Formalisation of Logical Relations proofs using the Nominal Package

Julien Narboux and Christian Urban

June 13, 2007

Abstract

We present in this paper a formalisation of the chapter *Logical Relations and a Case Study in Equivalence Checking* by Karl Crary from the book on *Advanced Topics in Types and Programming Languages*, MIT Press 2005. We use a fully nominal approach to deal with binders. The formalisation has been performed within the Isabelle/HOL proof assistant using the Nominal Package.

Contents

1 Introduction

For several reasons, proof assistants can be useful for proving properties of programming languages. Indeed, often the proofs consist in inductions involving cases, many of which are trivial. But it hard to guess in advance which case is trivial and even a small error can invalidate a result. Even more the use of a proof assistant can also help the researcher: it is possible to quickly check after a modification of the definitions if the proof is still valid. But in practice, the formalisation of proofs about programming has to address many troubles. The main problem, which is well known in the community is the representation of binders. Informal proofs contains arguments such as 'by renaming of the variables' or 'reasoning modulo alpha conversion'. These arguments are very hard to formalise. Several solutions have been proposed to try to solve this problem. On solution to represent binder is by using De-Bruijn indices. This alleviates such problems about too many details and in some cases leads to very slick proofs. Unfortunately, by using De-Bruijn indices the "symbol-pushing" involves a rather large amount of arithmetic on indices which is not present in informal descriptions. Another method of representing binders is by using higher-order abstract-syntax (HOAS) where the meta-language provides binding-constructs. The disadvantage with HOAS is that one has to encode the language at hand and use the reasoning infrastructure the theorem prover, for example Twelf, provides. In practice this means often that reasoning does not proceed as one would expect from the informal reasoning on paper.

These solution tend to force the user of the system to modify his proofs. We think that this should be the opposite, the system should be modified.

That is why we are currently developing a package for the Isabelle/HOL proof assistant [3]. which provides an infrastructure in the theorem prover Isabelle/HOL for representing binders as *named* α-equivalence classes [1, 5, 4].

In this paper, we formalise the chapter about Logical Relation and a Case Study in Equivalence Checking by Karl Crary of the book *Advanced Topics in Types and Programming Languages*[2]. This example is interesting because logical relations are a fundamental technique for proving properties of programming languages. The purpose of this formalisation is to test and improve the Nominal Package in the context of a 'real life' example. Indeed, this chapter is not an exception, the problem of binders is treated informally, on the first page the reader can find the following sentence: 'As usual, we will identify terms that differ only in the names of bound variables, and our substitution is capture avoiding'.

The formalisation we provide has been realized withing the Isar language [6] within the Isabelle/HOL proof assistant[3]. The definitions and proofs given in this paper have been generated automatically from the formal proofs.

theory *Crary*
  imports ../*Nominal*

2 Definition of the language

2.1 Definition of the terms and types

First we define the type of atom names which will be used for binders. Each atom type is infinitely many atoms and equality is decidable.

atom-decl *name*

We define the datatype representing types. Although, It does not contain any binder we still use the `nominal_datatype` command because the Nominal datatype package will prodive permutation functions and useful lemmas.

nominal-datatype *ty* =
  TBase
| *TUnit*
| *Arrow ty ty* (-→- [*100*,*100*] *100*)

The datatype of terms contains a binder. The notation ≪*name*≫ *trm* means that the name is bound inside trm.

nominal-datatype *trm* =
  Unit
| *Var name*
| *Lam* ≪*name*≫*trm* (*Lam* [-].- [*100*,*100*] *100*)
| *App trm trm*
| *Const nat*

types *Ctxt* = (*name*×*ty*) *list*
types *Subst* = (*name*×*trm*) *list*

As the datatype of types does not contain any binder, the application of a permutation is the identity function. In the future, this should be automatically derived by the package.

lemma *perm-ty*[*simp*]:
 fixes *T*::*ty*
 and *pi*::*name prm*
 shows $pi \cdot T = T$
 by (*induct T rule*: *ty.weak-induct*) (*simp-all*)

lemma *fresh-ty*[*simp*]:
 fixes *x*::*name*
 and *T*::*ty*
 shows $x \# T$
 by (*simp add*: *fresh-def supp-def*)

lemma *ty-cases*:
 fixes *T*::*ty*
 shows $(\exists\ T_1\ T_2.\ T{=}T_1{\to}T_2) \lor T{=}TUnit \lor T{=}TBase$
by (*induct T rule*:*ty.weak-induct*) (*auto*)

2.2 Size functions

We define size functions for types and terms. As Isabelle allows overloading we can use the same notation for both functions.

