Illative Combinatory Logic

Roger Bishop Jones

Abstract

Another approach to illative combinatory logic, based this time on the hol4 example on pure combinatory logic. Mainly an attempt to understand why that example is so much simpler than my own efforts.

Created 2012/12/18

Last Change Date: 2013/01/10 16:35:58

Id: t050.doc,v 1.3 2013/01/10 16:35:58 rbj Exp

http://www.rbjones.com/rbjpub/pp/doc/t050.pdf

Contents

1 Prelude

Every now and then I wonder why the things I try to do with ProofPower are so tortuously lengthy and slow. Too rarely I look at what is happening elsewhere and wonder whether I can learn anything useful from them.

This is part of such an enterprise.

I am engaged in work on an infinitary illative combinatory logic, but a large proportion of what I want to do with it does not depend on its infinitary nature but could be done with a finitary illative combinatory logic. So it might be worth progressing without the infinitary aspect, to test out the ideas which can be progressed with finite combinators.

In hol4 there is a very simple example of a Church Rosser proof for the combinatory logic which I imagine would be a good starting point for such an enterprise. I am therefore looking at this material both in ProofPower and in HOL4, trying to replicate the HOL4 proof in ProofPower and to extend both proofs to cover illative combinatory logics.

Discussion of what might become of this document in the future may be found the postscript (Section 5).

In this document, phrases in coloured text are hyperlinks, like on a web page, which will usually get you to another part of this document (the blue parts, the contents list, page numbers in the Index) but sometimes take you (the red bits) somewhere altogether different (if you happen to be online), e.g.: the online copy of this document.

[1]

2 Introduction

This is first a rough transcription into ProofPower of the simple HOL4 proof of the Church Rosser theorem for the pure combinatory logic, eventually perhaps to be upgraded to an illative combinatory logic.

Beause I am exploring this approach with a view to extending it in certain ways, I will be making some changes along the way, above and beyond those arising from recasting the material to work in ProofPower HOL.

The two principal extensions to this work which are of interest to me are firstly to go from pure combinatory logic to illative combinatory logics, which one could do with essentially the same term structure (but more combinators), and secondly to move to infinitary combinators.

I therefore aim to maximise the applicability of each part of the development so that as little as possible will need to be reworked as I undertake those extensions.

This may eventually result in the material being split across multiple theories and multiple documents, but initially, apart from the placement in t005 [2] of general results on reflexive transitive closure I will stick to one theory. The strategy in this respect is firstly to separate out materials which do not depend on term structure (and present this first), then to adopt a structure for finitary terms which is extensible in the number of primitive combinators.

Since we don't have a datatype package I will follow my own usual practice of using a set theoretic approach to the syntax.

SML

```
open_theory "misc3";
force_new_theory "icl";
force_new_pc "'icl";
merge_pcs ["'savedthm_cs_∃_proof"] "'icl";
```

SML

```
set_merge_pcs ["misc31", "'icl"];
```

3 Reduction

In this section I deal with matters which do not in the hol4 treatment depend upon the type of combinatory terms, and some matters which in the hol4 treatment do but need not. The latter requires some restructuring of the proof strategy

3.1 Diamond Preserved by RTC

In this section we obtain the result that the recursive transitive closure of a relation satisfying the diamond property also satisfies that property.

The proof in the hol4 version is slick because RTC is redefined using a package for defining relations which delivers some nice induction principles.

HOL Constant

$\boldsymbol{confluent}$: $('a \to 'a \to BOOL) \to BOOL$

$\forall R\bullet\ confluent\ R \Leftrightarrow \forall x\ y\ z\bullet\ rtc\ R\ x\ y \wedge rtc\ R\ x\ z \Rightarrow$
$\exists u\bullet\ rtc\ R\ y\ u \wedge rtc\ R\ z\ u$

HOL Constant

$\boldsymbol{normform}$: $('a \to 'a \to BOOL) \to 'a \to BOOL$

$\forall R\ x\bullet\ normform\ R\ x \Leftrightarrow \forall y\bullet \neg\ R\ x\ y$

