A New Axiomatisation of the Theory of Restricted Generality

Roger Bishop Jones

ICL Defence Systems

1. INTRODUCTION

This document specifies semi-formally and formally a version of illiative combinatory logic. 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.

2. SYNTAX

A term is a constant or a variable or a term applied to a term. Application is rendered by juxtaposition (left associative) except in some cases where an infix notation is used.

The primitive constants are:

K	(λx,y.x)
S	(λx,y,z.xz(yz))
=	equality (infix)
P	Purity
	logical OR (infix)
Ξ	restricted quantification
ι	choice function

A sequent is a list of terms followed by "" followed by a list of terms and should be read "if each term on the left is true then so is each term on the right".

The syntax expressed as a context free grammar is therefore:

term ::= constant | variable | term term | term infix_constant term

variable ::= x | y | z | x1 ...

constant ::= infix_constant | S | K | P | Ξ | ι ...

infix_constant ::= | = | ...

sequent ::= list_of_terms list_of_terms

list_of_terms ::= term | term "," list_of_terms

```
||<
%nolist
term ::=      V [char]      |       || variable
              C [char]      |       || constant
              term $A term        || application

seq ::= [term] $Sq [term]
||>
```

The inference system presented below is a sequent calculus which defines a set of sequents which are called theorems. This is presented as a number of axiom schemas, and three inference rule schemas.

3. AXIOM SCHEMAS

x, x2, y, z are metavariables ranging over variables.

u, u', v, v', w are metavariables ranging over terms.

Φ, Γ, Θ, Δ are metavariables ranging over lists of terms.

Φ ⊆ Γ should be read "every term in Φ is also in Γ".

The following specific abbreviation will be used:

```
true = K
false = KI
I = S K K
```

In Miranda:

```
||<
k = C "K"
s = C "S"
i = (s $A k) $A k
true = k
false = k $A i
p = C "P"
true = C "true"
false = C "false"
||>
```

The semi-formal description of the axiom schemas is:

(inc)	Γ	Θ	if $\Theta \subseteq \Gamma$
(pin)		P K, P S, P true, P false	
(pa)	P u, P v	P (u v)	
(ke)		K u v = u	
(se)		S u v w = (u w) (v w)	

(qi)		u = u
(qs)	u = v	v = u
(qtr)	u = v, v = w	u = w
(qap)	P (u v), u=u', v=v'	u v = u' v'
(qt)	u, u = v	v
(qk)	u	u = true

(oil)	u		u v
(oir)	v		u v
(re)	Ξ u v, u w		v w
(ni)		$\neg$ (S = K)	
(ch)	u v		u (ι u)

In Miranda we first define the abstract data type theorem.

The first set of constructor types are the types of the axiom schemas:

```
||<
abstype theorem
with        inc    :: [term] -> [term] -> theorem
            pin  :: theorem
            pa     :: term -> term -> theorem
            ke     :: term -> term -> theorem
            se     :: term -> term -> term -> theorem
||>

||<
            qi     :: term -> theorem
            qs     :: term -> term -> theorem
            qtr    :: term -> term -> term -> theorem
            qap    :: term -> term -> term -> term -> theorem
            qt     :: term -> term -> theorem
            qk     :: term -> theorem
||>

||<
            oil    :: term -> term -> theorem
            oir    :: term -> term -> theorem
            re     :: term -> term -> term -> theorem
            ni     :: theorem
            ch     :: term -> term -> theorem
||>
```

Types of Inference rules:

```
||<
            tr     :: theorem -> theorem -> theorem
            su     :: theorem -> theorem -> theorem
            oe     :: theorem -> theorem -> theorem
            ri     :: theorem -> theorem
            ra     :: theorem -> theorem
||>
```

Type of Definitions:

```
||<
            def    :: [char] -> term -> theorem
theorem     == seq
||>
```

We now introduce some functions in Miranda which will help make the specifications of the axioms and inference rules more transparent.

al constructs a term from a list of terms by left associative application.

```
||<
al (u:v:w) = al((u $A v):w)
al [u] = u
||>
```

ca applies a constant to a term, *ci* applies an infix constant occurring as its second argument to the terms supplied as its first and third. *cl* applies a constant to a list of terms.

```
||<
ca c u = (C c) $A u
ci u c v = ((C c) $A u) $A v
cl c l = al ((C c):l)
||>
```

```
||<
vx = V "x"
vy = V "y"
vz = V "z"
||>
```