These functions are automatically generated for non nominal datatypes. In the future, we need to extend the package to generate size functions automatically for nominal datatypes as well.

The definition of a function using the nominal package generates four groups of proof obligations.

The first group are goal of the form `finite(supp ())`, these often be solve using the `finite_guess` tactic. The second group of goals corresponds to the invariant. If the user has not chosen to setup an invariant, then it just true and hence can easily be solved.

instance *ty* :: *size* **..**

nominal-primrec
  size (*TBase*) = *1*
  size (*TUnit*) = *1*
  size ($T_1 \rightarrow T_2$) = *size* T_1 + *size* T_2
by (*rule TrueI*)+

lemma *ty-size-greater-zero*[*simp*]:
  fixes *T*::*ty*
  shows *size T* > *0*
by (*nominal-induct rule*:*ty.induct*) (*simp-all*)

3 Capture-avoiding substitutions

In this section we define parallel substitution. The usual substitution will be derived as a special case of parallel substitution. But first we define a function to lookup for the term corresponding to a type in an association list. Note that if the term does not appear in the list then we return a variable of that name.

fun
  lookup :: *Subst* ⇒ *name* ⇒ *trm*
where
  lookup [] *x*   = *Var x*
| *lookup* ((*y*,*T*)#θ) *x* = (*if x*=*y then T else lookup* θ *x*)

lemma *lookup-eqvt*[*eqvt*]:
  fixes *pi*::*name prm*
  shows *pi*•(*lookup* θ *x*) = *lookup* (*pi*•θ) (*pi*•*x*)
by (*induct* θ) (*auto simp add*: *perm-bij*)

lemma *lookup-fresh*:
  fixes *z*::*name*
  assumes *a*: *z*#θ *z*#*x*
  shows *z*# *lookup* θ *x*
using *a*
by (*induct rule*: *lookup.induct*)
   (*auto simp add*: *fresh-list-cons*)

lemma *lookup-fresh*′:

assumes *a*: *z*#*θ*
shows *lookup θ z* = *Var z*
using *a*
by (*induct rule*: *lookup.induct*)
(*auto simp add*: *fresh-list-cons fresh-prod fresh-atm*)

3.1 Parallel substitution

consts
psubst :: *Subst* ⇒ *trm* ⇒ *trm* (-<-> [*60*,*100*] *100*)

nominal-primrec
θ<(*Var x*)> = (*lookup θ x*)
θ<(*App* t_1 t_2)> = *App* (*θ*<t_1>) (*θ*<t_2>)
x#*θ* ⟹ *θ*<(*Lam* [*x*].*t*)> = *Lam* [*x*].(*θ*<*t*>)
θ<(*Const n*)> = *Const n*
θ<(*Unit*)> = *Unit*
apply(*finite-guess*)+
apply(*rule TrueI*)+
apply(*simp add*: *abs-fresh*)+
apply(*fresh-guess*)+
done

3.2 Substitution

The substitution function is defined just as a special case of parallel substitution.