$\boldsymbol{confluent\_normforms\_unique} =$
$\vdash\ \forall\ R\bullet\ confluent\ R \Rightarrow$
$(\forall\ x\ y\ z\bullet\ rtc\ R\ x\ y \wedge normform\ R\ y \wedge rtc\ R\ x\ z \wedge normform\ R\ z$
$\Rightarrow\ y = z)$

HOL Constant

$\boldsymbol{diamond}$: $('a \to 'a \to BOOL) \to BOOL$

$\forall R\bullet\ diamond\ R = \forall x\ y\ z\bullet\ R\ x\ y \wedge R\ x\ z \Rightarrow \exists u\bullet\ R\ y\ u \wedge R\ z\ u$

$\boldsymbol{confluent\_diamond\_rtc} =$
$\vdash\ \forall\ R\bullet\ confluent\ R \Leftrightarrow diamond\ (rtc\ R)$

$\boldsymbol{R\_rtc\_diamond} =$
$\vdash\ \forall\ R\bullet\ diamond\ R \Rightarrow (\forall\ x\ p\bullet\ rtc\ R\ x\ p \Rightarrow$
$(\forall\ z\bullet\ R\ x\ z \Rightarrow (\exists\ u\bullet\ rtc\ R\ p\ u \wedge rtc\ R\ z\ u)))$

RTC_RTC is the transitivity of reflexive transitive closure, which we have as $tran\_rtc\_thm2$.

$\boldsymbol{diamond\_RTC\_lemma} =$
$\vdash\ \forall\ R\bullet\ diamond\ R \Rightarrow (\forall\ x\ y\bullet\ rtc\ R\ x\ y$
$\Rightarrow (\forall\ z\bullet\ rtc\ R\ x\ z \Rightarrow (\exists\ u\bullet\ rtc\ R\ y\ u \wedge rtc\ R\ z\ u)))$

$\boldsymbol{diamond\_RTC} =$
$\vdash\ \forall\ R\bullet\ diamond\ R \Rightarrow diamond\ (rtc\ R)$

3.2 Further Results about Reduction Relations

In the hol4 proof reduction is defined over combinatory terms using a relation definition package which results in the following definition (this is the result of a modified notion admitting more combinatory constants):

```
redn_def
|- $--> =
  (λa0 a1.
    ∀-->' .
      (∀a0 a1.
        (∃x y f. (a0 = f # x) ∧ (a1 = f # y) ∧ -->' x y) ∨
        (∃f g x. (a0 = f # x) ∧ (a1 = g # x) ∧ -->' f g) ∨
        (∃y. a0 = C 0 # a1 # y) ∨
        (∃f g x.
          (a0 = C 1 # f # g # x) ∧ (a1 = f # x # (g # x))) ∨
        (∃x. (a0 = C 2 # x # x) ∧ (a1 = C 0)) ⇒
        -->' a0 a1) ⇒
      -->' a0 a1)
```

Of the various parts in this definition, the first two make this a congruence relation, i.e. reductions on parts yeild reductions on the whole, the third is the primitive reduction for K, the fourth for S, the fifth for Q. These are wrapped up to assert that the defined relation is the intersection of all relations closed under these rules.

The

So the natural way to proceed in the absence of the rule-based relation definition facility in hol4 is to define the primitive relations for each combinator, combine them (take the union or disjunction), form the congruence and take the transitive closure.

4 Combinatory Terms

4.1 Introduction

In hol4 the syntax of combinatory logic is give as a single line definition of a recursive datatype. We can't do this in ProofPower, datatypes have to be manually cranked. This section replicates that process, and aims to replicate pretty closely what hol4 produces for that datatype.

I did first of all modify the type to allow any number of combinatory constants after which the definition read:

```
hol4
val _ = Hol_datatype `CT = C of num | # of CT => CT`;
```

The theory resulting from that datatype definition is as follows:

```
hol4
Theory: cl

Parents:
    list

Type constants:
    CT 0

```

Term constants:

$\#$ $:CT \to CT \to CT$

C $:num \to CT$

CT_case $:(num \to \alpha) \to (CT \to CT \to \alpha) \to CT \to \alpha$

CT_size $:CT \to num$

Definitions:

