p$	$\vdash p \equiv \forall P\$ \bullet p, \quad P\$ \notin p$
$\Gamma \vdash p \Rightarrow q \quad \text{iff} \quad \Gamma \vdash p \Rightarrow \forall \overrightarrow{x} \bullet q, \quad \overrightarrow{x} \notin p$	(Universal Consequent 2)
$\Gamma \vdash p \Rightarrow q \quad \text{iff} \quad \Gamma \vdash p \Rightarrow \forall E\$ \bullet q, \quad E\$ \notin p$	
$\Gamma \vdash p \Rightarrow q \quad \text{iff} \quad \Gamma \vdash p \Rightarrow \forall P\$ \bullet q, \quad P\$ \notin p$	
$\Gamma \vdash q \Rightarrow p \quad \text{iff} \quad \Gamma \vdash (\exists \overrightarrow{x} \bullet q) \Rightarrow p, \quad \overrightarrow{x} \notin p$	(Corr. Univ.l Conseq. 2)
$\Gamma \vdash q \Rightarrow p \quad \text{iff} \quad \Gamma \vdash (\exists E\$ \bullet q) \Rightarrow p, \quad E\$ \notin p$	
$\Gamma \vdash q \Rightarrow p \quad \text{iff} \quad \Gamma \vdash (\exists P\$ \bullet q) \Rightarrow p, \quad P\$ \notin p$	
$\vdash (\forall \overrightarrow{x} \bullet p) \equiv (\forall \overrightarrow{z} \bullet p[\overrightarrow{x} := \overrightarrow{z}]), \quad \overrightarrow{z}$ fresh	(α-Renaming)
$\vdash (\forall E\$ \bullet p) \equiv (\forall F\$ \bullet p[E\$:= F\$]), \quad F\$$ fresh	
$\vdash (\forall P\$ \bullet p) \equiv (\forall Q\$ \bullet p[P\$:= Q\$]), \quad Q\$$ fresh	

Figure 2.14: $U{\cdot}(\mathit{TP})^2$ Derived Rules

$$
\begin{array}{ll}
\Gamma, p \vdash q \quad \text{then} \quad \Gamma \vdash p \Rightarrow q, \quad p \text{ closed} & \text{(Deduction Theorem)} \\
\Gamma \vdash p \Rightarrow q \quad \text{then} \quad \Gamma, p \vdash q & \text{(Deduc. Converse)} \\
\Gamma, p[\overrightarrow{x} := \overrightarrow{k}] \vdash q[\overrightarrow{x} := \overrightarrow{k}] \quad \text{then} \quad \Gamma \vdash p \Rightarrow q, & \text{(Flexible Deduction)} \\
\quad \overrightarrow{x} = \mathsf{fv}\, p, \overrightarrow{k} \text{ constants.} &
\end{array}
$$

Figure 2.15: $U\cdot(TP)^2$ Deduction Theorem

We can define inference rules for matching:

$$\llcorner\text{TM-}\textsc{Bool}\urcorner \qquad \frac{}{B \ddagger B \mid \theta}$$

$$\llcorner\text{TM-}\textsc{Int}\urcorner \qquad \frac{}{Z \ddagger Z \mid \theta}$$

$$\llcorner\text{TM-}\textsc{Var}\urcorner \qquad \frac{}{t \ddagger \tau \mid \{t \mapsto \tau\}}$$

$$\llcorner\text{TM-}\textsc{Arb}\urcorner \qquad \frac{}{? \ddagger \tau \mid \theta}$$

$$\llcorner\text{TM-}\textsc{Set}\urcorner \qquad \frac{\rho \ddagger \tau \mid \beta}{\mathcal{P}\rho \ddagger \mathcal{P}\tau \mid \beta}$$

$$\llcorner\text{TM-}\textsc{List}\urcorner \qquad \frac{\rho \ddagger \tau \mid \beta}{\rho^* \ddagger \tau^* \mid \beta}$$

$$\llcorner\text{TM-}\textsc{Prod}\urcorner \qquad \frac{\rho_1 \ddagger \tau_1 \mid \beta_1 \qquad \rho_2 \ddagger \tau_2 \mid \beta_2}{\rho_1 \times \rho_2 \ddagger \tau_1 \times \tau_2 \mid \beta_1 \uplus \beta_2}$$

$$\llcorner\text{TM-}\textsc{Fun}\urcorner \qquad \frac{\rho_1 \ddagger \tau_1 \mid \beta_1 \qquad \rho_2 \ddagger \tau_2 \mid \beta_2}{\rho_1 \rightarrow \rho_2 \ddagger \tau_1 \rightarrow \tau_2 \mid \beta_1 \uplus \beta_2}$$

The following rules are controversial (a form of reverse matching):

$$\llcorner\text{TM-}\textsc{Var}\text{R}\urcorner \qquad \frac{}{\rho \ddagger t \mid \{t \mapsto \rho\}}$$

$$\llcorner\text{TM-}\textsc{Arb}\text{R}\urcorner \qquad \frac{}{\rho \ddagger ? \mid \theta}$$

The rules are intended to check law matches for type-compatibility.

Chapter 3

Conventions

Chapter 4

Proof and Model Theory

I guess this'll have to be done at some point—sigh!

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.