abbreviation
subst :: *trm* ⇒ *name* ⇒ *trm* ⇒ *trm* (-[-::=-] [*100*,*100*,*100*] *100*)
where
t[*x*::=*t'*] ≡ ([(*x*,*t'*)])<*t*>

lemma *subst*[*simp*]:
shows (*Var x*)[*y*::=*t'*] = (*if x*=*y then t' else* (*Var x*))
and (*App* t_1 t_2)[*y*::=*t'*] = *App* (t_1[*y*::=*t'*]) (t_2[*y*::=*t'*])
and *x*#(*y*,*t'*) ⟹ (*Lam* [*x*].*t*)[*y*::=*t'*] = *Lam* [*x*].(*t*[*y*::=*t'*])
and *Const n*[*y*::=*t'*] = *Const n*
and *Unit* [*y*::=*t'*] = *Unit*
by (*simp-all add*: *fresh-list-cons fresh-list-nil*)

lemma *subst-eqvt*[*eqvt*]:
fixes *pi*::*name prm*
shows *pi*•(*t*[*x*::=*t'*]) = (*pi*•*t*)[(*pi*•*x*)::=(*pi*•*t'*)]
by (*nominal-induct t avoiding*: *x t' rule*: *trm.induct*)
(*perm-simp add*: *fresh-bij*)+

3.3 Lemmas about freshness and substitutions

lemma *subst-rename*:
fixes *c*::*name*
assumes *a*: *c*#t_1

shows $t_1[a{::=}t_2] = ([(c,a)]\bullet t_1)[c{::=}t_2]$
using *a*
apply(*nominal-induct* t_1 *avoiding*: *a c* t_2 *rule*: *trm.induct*)
apply(*simp add*: *trm.inject calc-atm fresh-atm abs-fresh perm-nat-def*)+
done

lemma *fresh-psubst*:
fixes *z*::*name*
assumes *a*: $z\#t$ $z\#\theta$
shows $z\#(\theta$<*t*>$)$
using *a*
by (*nominal-induct t avoiding*: *z* θ *t rule*: *trm.induct*)
(*auto simp add*: *abs-fresh lookup-fresh*)

lemma *fresh-subst''*:
fixes *z*::*name*
assumes $z\#t_2$
shows $z\#t_1[z{::=}t_2]$
using *assms*
by (*nominal-induct* t_1 *avoiding*: t_2 *z rule*: *trm.induct*)
(*auto simp add*: *abs-fresh fresh-nat fresh-atm*)

lemma *fresh-subst'*:
fixes *z*::*name*
assumes $z\#[y].t_1$ $z\#t_2$
shows $z\#t_1[y{::=}t_2]$
using *assms*
by (*nominal-induct* t_1 *avoiding*: *y* t_2 *z rule*: *trm.induct*)
(*auto simp add*: *abs-fresh fresh-nat fresh-atm*)

lemma *fresh-subst*:
fixes *z*::*name*
assumes *a*: $z\#t_1$ $z\#t_2$
shows $z\#t_1[y{::=}t_2]$
using *a*
by (*auto simp add*: *fresh-subst' abs-fresh*)