CT_TY_DEF

$$\vdash \exists rep.\ TYPE\_DEFINITION\ (\lambda a0'.\ \forall' CT'.\ (\forall a0'.\ (\exists a.\ a0' = (\lambda a.\ ind\_type\$CONSTR\ 0\ a\ (\lambda n.\ ind\_type\$BOTTOM))\ a) \vee (\exists a0\ a1.\ (a0' = (\lambda a0\ a1.\ ind\_type\$CONSTR\ (SUC\ 0)\ ARB\ (ind\_type\$FCONS\ a0\ (ind\_type\$FCONS\ a1\ (\lambda n.\ ind\_type\$BOTTOM))))\ a0\ a1) \wedge {}'CT'\ a0 \wedge {}'CT'\ a1) \Rightarrow {}'CT'\ a0') \Rightarrow {}'CT'\ a0')\ rep$$

CT_case_def

$$\vdash (\forall f\ f1\ a.\ CT\_case\ f\ f1\ (C\ a) = f\ a) \wedge \forall f\ f1\ a0\ a1.\ CT\_case\ f\ f1\ (\#\ a0\ a1) = f1\ a0\ a1$$

CT_size_def

$$\vdash (\forall a.\ CT\_size\ (C\ a) = 1 + a) \wedge \forall a0\ a1.\ CT\_size\ (\#\ a0\ a1) = 1 + (CT\_size\ a0 + CT\_size\ a1)$$

Theorems:

$datatype\_CT$ $\vdash DATATYPE\ (CT\ C\ \#)$

CT_11

$$\vdash (\forall a\ a'.\ (C\ a = C\ a') \Leftrightarrow (a = a')) \wedge \forall a0\ a1\ a0'\ a1'.\ (\#\ a0\ a1 = \#\ a0'\ a1') \Leftrightarrow (a0 = a0') \wedge (a1 = a1')$$

$CT\_distinct$ $\vdash \forall a1\ a0\ a.\ C\ a \neq \#\ a0\ a1$

CT_case_cong

$$\vdash \forall M\ M'\ f\ f1.\ (M = M') \wedge (\forall a.\ (M' = C\ a) \Rightarrow (f\ a = f'\ a)) \wedge (\forall a0\ a1.\ (M' = \#\ a0\ a1) \Rightarrow (f1\ a0\ a1 = f1'\ a0\ a1)) \Rightarrow (CT\_case\ f\ f1\ M = CT\_case\ f'\ f1'\ M')$$

$CT\_nchotomy \ \vdash \forall CC.\ (\exists n.\ CC = C\ n) \vee \exists C\ C0.\ CC = \#\ C\ C0$
CT_Axiom
$\vdash \forall f0\ f1.$
$\quad \exists fn.$
$\quad\quad (\forall a.\ fn\ (C\ a) = f0\ a) \wedge$
$\quad\quad \forall a0\ a1.\ fn\ (\#\ a0\ a1) = f1\ a0\ a1\ (fn\ a0)\ (fn\ a1)$
$CT\_induction$
$\vdash \forall P.\ (\forall n.\ P\ (C\ n)) \wedge (\forall C\ C0.\ P\ C \wedge P\ C0 \Rightarrow P\ (\#\ C\ C0)) \Rightarrow \forall C.\ P\ C$

In this section I aim to replicate this quite closely, though using a different representation. I am assuming that once this is done here, there will be no further need to refer to the representing type.

4.2 Introducing the type CT

The natural way to hand crank such a datatype in ProofPower HOL would be to use the theory of trees in Arthan's mathematical examples as a representation type. However, I am more familiar with using the domain of an axiomatic set theory so it will be slightly easier for me to do that.

I don't define the constructors over the representation type since they are simple enough to write out, instead I go straight to defining the closure conditions on the set of representatives.

There are two constructors, a constant constructor and an application constructor. There are countably many constants named by natural numbers, and tagged by zero to distinguish them from applications. An application is constructed from two combinators and is tagged by 1.

The closure condition is:

HOL Constant

$\boldsymbol{CTrepClosed}$: $(ONE\ GSU \to BOOL) \to BOOL$

$\forall\ s\bullet\ CTrepClosed\ s \Leftrightarrow (\forall n\bullet\ s\ (Nat_u\ 0 \mapsto_u Nat_u\ n))$
$\quad \wedge\ (\forall f\ a\bullet\ s\ f \wedge s\ a \Rightarrow s\ (Nat_u\ 1 \mapsto_u (f \mapsto_u a)))$

The property of being a combinatory term is then the smallest property which is *CTrepClosed*.

