The Calculus of Constructions in Miranda

Roger Bishop Jones
ICL Defence Systems

ABSTRACT

This paper consists of a translation into Miranda of the definition of Coquand’s Calculus of Constructions.

1. INTRODUCTION

The following specifies in Miranda the formal system which is given by Coquand and Huet in INRIA Rapport de Recherche No. 503.

2. The Abstract Syntax of Terms

```
term ::= Type | Prop | Id [char] | In num || de Bruijn index
term || App | Lambda term || Abstraction
Pi term term || Product
```

We now define some operations over lists (or sets).

The union of two sets:

```
[] $union a = a
(a:b) $union c = a:d where d = b, c=[]
    = (hd c):(b $union (tl c)), otherwise
```

The union of a set of sets:

```
us [] = []
us (a:b) = a $union (us b)
```

The length of a list (or cardinality of a finite set)
len [] = 0
len (a:b) = 1 + (len b)

assignment == [term,term]

We now define the set of valid assignments.

abstype valid_assignment
with empty_assignment :: valid_assignment
valid_assignment == assignment

terms = objects $union contexts
objects = us (map o [0..])
contexts= us (map c [0..])

rap n f a applies f to a n times
rap 0 f a = a
rap n f a = f (rap (n-1) f a)

sel selects a term from a list of terms
sel 1 (a:b) = a
sel n (a:b) = sel (n-1) b, n>1

l n selects term n from a context and adds n to its indices
t $ln = rap n ee (sel n t)

m $sub (I 1) = m
m $sub S = S
m $sub (n $L o) = (m $sub n) $L (m $sub o)
m $sub (n $P o) = (m $sub n) $P (m $sub o)
m $sub (n $A o) = (m $sub n) $A (m $sub o)
3. The Inference System of Constructions

sequent ::= [term] $Se rhs

rhs ::= Co term | || context
term $Ta term | || type assignment
term $Eq term | || equation (conversion)

abstype theorem
with star :: theorem
  assu,proi,absi,refl,symm :: theorem -> theorem
  vari :: num -> theorem -> theorem
  appi,tran,prco,beta,tyco :: theorem -> theorem -> theorem
  abco,apco :: theorem -> theorem -> theorem -> theorem

theorem == sequent

star = [] $Se (Co S)

|| assumption

assu (t $Se (Co u)) = (u:t) $Se (Co S)
asusu (t $Se (u $Ta S)) = (u:t) $Se (Co S)
assu s = star

|| product introduction

proi ((mm:t) $Se (Co u)) = t $Se (Co (mm $P u))
proi ((m1:t) $Se (m2 $Ta S)) = t $Se ((m1 $P m2) $Ta S)

|| variable introduction

vari k (t $Se (Co S)) = t $Se ((I k) $Ta (t $I k)), k < (len t)
vari k s = star

|| abstraction introduction

absi ((m1:t) $Se (m2 $Ta pp)) = t $Se ((m1 $L m2) $Ta (m1 $P pp))
absi s = star

|| application introduction

appi (z $Se (mm $Ta (pp $P qq))) (z $Se (nn $Ta pp))
  = z $Se ((mm $A nn) $Ta (nn $sub qq))
appi t u = star
4. The Conversion Rules

|| reflexivity
refl (t $Se (Co d)) = (t $Se (d $Eq d))
refl t = star

|| symmetry
symm (t $Se (mm $Ta d)) = (t $Se (mm $Eq mm))
symm t = star

|| transitivity
tran (z $Se (p1 $Eq p2)) (z $Se (p2 $Eq p3)) = (z $Se (p1 $Eq p3))
tran t u = star

|| product conversion
prco (z $Se (p1 $Eq p2)) ((p1:z) $Se (m1 $Eq m2))
prco t u = star

|| abstraction conversion
abco (z $Se (p1 $Eq p2)) ((p1:z) $Se (m1 $Eq m2)) ((p1:z) $Se (m1 $Ta n1))
abco t u v = star

|| application conversion
apco (z $Se ((mm $A nn) $Ta pp)) (z $Se (mm $Eq m1)) (z $Se (nn $Eq n1))
apco t u v = star

|| beta conversion
beta ((aa:z) $Se (mm $Ta pp)) (z $Se (nn $Ta aa))
beta t u = star

|| type conversion
tyco (z $Se (mm $Ta pp)) (z $Se (pp $Eq qq))
tyco t u = star
tyco t u = star