lemma *fresh-psubst-simp*:
assumes $x\#t$
shows $(x,u)\#\theta$<*t*> = θ<*t*>
using *assms*
proof (*nominal-induct t avoiding*: *x u* θ *rule*: *trm.induct*)
case (*Lam y t x u*)
have *fs*: $y\#\theta$ $y\#x$ $y\#u$ **by** *fact*
moreover have $x\#$ *Lam* $[y].t$ **by** *fact*
ultimately have $x\#t$ **by** (*simp add*: *abs-fresh fresh-atm*)
moreover have *ih*:$\bigwedge n\ T.\ n\#t \Longrightarrow ((n,T)\#\theta)$<*t*> = θ<*t*> **by** *fact*
ultimately have $(x,u)\#\theta$<*t*> = θ<*t*> **by** *auto*
moreover have $(x,u)\#\theta$<*Lam* $[y].t$> = *Lam* $[y].$ $((x,u)\#\theta$<*t*>$)$ **using** *fs*
by (*simp add*: *fresh-list-cons fresh-prod*)
moreover have θ<*Lam* $[y].t$> = *Lam* $[y].$ $(\theta$<*t*>$)$ **using** *fs* **by** *simp*
ultimately show $(x,u)\#\theta$<*Lam* $[y].t$> = θ<*Lam* $[y].t$> **by** *auto*
qed (*auto simp add*: *fresh-atm abs-fresh*)

```
lemma forget:
  fixes x::name
  assumes a: x#t
  shows t[x::=t'] = t
  using a
by (nominal-induct t avoiding: x t' rule: trm.induct)
   (auto simp add: fresh-atm abs-fresh)

lemma subst-fun-eq:
  fixes u::trm
  assumes h:[x].t1 = [y].t2
  shows t1[x::=u] = t2[y::=u]
proof −
  {
    assume x=y and t1=t2
    then have ?thesis using h by simp
  }
  moreover
  {
    assume h1:x ≠ y and h2:t1=[(x,y)] • t2 and h3:x # t2
    then have ([(x,y)] • t2)[x::=u] = t2[y::=u] by (simp add: subst-rename)
    then have ?thesis using h2 by simp
  }
  ultimately show ?thesis using alpha h by blast
qed

lemma psubst-empty[simp]:
  shows []<t> = t
by (nominal-induct t rule: trm.induct)
   (auto simp add: fresh-list-nil)

lemma psubst-subst-psubst:
  assumes h:c#θ
  shows θ<t>[c::=s] = (c,s)#θ<t>
  using h
by (nominal-induct t avoiding: θ c s rule: trm.induct)
   (auto simp add: fresh-list-cons fresh-atm forget lookup-fresh lookup-fresh' fresh-psubst)

lemma subst-fresh-simp:
  assumes a: x#θ
  shows θ<Var x> = Var x
using a
by (induct θ arbitrary: x, auto simp add:fresh-list-cons fresh-prod fresh-atm)

lemma psubst-subst-propagate:
  assumes x#θ
  shows θ<t[x::=u]> = θ<t>[x::=θ<u>]
using assms
proof (nominal-induct t avoiding: x u θ rule: trm.induct)
  case (Var n x u θ)
  { assume x=n
    moreover have x#θ by fact
```

ultimately have $\theta<Var\ n[x::=u]> = \theta<Var\ n>[x::=\theta<u>]$ **using** *subst-fresh-simp* **by** *auto*
}
moreover
{ assume $h{:}x \neq n$
then have $x\#Var\ n$ **by** (*auto simp add*: *fresh-atm*)
moreover have $x\#\theta$ **by** *fact*
ultimately have $x\#\theta<Var\ n>$ **using** *fresh-psubst* **by** *blast*
then have $\theta<Var\ n>[x::=\theta<u>] = \theta<Var\ n>$ **using** *forget* **by** *auto*
then have $\theta<Var\ n[x::=u]> = \theta<Var\ n>[x::=\theta<u>]$ **using** *h* **by** *auto*
}
ultimately show *?case* **by** *auto*
next
case $(Lam\ n\ t\ x\ u\ \theta)$
have $fs{:}n\#x\ n\#u\ n\#\theta\ x\#\theta$ **by** *fact*
have $ih{:}\bigwedge\ y\ s\ \theta.\ y\#\theta \Longrightarrow ((\theta<(t[y::=s])>) = ((\theta<t>)[y::=(\theta<s>)]))$ **by** *fact*
have $\theta\ <(Lam\ [n].t)[x::=u]> = \theta<Lam\ [n].\ (t\ [x::=u])>$ **using** *fs* **by** *auto*
then have $\theta\ <(Lam\ [n].t)[x::=u]> = Lam\ [n].\ \theta<t\ [x::=u]>$ **using** *fs* **by** *auto*
moreover have $\theta<t[x::=u]> = \theta<t>[x::=\theta<u>]$ **using** *ih fs* **by** *blast*
ultimately have $\theta\ <(Lam\ [n].t)[x::=u]> = Lam\ [n].(\theta<t>[x::=\theta<u>])$ **by** *auto*
moreover have $Lam\ [n].(\theta<t>[x::=\theta<u>]) = (Lam\ [n].\theta<t>)[x::=\theta<u>]$ **using** *fs fresh-psubst* **by** *auto*
ultimately have $\theta<(Lam\ [n].t)[x::=u]> = (Lam\ [n].\theta<t>)[x::=\theta<u>]$ **using** *fs* **by** *auto*
then show $\theta<(Lam\ [n].t)[x::=u]> = \theta<Lam\ [n].t>[x::=\theta<u>]$ **using** *fs* **by** *auto*
qed (*auto*)

4 Typing

4.1 Typing contexts

This section contains the definition and some properties of a typing context. As the concept of context often appears in the litterature and is general, we should in the future provide these lemmas in a library.