HOL Constant

$\boldsymbol{CTsyntax}$: $ONE\ GSU \to BOOL$

$\forall s\bullet\ CTsyntax\ s = \forall t \bullet CTrepClosed\ t \Rightarrow t\ s$

The following theorems about the representation are helpful in proving related result for the new type:

$\boldsymbol{CTrepclosed\_CTsyntax\_lemma1} =$
$\quad \vdash CTrepClosed\ CTsyntax$

$\boldsymbol{CTrepclosed\_CTsyntax\_thm} =$
$\quad \vdash (\forall n\bullet\ CTsyntax\ (Nat_u\ 0 \mapsto_u Nat_u\ n))$
$\quad \wedge\ (\forall f\ a\bullet\ CTsyntax\ f \wedge CTsyntax\ a \Rightarrow CTsyntax\ (Nat_u\ 1 \mapsto_u f \mapsto_u a))$

$$\textbf{\textit{CTrepclosed\_CTsyntax\_lemma2}} =$$
$$\vdash \forall s\bullet\ CTrepClosed\ s \Rightarrow (\forall\ t\bullet\ CTsyntax\ t \Rightarrow s\ t)$$

$$\textbf{\textit{CTsyntax\_cases\_thm}} =$$
$$\vdash \forall c\bullet\ CTsyntax\ c$$
$$\Rightarrow (\exists\ n\bullet\ c = Nat_u\ 0 \mapsto_u Nat_u\ n)$$
$$\vee\ (\exists\ f\ a\bullet\ CTsyntax\ f \wedge CTsyntax\ a \wedge c = Nat_u\ 1 \mapsto_u f \mapsto_u a)$$

To introduce the new type we must prove that *CTsyntax* is inhabited:

$$\textbf{\textit{CTsyntax\_nonempty}} =$$
$$\vdash \exists c\bullet\ CTsyntax\ c$$

The type *CT* is then introduced as follows:

SML
```
val CT_type_defn_thm = new_type_defn(["CT"], "CT", [], CTsyntax_nonempty);
```

Which gives the following result:

$$\textbf{\textit{CT\_type\_defn\_thm}} = \vdash \exists\ f\bullet\ TypeDefn\ CTsyntax\ f$$

From which we can derive:

$$\textbf{\textit{CT\_type\_lemma2}} =$$
$$\vdash \exists\ abs\ rep\bullet\ (\forall\ a\bullet\ abs\ (rep\ a) = a)$$
$$\wedge\ (\forall\ r\bullet\ CTsyntax\ r \Leftrightarrow rep\ (abs\ r) = r)$$
$$\wedge\ OneOne\ rep$$
$$\wedge\ (\forall\ a\bullet\ CTsyntax\ (rep\ a))$$

4.3 Primitive Constructors over CT

I could manage without defining the following mappings between the new type and its representations , but its easier to define them.

HOL Constant
$$\textbf{\textit{CTabs}}: ONE\ GSU \rightarrow CT;$$
$$\textbf{\textit{CTrep}}: CT \rightarrow ONE\ GSU$$

$$(\forall\ a\bullet\ CTabs\ (CTrep\ a) = a)$$
$$\wedge\ (\forall\ r\bullet\ CTsyntax\ r \Leftrightarrow CTrep\ (CTabs\ r) = r)$$
$$\wedge\ OneOne\ CTrep$$
$$\wedge\ (\forall\ a\bullet\ CTsyntax\ (CTrep\ a))$$

There will be two constructors one for constants and one for applications. Applications are infix.

