1. INTRODUCTION

This document is an attempt to prove the consistency of the formal system specified in DTC/RBJ/037, an axiomatisation of the Theory of Restricted Generality. The proof involves extensive use of the functional programming language Miranda, in which the specification of T37 was expressed. Sections of the document written in Miranda are enclosed between marks ||< and ||> in the text.

We first load the definition of the formal system (for which the reader should refer to DTC/RBJ/037), and extensions (defined in DTC/RBJ/039).

||<
%nolist
%insert "039"
||>

2. THE PRIMITIVE SYSTEM

||<
%insert "sets"
||>

2.1. Abstract Syntax

||<
pterm ::= S | K | pterm $Apterm
||>

2.2. Auxiliary Definitions

We define a proforma and the operation st of substituting a pterm into a proforma to facilitate precise definition of the inference rules.
The following abbreviations improve the readability of the formal specification.

\[
\begin{align*}
\text{pk } u \ v & = (K \ Ap u) \ Ap v \\
\text{ps } u \ v \ w & = ((S \ Ap u) \ Ap v) \ Ap w
\end{align*}
\]

2.3. Axioms and Inference Rules

2.3.1. signature

\[
\begin{align*}
\text{abstype } & \text{ ptheorem} \\
\text{with } \text{ pkaxiom :: ptheorem} & \\
\text{pkrule :: (ptheorem,proforma,pterm,pterm)} & \to \text{ ptheorem} \\
\text{srule :: (ptheorem,proforma,pterm,pterm,pterm)} & \to \text{ ptheorem}
\end{align*}
\]

\[
\text{ptheorem} \ = \ \text{pterm}
\]

2.3.2. axiom

\[
\begin{align*}
\text{pkaxiom } & = K
\end{align*}
\]

2.3.3. inference rules

\[
\begin{align*}
\text{pkrule (th,p,u,v)} & = \text{st } p (\text{pk } u \ v), \text{th} = \text{st } p \ u \\
\text{pkrule s} & = \text{pkaxiom} \\
\text{srule (th,p,u,v,w)} & = \text{st } p (\text{ps } u \ v \ w), \text{th} = \text{st } p ((u \ Ap w) \ Ap (v \ Ap w)) \\
\text{srule s} & = \text{pkaxiom}
\end{align*}
\]
2.4. The Set of Theorems

||<
pterms = us [ [K],
[S],
{ t $Ap u | t,u <- pterms } ]

proformas = us [ [M],
[T t | t <- pterms],
{ p $Pap q | p,q <- proformas } ]

ptheorems = us [ [pk axiom],
{ pkrule (t,p,u,v) | t <- ptheorems;
p,v <- proformas },
{ srule (t,p,u,v,w) | t <- ptheorems;
p,w <- proformas;
u,v,w <- pterms } ] ||>

3. ???

The extended primitive system includes one further primitive, and further axioms and inference rules to support this primitive. We wish to add to the primitive system provision for the use of derived inference rules. This requires an additional rule which enables the result of a derived inference rule to be accepted as a ptheorem.

The definition of the third inference rule depends upon the following function and pterm definitions:

red a function which reduces a pterm to head normal form (if if has one).
enc a function which encodes pterms into (normal) pterms.
dec the inverse of enc
dec_pterm
 a pterm which represents the dec function.

||<
red :: pterm -> pterm
enc :: pterm -> pterm
dec :: pterm -> pterm
dec_pterm :: pterm ||>

4. FUNCTIONS IN THE METANOTATION

In this section we define some useful Miranda functions. These consist of the functions red, enc and dec, mentioned in the previous section. We also define in Miranda functions corresponding to the pterms function_space_pterm and enc_type_of_enc_theo. These latter are provided to
make more transparent the definition of the \textit{pterms} in the following section (and to act as a guide in the construction of these \textit{pterms}).

To facilitate definitions of \textit{pterms} in the object language we make provision for the use of functional abstraction, and define a fix point operator to permit recursive definitions. Since the object language itself contains no variables we use a slightly richer language including variables, and define a mapping from abstractions in the latter language into the \textit{pterms} of our primitive system.

\section*{4.1. Defining Abstraction}

\subsection*{4.1.1. abstract syntax
\begin{verbatim}
var_pterm ::= Vp [char] | Kv | Sv | var_pterm $Vpap var_pterm
\end{verbatim}

\subsection*{4.1.2. identity \textit{pterm}
\begin{verbatim}
ip = (Sv $Vpap Kv) $Vpap Kv
\end{verbatim}

\subsection*{4.1.3. abstraction
\begin{verbatim}
vabst :: [char] -> var_pterm -> var_pterm
vabst v (Vp w) = ip, v=w
vabst v (t $Vpap u) = (Sv $Vpap (vabst v t)) $Vpap (vabst v u)
vabst v u = Kv $Vpap u
\end{verbatim}

\subsection*{4.1.4. multiple abstraction
\begin{verbatim}
vabsl :: [[char]] -> var_pterm -> var_pterm
vabsl [] vt = vt
vabsl (a:b) vt = vabst a (vabsl b vt)
\end{verbatim}

\subsection*{4.1.5. closure}
We define a predicate "closed" which is satisfied by "var_pterm"s which contain no variables.
Closed (Vp n) = False
Closed Kv = True
Closed Sv = True
Closed (t $Vpap u) = (closed t) & (closed u)

and a mapping from closed var_pterms to pterms:

\[
\begin{align*}
\text{v_to_t} &:: \text{var_pterm} \rightarrow \text{pterm} \\
\text{v_to_t} \text{Kv} & = K \\
\text{v_to_t} \text{Sv} & = S \\
\text{v_to_t} (t $Vpap u) & = (\text{v_to_t} t) $Ap (\text{v_to_t} u)
\end{align*}
\]