Definition of the Validity of contexts

First we define what valid contexts are. Informally a context is valid is it does not contains twice the same variable.
We use the following two inference rules:

$$\textit{valid}\ []\ \text{V\_NIL} \qquad \frac{\textit{valid}\ \Gamma \qquad a\ \#\ \Gamma}{\textit{valid}\ ((a,\ T)\ \#\ \Gamma)}\text{V\_CONS}$$

We need to derive the equivariance lemma for the relation `valid`. If all the constants which appear in the inductive definition have previously been shown to be equivariant and the lemmas have been tagged using the equivariant attribute then this proof can automated using the `nominal_inductive` command.

equivariance *valid*

We obtain the following lemma under the name `valid.eqvt`:

$$\text{If } \textit{valid } x \text{ then } \textit{valid } (pi \cdot x).$$

Now, we generate the inversion lemma for non empty lists. We add the `elim` attribute to tell the automated tactics to use it.

inductive-cases2
valid-cons-elim-auto[*elim*]:*valid* $((x,T)\#\Gamma)$

The generated theorem is the following:

$$[\![\textit{valid } ((x,\ T)\ \#\ \Gamma);\ [\![\textit{valid } \Gamma;\ x\ \#\ \Gamma]\!] \Longrightarrow P]\!] \Longrightarrow P$$

Definition of sub-contexts The definition of sub context is standard. We do not use the subset definition to prevent the need for unfolding the definition. We include validity in the definition to shorten the statements.

abbreviation
sub-context :: *Ctxt* ⇒ *Ctxt* ⇒ *bool* (- ⊆ - [*55*,*55*] *55*)
where
$\Gamma_1 \subseteq \Gamma_2 \equiv \forall\, a\ T.\ (a,T) \in set\ \Gamma_1 \longrightarrow (a,T) \in set\ \Gamma_2$

Lemmas about valid contexts Now, we can prove two useful lemmas about valid contexts.

lemma *valid-monotonicity*[*elim*]:
assumes *a*: $\Gamma \subseteq \Gamma'$
and *b*: $x\#\Gamma'$
shows $(x,T_1)\#\Gamma \subseteq (x,T_1)\#\Gamma'$
using *a b* **by** *auto*

lemma *fresh-context*:
fixes Γ :: *Ctxt*
and *a* :: *name*
assumes $a\#\Gamma$
shows $\neg(\exists\,\tau{::}ty.\ (a,\tau) \in set\ \Gamma)$
using *assms*
by (*induct* Γ)
(*auto simp add*: *fresh-prod fresh-list-cons fresh-atm*)

lemma *type-unicity-in-context*:
assumes *a*: *valid* Γ
and *b*: $(x,T_1) \in set\ \Gamma$
and *c*: $(x,T_2) \in set\ \Gamma$
shows $T_1 = T_2$
using *a b c*
by (*induct* Γ)
(*auto dest*!: *fresh-context*)

$$\frac{\mathit{valid}\ \Gamma \qquad (x,\ T) \in \mathit{set}\ \Gamma}{\Gamma \vdash \mathit{Var}\ x : T}\text{T\_VAR} \qquad \frac{\Gamma \vdash e_1 : T_1 \to T_2 \qquad \Gamma \vdash e_2 : T_1}{\Gamma \vdash \mathit{App}\ e_1\ e_2 : T_2}\text{T\_APP}$$

$$\frac{x \# \Gamma \qquad (x,\ T_1) \# \Gamma \vdash t : T_2}{\Gamma \vdash \mathit{Lam}\ [x].t : T_1 \to T_2}\text{T\_LAM}$$

$$\frac{\mathit{valid}\ \Gamma}{\Gamma \vdash \mathit{Const}\ n : \mathit{TBase}}\text{T\_CONST} \qquad \frac{\mathit{valid}\ \Gamma}{\Gamma \vdash \mathit{Unit} : \mathit{TUnit}}\text{T\_UNIT}$$

Figure 1: Typing rules

4.2 Definition of the typing relation

Now, we can define the typing judgements for terms. The rules are given in figure 1.