SML
```
declare_infix(1000, "c");
```

HOL Constant

$\$_c$: $CT \to CT \to CT$

$\forall s\ t\bullet\ s\ {}_c\ t = CTabs\ (Nat_u\ 1 \mapsto_u (CTrep\ s) \mapsto_u (CTrep\ t))$

HOL Constant

C: $\mathbb{N} \to CT$

$\forall s\bullet\ C\ s = CTabs(Nat_u\ 0 \mapsto_u (Nat_u\ s))$

CT_cases =

$\vdash \forall\ t\bullet (\exists\ n\bullet\ t = C\ n) \vee (\exists\ c1\ c2\bullet\ t = c1\ {}_c\ c2)$

4.4 Induction and Recursion

Now we obtain a principle of induction for reasoning about these combinatory terms and a recursion theorem to justify definitions over the terms by pattern matching recursion.

We can derive a well-founded ordering for combinatory terms from the well-foundedness of membership over the sets used to represent the terms. This provides a first method of proof by induction over the combinatory terms which is used to prove the induction principle produced by hol4.

wf_CT_thm =

$\vdash well\_founded\ (\lambda\ x\ y\bullet\ CTrep\ x \in_u CTrep\ y)$

tc∈u_rep_c_lemma =

$\vdash \forall\ c1\ c2\bullet\ CTrep\ c1 \in_u{}^+\ CTrep\ (c1\ {}_c\ c2)$
$\wedge\ CTrep\ c2 \in_u{}^+\ CTrep\ (c1\ {}_c\ c2)$

CT_induction =

$\vdash \forall\ P\bullet (\forall\ n\bullet\ P\ (C\ n)) \wedge (\forall\ c\ c0\bullet\ P\ c \wedge P\ c0 \Rightarrow P\ (c\ {}_c\ c0))$
$\Rightarrow (\forall\ c\bullet\ P\ c)$

CT_axiom =

$\vdash \forall\ f0\ f1\bullet \exists\ fn\bullet$
$(\forall\ a\bullet\ fn\ (C\ a) = f0\ a)$
$\wedge (\forall\ a0\ a1\bullet\ fn\ (a0\ {}_c\ a1) = f1\ a0\ a1\ (fn\ a0)\ (fn\ a1))$

4.5 Further Definitions

We can now define constants over CT by pattern matching primitive recursion.

I would normally obtain a 'course of values' induction by defining a well-founded relation over the terms, which is easily done thus:

SML

```
declare_infix(200, "in_c");
```

HOL Constant

$\$\boldsymbol{in_c}$: $CT \to CT \to BOOL$

$\forall x\ y\bullet\ x\ in_c\ y \Leftrightarrow \exists z\bullet\ y = x\ _c\ z \vee y = z\ _c\ x$

However, the method used in the hol4 datatype system is to define a numeric 'size' for combinatory terms as follows:

HOL Constant

$\boldsymbol{CT\_size}$: $CT \to \mathbb{N}$

$(\forall a\bullet\ CT\_size\ (C\ a) = 1 + a)$
$\wedge\ \forall a0\ a1\bullet\ CT\_size\ (a0\ _c\ a1) = 1 + CT\_size\ a0 + CT\_size\ a1$

Course of values induction can then be obtained by induction over the size of combinatory terms.

HOL Constant

$\boldsymbol{CT\_case}$: $(\mathbb{N} \to CT) \to (CT \to CT \to CT) \to (CT \to CT)$

$\forall f\ g\ x\ y\ n\bullet\ CT\_case\ f\ g\ (C\ n) = f\ n$
$\wedge\ CT\_case\ f\ g\ (x\ _c\ y) = g\ x\ y$

We now prove the rest of the theorems which are derived automatically for the datatype in hol4.

$\boldsymbol{CT\_11}$ =
$\vdash (\forall\ a\ a'\bullet\ C\ a = C\ a' \Rightarrow a = a')$
$\wedge (\forall\ a0\ a1\ a0'\ a1'\bullet\ a0\ _c\ a1 = a0'\ _c\ a1' \Rightarrow a0 = a0' \wedge a1 = a1')$

$\boldsymbol{CT\_distinct}$ =
$\vdash \forall\ a\ a0\ a1\bullet \neg\ C\ a = a0\ _c\ a1$

$\boldsymbol{CT\_nchotomy}$ =
$\vdash \forall\ cc\bullet \neg\ ((\exists\ n\bullet\ cc = C\ n) \Leftrightarrow (\exists\ c\ c0\bullet\ cc = c\ _c\ c0))$

4.6 Combinator Names

In the original hol4 treatment the combinators K and S were constructors of the abstract datatype. In this quasi replica I made the minor generalisation of allowing countably many constants using a constructor taking a natural number parameter.