Next we give the Miranda definitions of all the functions which may construct theorems from terms, these correspond to axiom schemas in the semiformal definition.

```
||<
inc tl1 tl2    = tl1                $Sq tl2, (tl2--tl1)=[]
               = ki
pin            = []                 $Sq [p $A k, p $A s, p $a true, p $A false]
pa u v= [p $A u, p$A v] $Sq [p $A (u $A v)]
ke u v= []                  $Sq [cl "="[al [k, u, v], u]]
se u v w       = []                 $Sq [ci (al [s, u, v, w]) "=" (al[u $A w, v $A w])]
||>
```

```
||<
qi u           = []                 $Sq [cl "="[u, u]]
qs u v= [ci u "=" v]        $Sq [ci v "=" u]
qtr u v w      = [ci u "=" v, ci v "=" w]
                                    $Sq [ci u "=" w]
qap u1 u2 v1 v2    = [p $A (u $A v), ci u1 "=" u2, ci v1 "=" v2]
                                    $Sq [ci (u1 $A v1) "=" (u2 $A v2)]
qt u v         = [u, ci u "=" v]            $Sq [v]
qk u           = [u]                $Sq [ci u "=" true, ci (ca "¬" u) "=" false]
||>
```

```
||<
oil u v                    = [u]        $Sq [cl ""[u, v]]
oir u v                    = [v]        $Sq [cl ""[u, v]]
re u v w           = [cl "Ξ"[u, v], u $A w]
                                $Sq [v $A w]
ni                 = []    $Sq [ca "¬" (ci s "=" k)]
ch u v       = [u $A v] $Sq [u $A (C "ι" $A u)]
||>
```

4. INFERENCE RULE SCHEMAS

4.1. Free Variables

We write "fv(c)" to denote the (free) variables occuring in some syntactic construct. A variable "x" occurs in an atomic term iff the variable is the term. A variable "x" occurs in a term "(u v)" iff the variable occurs in u or in v. A variable "x" occurs in a list of terms "Φ" iff it occurs in one or more of the constituent terms.

```
||<
fv_t (V x)   = [x]
fv_t (C x)   = []
fv_t (u $A v)       = (fv_t u) ++ (fv_t v)

fv_lt []       = []
fv_lt (u:f)    = (fv_t u) ++ (fv_lt f)
||>
```

4.2. Transitivity of Sequents

```
Φ Γ; Γ Θ
_____________
Φ Θ
```

```
||<
tr (a $Sq b) (b $Sq c)     = (a $Sq c)
tr u v         = ki
||>
```

4.3. Union of Sequents

$$\frac{\Phi \ \Gamma;\ \Theta \ \Delta}{\Phi \cup \Theta \ \ \Gamma \cup \Delta}$$

```
||<
su (a $Sq b) (c $Sq d) = (a ++ c) $Sq (b ++ d)
||>
```

4.4. Or Elimination

$$\frac{u, \Phi \ \ w;\ v, \Gamma \ \ w}{u \ \ v, \Phi, \Gamma \ \ w}$$

```
||<
oe ((u:f) $Sq [w]) ((v:g) $Sq [w]) = (((cl ""[u,v]):f++g) $Sq [w])
||>
```

4.5. Ξ Introduction

$$\frac{u\ x, \Phi \qquad v\ x \qquad x \notin (fv\ (\Phi) \cup fv(u))}{\Phi \ \ \Xi\ u\ v}$$

```
||<
ri (((u $A (V x)):f) $Sq [v $A (V x)])= f $Sq [cl "Ξ"[u, v]], ~ (member (fv_lt (u:f)) x)
ri u                    = ki
||>
```

4.6. Reductio ad Absurdum

$$\frac{u, \Phi \quad \text{false}}{\Phi \ \ \neg\ u}$$

```
||<
ra ((u:f) $Sq [false])      = f $Sq [ca "¬" u]
ra u                    = ki
||>
```

5. DEFINITIONS AND DERIVED RULES

New constants may be introduced by defining them to be equal to some term formed using only primitive or previously defined constants. The necessary theorem expressing the equation may be obtained using the axiom schema "def" shown below.

The reader should be aware that "def" represents a hole in the type checking of proofs since it does not check either that the constant thus defined has not previously been defined. Nor does it check that the constants used in the definiendum have all been defined.

```
(def)        name = term      || where name is a new constant
                              || and term mentions only previously
                              || defined constants
```

```
||<
def c u = [] $Sq [cl "="[C c, u]]
%list
||>
```