Now,we generate the equivariance lemma and the strong induction principle and we derive the lemma about validity.

equivariance *typing*

nominal-inductive *typing*
  by (*simp-all add*: *abs-fresh*)

lemma *typing-implies-valid*:
  assumes *a*: $\Gamma \vdash t : T$
  shows *valid* Γ
  using *a* **by** (*induct*) (*auto*)

4.3 Inversion lemmas for the typing relation

We generate some inversion lemmas for the typing judgment and add them as elimination rules for the automatic tactics. During the generation of these lemmas, we need the injectivity properties of the constructor of the nominal datatypes. These are not added by default in the set of simplification rules to prevent unwanted simplifications in the rest of the development. In the future, the `inductive_cases` will be reworked to allow to use its own set of rules instead of the whole 'simpset'.

declare *trm.inject* [*simp add*]
declare *ty.inject* [*simp add*]

inductive-cases2 *t-Lam-elim-auto*[*elim*]: $\Gamma \vdash \mathit{Lam}\ [x].t : T$
inductive-cases2 *t-Var-elim-auto*[*elim*]: $\Gamma \vdash \mathit{Var}\ x : T$
inductive-cases2 *t-App-elim-auto*[*elim*]: $\Gamma \vdash \mathit{App}\ x\ y : T$
inductive-cases2 *t-Const-elim-auto*[*elim*]: $\Gamma \vdash \mathit{Const}\ n : T$
inductive-cases2 *t-Unit-elim-auto*[*elim*]: $\Gamma \vdash \mathit{Unit} : \mathit{TUnit}$
inductive-cases2 *t-Unit-elim-auto'*[*elim*]: $\Gamma \vdash s : \mathit{TUnit}$

declare *trm.inject* [*simp del*]
declare *ty.inject* [*simp del*]

$$App\ (Lam\ [x].t_1)\ t_2 \leadsto t_1[x{::=}t_2]\text{QAR\_BETA} \qquad \frac{t_1 \leadsto t_1{}'}{App\ t_1\ t_2 \leadsto App\ t_1{}'\ t_2}\text{QAR\_APP}$$

5 Definitional Equivalence

$$\frac{\Gamma \vdash t : T}{\Gamma \vdash t \equiv t : T}\text{Q\_REFL} \qquad \frac{\Gamma \vdash t \equiv s : T}{\Gamma \vdash s \equiv t : T}\text{Q\_SYMM}$$

$$\frac{\Gamma \vdash s \equiv t : T \qquad \Gamma \vdash t \equiv u : T}{\Gamma \vdash s \equiv u : T}\text{Q\_TRANS}$$

$$\frac{\Gamma \vdash s_1 \equiv t_1 : T_1 \rightarrow T_2 \qquad \Gamma \vdash s_2 \equiv t_2 : T_1}{\Gamma \vdash App\ s_1\ s_2 \equiv App\ t_1\ t_2 : T_2}\text{Q\_APP}$$

$$\frac{x \# \Gamma \qquad (x,\ T_1) \# \Gamma \vdash s_2 \equiv t_2 : T_2}{\Gamma \vdash Lam\ [x].s_2 \equiv Lam\ [x].t_2 : T_1 \rightarrow T_2}\text{Q\_ABS}$$

$$\frac{x \# (\Gamma,\ s_2,\ t_2) \qquad (x,\ T_1) \# \Gamma \vdash s_1 \equiv t_1 : T_2 \qquad \Gamma \vdash s_2 \equiv t_2 : T_1}{\Gamma \vdash App\ (Lam\ [x].s_1)\ s_2 \equiv t_1[x{::=}t_2] : T_2}\text{Q\_BETA}$$

$$\frac{x \# (\Gamma,\ s,\ t) \qquad (x,\ T_1) \# \Gamma \vdash App\ s\ (Var\ x) \equiv App\ t\ (Var\ x) : T_2}{\Gamma \vdash s \equiv t : T_1 \rightarrow T_2}\text{Q\_EXT}$$

$$\frac{\Gamma \vdash s : TUnit \qquad \Gamma \vdash t : TUnit}{\Gamma \vdash s \equiv t : TUnit}\text{Q\_UNIT}$$