4.1.6. Recursion

To enable recursive definitions of pterms representing functions we define a fix point operator.

\[
\begin{align*}
\text{vsap} & = (\text{vabsl }["x"] ((\text{Vp }"x") $Vpap (\text{Vp }"x" ))) \\
\text{vfix} & = \text{vabsl }["f"] (\text{vsap} $Vpap (\text{vabsl }["x"] ((\text{Vp }"f") $Vpap ((\text{Vp }"x") $Vpap (\text{Vp }"x" ))))) \\
\text{fix} & = \text{v_to_t} \text{vfix} \\
\text{defrec vars expr} & = \text{v_to_t} (\text{vfix} $Vpap (\text{vabsl vars expr}))
\end{align*}
\]

4.2. Reduction

\[
\begin{align*}
\text{red K} & = K \\
\text{red S} & = S \\
\text{red ((K $Ap u) $Ap v)} & = \text{red u} \\
\text{red (t $Ap u)} & = \text{red (red t $Ap u),"((red t) = t)"} \\
\text{red (t $Ap u)} & = \text{red t $Ap u}
\end{align*}
\]

4.3. Encoding

To define encodings of pterms we define first the two pterms "ptrue" and "pfalse", and then a pair constructor. "ptrue" and "pfalse" are both in normal form, as is any element of the free algebra generated from them by the pair constructor. Furthermore, "ptrue" and "pfalse" are distinguishable, and pairs may be dismembered, so we may construct with these elements an encoding and a
decoding of the pterms of our primitive system.

\[
\begin{align*}
\text{pttrue, pfalse, pair} & : \text{pterm} \\
\text{vtrue} &= \text{Kv} \\
\text{vfalse} &= (\text{vabs} \ ["x","y"] (\text{Vp} "y")) \\
\text{vpair} &= (\text{vabs} \ ["x","y","z"] (((\text{Vp} "z") $\text{Vpap} (\text{Vp} "x")) $\text{Vpap} (\text{Vp} "y"))) \\
\text{pttrue} &= \text{red} (\text{v_to_t vtrue}) \\
\text{pfalse} &= \text{red} (\text{v_to_t vfalse}) \\
\text{pair} &= \text{red} (\text{v_to_t vpair})
\end{align*}
\]

It is convenient to introduce the abbreviations:

\[
\begin{align*}
\text{pearuv} &= \text{red} ((\text{pair} $\text{Ap} \ u) $\text{Ap} \ v) \\
\text{first} \ u &= u $\text{Ap} \ ptrue \\
\text{secnd} \ u &= u $\text{Ap} \ pfalse \\
\text{cond} \ t \ u \ v &= (t $\text{Ap} \ u) $\text{Ap} \ v
\end{align*}
\]

The algorithm for encoding may now be specified:

\[
\begin{align*}
\text{enc K} &= \text{pear ptrue ptrue} \\
\text{enc S} &= \text{pear ptrue pfalse} \\
\text{enc} \ (t $\text{Ap} \ u) &= \text{pear pfalse} (\text{pear} (\text{enc} t) (\text{enc} u))
\end{align*}
\]

4.4. Decoding

We define \text{dec}, the inverse of \text{enc}. Informally the algorithm is:

\[
\begin{align*}
\text{dec} \ (\text{pear ptrue ptrue}) &= K \\
\text{dec} \ (\text{pear ptrue pfalse}) &= S \\
\text{dec} \ (\text{pear pfalse} (\text{pear} A \ B)) &= (\text{dec} A) $\text{Ap} (\text{dec} B)
\end{align*}
\]

However, this is not legal Miranda, and we would like to define \text{dec} so that it will work even when the encoding is not in normal form. We therefore use a definition which does not rely on pattern matching. Note that "\text{dec}" is a total function over encodings, but not over pterms in general, and that it is only weakly extensional.
5. TERMS IN THE OBJECT LANGUAGE

In this section we complete the formal definition of the extended primitive system by defining the pterms in the object language \( \text{dec\_pterm} \), \( \text{function\_space\_pterm} \) and \( \text{enc\_type\_of\_enc\_theo} \).

5.1. Decoding

We translate the definition of \( \text{dec} \) to give a pterm \( \text{dec\_pterm} \). First we supply an informal recursive definition with sugared conditional:

\[
\text{dect } x = \begin{cases} 
  \text{K, red } x = \text{enc } K \\
  \text{S, red } x = \text{enc } S \\
  (\text{dec } a) \text{ Ap } (\text{dec } b) 
  \end{cases}
\]

where
\[
\begin{align*}
  a &= \text{first } (\text{secnd } x) \\
  b &= \text{secnd } (\text{secnd } x)
\end{align*}
\]

And now we translate the definition of \( \text{dec} \) using the fixpoint operator previously defined.

\[
\text{dec\_pterm} = \text{defrec } [\"\text{dec}\",\"x\"]
\]

\[
\begin{align*}
  &\text{defrec } [\"\text{dec}\",\"x\"] \\
  &\text{defrec } [\"\text{dec}\",\"x\"] \\
  &\text{defrec } [\"\text{dec}\",\"x\"] \\
  &\text{defrec } [\"\text{dec}\",\"x\"] \\
  &\text{defrec } [\"\text{dec}\",\"x\"] \\
  &\text{defrec } [\"\text{dec}\",\"x\"]
\end{align*}
\]

We note that \( \text{dec\_pterm} \), in consequence of its recursive definition, has no normal form.

\[
\begin{align*}
  \text{tp } t &= (\text{t,dec } (\text{enc } t)) \\
  \text{tpl} &= \text{map tp pterms}
\end{align*}
\]
5.2. The Type of Theorems

||<
enc_type_of_enc_theo = enc dec_pterm
||>

||<
%list
||>