Further Developing T37

Roger Bishop Jones

ICL Defence Systems

1. INTRODUCTION

This document further develops the theory T37, specified in document DTC/RBJ/037. In particular it develops within the essentially type free framework of T37, an expressive type system. The specification is conducted in a readable semi-formal notation, backed up by a fully formal executable specification in Miranda. The semi-formal notation is marked by a vertical bar in the left margin. The formal notation is in the language Miranda and occurs between marks | |< and | |> in the text.

The specification of the formal system T37 is loaded directly from DTC/RBJ/037 to which the reader should refer. Further definitions are also loaded from DTC/RBJ/040.

```
||<
%insert "037"
%nolist
||>
```

We use the notation [x]u as an abbreviation defined informally as follows:

```
[x]x = I
[x]u = K u (if x does not occur in u)
[x](u v) = S ([x]u) ([x]v)
```

We will also abbreviate multiple abstractions thus:

[x,y...]u = [x][y]... u

The miranda version of [x]u is the function *abst*.

```
||<
abst x (V y) = i, x=y
abst x (V y) = k $A (V y), ~(x=y)
abst x (C y) = k $A (C y)
abst x (t $A u) = (s $A (abst x t)) $A (abst x u)
||>
```

The Miranda function providing multiple abstraction on a list of variables is *abstl*.

```
||<
abstl [] t = t
abstl (x:y) t = abst x (abstl y t)
||>
```

2. DERIVING CURRY'S PARADOX

In this section we try hard to reconstruct Curry's Paradox in T37.

Curry's paradox cannot be constructed using our version of implication since the deduction theorem does not hold for it. However, using Ξ an alternative definition of implication can be given for which the deduction theorem probably does hold.

(dci) ⊃ = [x,y] Ξ (K x)(K y) (curry implication)

```
||<
dci = def "⊃" (abstl ["x","y"] (cl "Ξ" [ca "K" vx, ca "K" vy]))
||>
```

3. LOGICAL CONNECTIVES

(dneg)	¬	= [x] (x = False)	(negation over Bool)
(dbool)	Bool	= [x] x ¬ x	(the type of truth values)
(dand)	∧	= [x,y] ¬ (¬ x ¬ y)	(and)
(dimp)	☞	= [x,y] ¬ x y	(implication)
(dequiv)	⇔	= [x,y] (x ☞ y) ∧ (y ☞ x)	(equivalence)

```
||<
dneg  = def "¬"    (abst "x" (cl "="[vx, false]))
dbool = def "Bool"         (abst "x" (cl ""[vx, ca "¬" vx]))
dand  = def "∧"    (abstl ["x","y"] (ca "¬" ((ca "¬" vx) $A (ca "¬" vy))))
dimp  = def "☞"    (abstl ["x","y"] (cl ""[ca "¬" vx, vy]))
deqiv = def "⇔"    (abstl ["x","y"] (cl "∧"[cl "☞"[vx, vy], cl "☞"[vy, vx]]))
||>
```

4. QUANTIFIERS

4.1. The Universe and the Empty Type

(duniv)	U	= K True	(the Universe)
(dempty)	∅	= K False	(the empty type)

```
||<
duniv = def "U" (k $A true)
dempty       = def "∅" (k $A false)
||>
```

4.2. Universal Quantification

(dpi)	Π	= [x] Ξ U x	(unrestricted generality)
(duq)	∀	= Π	(universal quantifier)

```
||<
dpi    = def "Π" (abst "x" (cl "Ξ"[C "U", vx]))
duq    = def "∀" (C "Π")
||>
```

4.3. Existential Quantification

(dexists)	∃	= [x]¬ ∀ [y]¬(x y)	(existential quantifier)

```
||<
dexists = def "∃" (abst "x" (ca "¬" (ca "∀" (abst "y" (ca "¬" (vx $A vy))))))
||>
```

5. DOMAIN CONSTRUCTORS

We now define a selection of domain constructors. A domain is essentially a partial characteristic function which determines two disjoint recursively enumerable sets of combinatory terms. Domains are one component of types, the other is an equivalence relation representing equality over the type.

```
(df)      →      = [x,y,z] Ξ x ([x2] y (z x2))          (function domain constructor)
(dg)             = [x,y,z] Ξ x ([x2] (y x2)(z x2))      (dependent function domain)
(dpair)   pair   = [x,y,z]z x y                         (ordered pair constructor)
(dfst)    fst    = [x]x true                            (left projection)
(dsnd)    snd    = [x]x false                           (right projection)
(dcp)     ×      = [x,y,z](x (fst z)) ∧ (y (snd z))     (cartesian product domain)
(ddp)            = [x,y,z](x (fst z)) ∧ (y (fst z) (snd z)) (dependent product domain)

(rxe)            = [A,x,y]ΞA([z]x z = y z)              (restricted extensional equality)
```

```
||<

df    = def "→" (abstl ["x","y","z"]
        (cl "Ξ" [vx, (abst "x2" (vy $A (vz $A (V "x2"))))]))

dg    = def "" (abstl ["x","y","z"] (cl "Ξ" [vx,
        (abst "x2" ((vy $A (V "x2")) $A (vz $A (V "x2"))))]))

dpair = def "pair" (abstl ["x","y","z"]((vz $A vy) $A vz))

dfst  = def "fst" (abst "x" (vx $A true))

dsnd  = def "snd" (abst "x" (vx $A false))

dcp   = def "×" (abstl ["x","y","z"]
        (cl "∧" [vx $A (ca "fst" vz), vy $A (ca "snd" vz)]))

ddp   = def "" (abstl ["x","y","z"]
        (cl "∧" [vx $A (ca "fst" vz),
        al [vy, ca "fst" vz, ca "snd" vz]]))
%list
||>
```

6. TYPE CONSTRUCTORS

A type consists of a pair. The first element of the pair a domain, the second an equivalence relation over the domain. Objects of the type are equivalence classes of elements in the domain.

(fst) ☞> = [A,B,f]Ξ(A×A)[x](Ξ(snd A x) (snd B (pair (f (fst x)) (f (snd x)))) (function type constructor)
(cpt) ×> = [A,B,x] (A (fst x)) ∧ (B (snd x))) (cartesian product type constructor)

7. OBSERVATIONS

Curry's paradox is not obviously derivable here because the "logical connectives" only work properly with values of type "bool".

Thus "u ☞ u" should not be provable, only "Ξ bool [u] u ☞ u". All the classical tautologies should be provable within the type "bool", but over U in general only a few tautologies involving explicit use of "true" and/or "false" should be provable. Modus Ponens is derivable, but we have at best a weakened form of the "deduction theorem".

Alternative definitions of the logical connectives can be constructed using Ξ rather than . These would have permitted the construction of curry's paradox had made equality extensional.

Russell's paradox should fail because the law of the excluded middle only holds for values of type "bool".

Previous versions up to (at least) Issue 1.3 of this document were inconsistent by virtue of axiom (qx). This particular problem could have been fixed by replacing the axiom with a similar rule, but further reflection on the match between this formal system and the intended model has persuaded me to make drastic reductions in the strength of equality. This makes the formal system much more likely to be provably consistent, but I may have overdone it and made equality too weak.