It is now a tradition, we derive the lemma about validity, and we generate the equivariance lemma and the strong induction principle.

equivariance *def-equiv*

nominal-inductive *def-equiv*
  by (*simp-all add*: *abs-fresh fresh-subst''*)

lemma *def-equiv-implies-valid*:
  assumes *a*: $\Gamma \vdash t \equiv s : T$
  shows *valid* Γ
using *a* **by** (*induct*) (*auto elim*: *typing-implies-valid*)

6 Type-driven equivalence algorithm

We follow the original presentation. The algorithm is described using inference rules only.

6.1 Weak head reduction

6.1.1 Inversion lemma for weak head reduction

declare *trm.inject* [*simp add*]
declare *ty.inject* [*simp add*]

inductive-cases2 *whr-Gen*[*elim*]: $t \leadsto t'$

inductive-cases2 *whr-Lam*[*elim*]: *Lam* [*x*].*t* $\leadsto$ *t'*
inductive-cases2 *whr-App-Lam*[*elim*]: *App* (*Lam* [*x*].*t12*) *t2* $\leadsto$ *t*
inductive-cases2 *whr-Var*[*elim*]: *Var x* $\leadsto$ *t*
inductive-cases2 *whr-Const*[*elim*]: *Const n* $\leadsto$ *t*
inductive-cases2 *whr-App*[*elim*]: *App p q* $\leadsto$ *t*
inductive-cases2 *whr-Const-Right*[*elim*]: *t* $\leadsto$ *Const n*
inductive-cases2 *whr-Var-Right*[*elim*]: *t* $\leadsto$ *Var x*
inductive-cases2 *whr-App-Right*[*elim*]: *t* $\leadsto$ *App p q*

declare *trm.inject* [*simp del*]
declare *ty.inject* [*simp del*]

equivariance *whr-def*

6.2 Weak head normalization

abbreviation
nf :: *trm* ⇒ *bool* (- $\leadsto$| [*100*] *100*)
where
t$\leadsto$| ≡ ¬(∃ *u*. *t* $\leadsto$ *u*)

$$\frac{s \leadsto t \qquad t \Downarrow u}{s \Downarrow u}\text{QAN\_REDUCE} \qquad \frac{t \leadsto|}{t \Downarrow t}\text{QAN\_NORMAL}$$

declare *trm.inject*[*simp*]

inductive-cases2 *whn-inv-auto*[*elim*]: *t* ⇓ *t'*

declare *trm.inject*[*simp del*]

lemma *whn-eqvt*[*eqvt*]:
fixes *pi*::*name prm*
assumes *a*: *t* ⇓ *t'*
shows (*pi*•*t*) ⇓ (*pi*•*t'*)
using *a*
apply(*induct*)
apply(*rule QAN-Reduce*)
apply(*rule whr-def.eqvt*)
apply(*assumption*)+
apply(*rule QAN-Normal*)
apply(*auto*)
apply(*drule-tac pi=rev pi* **in** *whr-def.eqvt*)
apply(*perm-simp*)
done

lemma *red-unicity* :
assumes *a*: *x* $\leadsto$ *a*
and *b*: *x* $\leadsto$ *b*
shows *a*=*b*

```
  using a b
apply (induct arbitrary: b)
apply (erule whr-App-Lam)
apply (clarify)
apply (rule subst-fun-eq)
apply (simp)
apply (force)
apply (erule whr-App)
apply (blast)+
done
```

```
lemma nf-unicity :
  assumes x ⇓ a and x ⇓ b
  shows a=b
  using assms
proof (induct arbitrary: b)
  case (QAN-Reduce x t a b)
  have h:x ⇝ t t ⇓ a by fact
  have ih:⋀b. t ⇓ b ⟹ a = b by fact
  have x ⇓ b by fact
  then obtain t′ where x ⇝ t′ and hl:t′ ⇓ b using h by auto
  then have t=t′ using h red-unicity by auto
  then show a=b using ih hl by auto
qed (auto)
```