To make the material more legible I now define constants to give the conventional names to the S and K constructors as follows.

HOL Constant

$\boldsymbol{K_c}$: CT

$K_c = C\ 0$

HOL Constant

$$\boldsymbol{S_c}\colon CT$$

$$S_c = C\ 1$$

Once I have replicated essentially the same Church Rosser proof for this pure combinatory logic I will then attempt a proof for an illative system with one illative combinatory. This will be called 'Q'.

HOL Constant

$$\boldsymbol{Q_c}\colon CT$$

$$Q_c = C\ 2$$

Since 'Q' is intended to be a close approximation to equality, it might be nice to have the following way of writing equations:

SML

```
declare_infix(300, "=c");
```

HOL Constant

$$\$=_{\boldsymbol{c}}\colon CT \to CT \to CT$$

$$\forall a0\ a1 \bullet a0 =_c a1 = (Q_c\ _c\ a0)\ _c\ a1$$

This kind of equality delivers a combinatory term rather than a BOOL. True and false will probably be coded as left and right projections, i.e. K and KI (where I is the identity, which is SKK).

These definitions really only give names to the combinators, to know what they mean you have to look at the reduction relations over the combinators.

SML

```
set_flag ("subgoal_package_quiet", false);
set_flag ("pp_use_alias", true);
```

5 Postscript

Nothing to say yet.

A Theory Listings

A.1 The Theory icl

Parents

misc3

Constants

$\boldsymbol{confluent}$	$('a,\ BOOL)\ BR \to BOOL$
$\boldsymbol{normform}$	$('a,\ BOOL)\ BR \to 'a \to BOOL$
$\boldsymbol{diamond}$	$('a,\ BOOL)\ BR \to BOOL$
$\boldsymbol{CTrepClosed}$	$(ONE\ GSU \to BOOL) \to BOOL$
$\boldsymbol{CTsyntax}$	$ONE\ GSU \to BOOL$
$\boldsymbol{CTrep}$	$CT \to ONE\ GSU$
$\boldsymbol{CTabs}$	$ONE\ GSU \to CT$
$\$_{\boldsymbol{c}}$	$(CT,\ CT)\ BR$
$\boldsymbol{C}$	$\mathbb{N} \to CT$
$\$\boldsymbol{in}_{\boldsymbol{c}}$	$(CT,\ BOOL)\ BR$
$\boldsymbol{CT\_size}$	$CT \to \mathbb{N}$
$\boldsymbol{CT\_case}$	$(\mathbb{N} \to CT) \to (CT,\ CT)\ BR \to CT \to CT$
$\boldsymbol{K}_{\boldsymbol{c}}$	CT
$\boldsymbol{S}_{\boldsymbol{c}}$	CT
$\boldsymbol{Q}_{\boldsymbol{c}}$	CT
$\$=_{\boldsymbol{c}}$	$(CT,\ CT)\ BR$

Types

$\boldsymbol{CT}$

Fixity

Right Infix 200:

$\boldsymbol{in}_{\boldsymbol{c}}$

Right Infix 300:

$=_{\boldsymbol{c}}$

Right Infix 1000:

$\boldsymbol{c}$

Definitions

confluent $\vdash \forall\ R$
$\bullet\ confluent\ R$
$\Leftrightarrow (\forall\ x\ y\ z$
$\bullet\ rtc\ R\ x\ y \wedge rtc\ R\ x\ z$
$\Rightarrow (\exists\ u \bullet\ rtc\ R\ y\ u \wedge rtc\ R\ z\ u))$

normform $\vdash \forall\ R\ x \bullet\ normform\ R\ x \Leftrightarrow (\forall\ y \bullet \neg\ R\ x\ y)$

diamond $\vdash \forall\ R$
$\bullet\ diamond\ R$
$\Leftrightarrow (\forall\ x\ y\ z \bullet\ R\ x\ y \wedge R\ x\ z \Rightarrow (\exists\ u \bullet\ R\ y\ u \wedge R\ z\ u))$

CTrepClosed $\vdash \forall\ s$
$\bullet\ CTrepClosed\ s$
$\Leftrightarrow (\forall\ n \bullet\ s\ (Nat_u\ 0 \mapsto_u Nat_u\ n))$
$\wedge\ (\forall\ f\ a \bullet\ s\ f \wedge s\ a \Rightarrow s\ (Nat_u\ 1 \mapsto_u f \mapsto_u a))$

CTsyntax $\vdash \forall\ s \bullet\ CTsyntax\ s \Leftrightarrow (\forall\ t \bullet\ CTrepClosed\ t \Rightarrow t\ s)$

CT $\vdash \exists\ f \bullet\ TypeDefn\ CTsyntax\ f$

CTabs

CTrep $\vdash (\forall\ a \bullet\ CTabs\ (CTrep\ a) = a)$
$\wedge\ (\forall\ r \bullet\ CTsyntax\ r \Leftrightarrow CTrep\ (CTabs\ r) = r)$
$\wedge\ OneOne\ CTrep$
$\wedge\ (\forall\ a \bullet\ CTsyntax\ (CTrep\ a))$

c $\vdash \forall\ s\ t$
$\bullet\ s\ _c\ t = CTabs\ (Nat_u\ 1 \mapsto_u CTrep\ s \mapsto_u CTrep\ t)$

C $\vdash \forall\ s \bullet\ C\ s = CTabs\ (Nat_u\ 0 \mapsto_u Nat_u\ s)$

in_c $\vdash \forall\ x\ y \bullet\ x\ in_c\ y \Leftrightarrow (\exists\ z \bullet\ y = x\ _c\ z \vee y = z\ _c\ x)$

CT_size $\vdash (\forall\ a \bullet\ CT\_size\ (C\ a) = 1 + a)$
$\wedge\ (\forall\ a0\ a1$
$\bullet\ CT\_size\ (a0\ _c\ a1) = 1 + CT\_size\ a0 + CT\_size\ a1)$

CT_case $\vdash \forall\ f\ g\ x\ y\ n$
$\bullet\ CT\_case\ f\ g\ (C\ n) = f\ n$
$\wedge\ CT\_case\ f\ g\ (x\ _c\ y) = g\ x\ y$

K_c $\vdash K_c = C\ 0$

S_c $\vdash S_c = C\ 1$

Q_c $\vdash Q_c = C\ 2$

=_c $\vdash \forall\ a0\ a1 \bullet\ a0 =_c a1 = (Q_c\ _c\ a0)\ _c\ a1$

Theorems

confluent_normforms_unique
$\vdash \forall\ R$
$\bullet\ confluent\ R$
$\Rightarrow (\forall\ x\ y\ z$
$\bullet\ rtc\ R\ x\ y$
$\wedge\ normform\ R\ y$
$\wedge\ rtc\ R\ x\ z$
$\wedge\ normform\ R\ z$
$\Rightarrow y = z)$

confluent_diamond_rtc
$\vdash \forall\ R \bullet\ confluent\ R \Leftrightarrow diamond\ (rtc\ R)$

R_rtc_diamond

$\vdash \forall\ R$
$\bullet\ diamond\ R$
$\Rightarrow (\forall\ x\ p$
$\bullet\ rtc\ R\ x\ p$
$\Rightarrow (\forall\ z$
$\bullet\ R\ x\ z \Rightarrow (\exists\ u \bullet\ rtc\ R\ p\ u \wedge rtc\ R\ z\ u)))$

diamond_RTC_lemma
$\vdash \forall\ R$
$\bullet\ diamond\ R$
$\Rightarrow (\forall\ x\ y$
$\bullet\ rtc\ R\ x\ y$
$\Rightarrow (\forall\ z$
$\bullet\ rtc\ R\ x\ z \Rightarrow (\exists\ u \bullet\ rtc\ R\ y\ u \wedge rtc\ R\ z\ u)))$

diamond_RTC $\vdash \forall\ R \bullet\ diamond\ R \Rightarrow diamond\ (rtc\ R)$

CTrepclosed_CTsyntax_lemma1
$\vdash CTrepClosed\ CTsyntax$

CTrepclosed_CTsyntax_thm
$\vdash (\forall\ n \bullet\ CTsyntax\ (Nat_u\ 0 \mapsto_u Nat_u\ n))$
$\wedge (\forall\ f\ a$
$\bullet\ CTsyntax\ f \wedge CTsyntax\ a$
$\Rightarrow CTsyntax\ (Nat_u\ 1 \mapsto_u f \mapsto_u a))$

CTrepclosed_CTsyntax_lemma2
$\vdash \forall\ s \bullet\ CTrepClosed\ s \Rightarrow (\forall\ t \bullet\ CTsyntax\ t \Rightarrow s\ t)$

CTsyntax_cases_thm
$\vdash \forall\ c$
$\bullet\ CTsyntax\ c$
$\Rightarrow (\exists\ n \bullet\ c = Nat_u\ 0 \mapsto_u Nat_u\ n)$
$\vee (\exists\ f\ a$
$\bullet\ CTsyntax\ f$
$\wedge\ CTsyntax\ a$
$\wedge\ c = Nat_u\ 1 \mapsto_u f \mapsto_u a)$

CT_cases $\vdash \forall\ t \bullet (\exists\ n \bullet\ t = C\ n) \vee (\exists\ c1\ c2 \bullet\ t = c1\ {}_c\ c2)$

wf_CT_thm $\vdash well\_founded\ (\lambda\ x\ y \bullet\ CTrep\ x \in_u{}^+\ CTrep\ y)$

tc$\in_u$_rep_c_lemma
$\vdash \forall\ c1\ c2$
$\bullet\ CTrep\ c1 \in_u{}^+ CTrep\ (c1\ {}_c\ c2)$
$\wedge\ CTrep\ c2 \in_u{}^+ CTrep\ (c1\ {}_c\ c2)$

CT_induction $\vdash \forall\ P$
$\bullet (\forall\ n \bullet\ P\ (C\ n)) \wedge (\forall\ c\ c0 \bullet\ P\ c \wedge P\ c0 \Rightarrow P\ (c\ {}_c\ c0))$
$\Rightarrow (\forall\ c \bullet\ P\ c)$

CT_axiom $\vdash \forall\ f0\ f1$
$\bullet\ \exists\ fn$
$\bullet (\forall\ a \bullet\ fn\ (C\ a) = f0\ a)$
$\wedge (\forall\ a0\ a1$
$\bullet\ fn\ (a0\ {}_c\ a1) = f1\ a0\ a1\ (fn\ a0)\ (fn\ a1))$

CT_11 $\vdash (\forall\ a\ a' \bullet\ C\ a = C\ a' \Rightarrow a = a')$
$\wedge (\forall\ a0\ a1\ a0'\ a1'$
$\bullet\ a0\ {}_c\ a1 = a0'\ {}_c\ a1' \Rightarrow a0 = a0' \wedge a1 = a1')$

CT_distinct $\vdash \forall\ a\ a0\ a1 \bullet \neg\ C\ a = a0\ {}_c\ a1$

CT_case_cong $\vdash \forall\ M\ M'\ f\ f1$
$\bullet\ M = M'$

$$\begin{array}{l} \wedge\ (\forall\ a\bullet\ M' = C\ a \Rightarrow f\ a = f'\ a) \\ \wedge\ (\forall\ a0\ a1 \\ \bullet\ M' = a0\ {}_c\ a1 \Rightarrow f1\ a0\ a1 = f1'\ a0\ a1) \\ \Rightarrow\ CT\_case\ f\ f1\ M = CT\_case\ f'\ f1'\ M' \end{array}$$

$$\boldsymbol{CT\_nchotomy}\ \vdash\ \forall\ cc\bullet\ \neg\ ((\exists\ n\bullet\ cc = C\ n) \Leftrightarrow (\exists\ c\ c0\bullet\ cc = c\ {}_c\ c0))$$

Bibliography

[1] Roger Bishop Jones. Introduction to Work in Progress. *RBJones.com*, 2010. http://www.rbjones.com/rbjpub/pp/doc/t000.pdf.

[2] Roger Bishop Jones. Well Founded Relations. *RBJones.com*, 2010. http://www.rbjones.com/rbjpub/pp/doc/t005.pdf.

Index