6.3 Algorithmic term equivalence and algorithmic path equivalence

$$\frac{s \Downarrow p \qquad t \Downarrow q \qquad \Gamma \vdash p \leftrightarrow q : \mathit{TBase}}{\Gamma \vdash s \Leftrightarrow t : \mathit{TBase}}\text{QAT\_BASE}$$

$$\frac{x \# (\Gamma,\ s,\ t) \qquad (x,\ T_1) \# \Gamma \vdash \mathit{App}\ s\ (\mathit{Var}\ x) \Leftrightarrow \mathit{App}\ t\ (\mathit{Var}\ x) : T_2}{\Gamma \vdash s \Leftrightarrow t : T_1 \rightarrow T_2}\text{QAT\_ARROW}$$

$$\frac{\mathit{valid}\ \Gamma}{\Gamma \vdash s \Leftrightarrow t : \mathit{TUnit}}\text{QAT\_ONE}$$

$$\frac{\mathit{valid}\ \Gamma \qquad (x,\ T) \in \mathit{set}\ \Gamma}{\Gamma \vdash \mathit{Var}\ x \leftrightarrow \mathit{Var}\ x : T}\text{QAP\_VAR}$$

$$\frac{\Gamma \vdash p \leftrightarrow q : T_1 \rightarrow T_2 \qquad \Gamma \vdash s \Leftrightarrow t : T_1}{\Gamma \vdash \mathit{App}\ p\ s \leftrightarrow \mathit{App}\ q\ t : T_2}\text{QAP\_APP}$$

$$\frac{\mathit{valid}\ \Gamma}{\Gamma \vdash \mathit{Const}\ n \leftrightarrow \mathit{Const}\ n : \mathit{TBase}}\text{QAP\_CONST}$$

Again we generate the equivariance lemma and the strong induction principle.

```
equivariance alg-equiv
```

```
nominal-inductive alg-equiv
  avoids QAT-Arrow: x
  by simp-all
```

thm *alg-equiv-alg-path-equiv.strong-induct*

6.3.1 Inversion lemmas for algorithmic term and path equivalences

declare *trm.inject* [*simp add*]
declare *ty.inject* [*simp add*]

inductive-cases2 *alg-equiv-Base-inv-auto*[*elim*]: $\Gamma \vdash s \Leftrightarrow t : TBase$
inductive-cases2 *alg-equiv-Arrow-inv-auto*[*elim*]: $\Gamma \vdash s \Leftrightarrow t : T_1 \to T_2$

inductive-cases2 *alg-path-equiv-Base-inv-auto*[*elim*]: $\Gamma \vdash s \leftrightarrow t : TBase$
inductive-cases2 *alg-path-equiv-Unit-inv-auto*[*elim*]: $\Gamma \vdash s \leftrightarrow t : TUnit$
inductive-cases2 *alg-path-equiv-Arrow-inv-auto*[*elim*]: $\Gamma \vdash s \leftrightarrow t : T_1 \to T_2$

inductive-cases2 *alg-path-equiv-Var-left-inv-auto*[*elim*]: $\Gamma \vdash Var\ x \leftrightarrow t : T$
inductive-cases2 *alg-path-equiv-Var-left-inv-auto'*[*elim*]: $\Gamma \vdash Var\ x \leftrightarrow t : T'$
inductive-cases2 *alg-path-equiv-Var-right-inv-auto*[*elim*]: $\Gamma \vdash s \leftrightarrow Var\ x : T$
inductive-cases2 *alg-path-equiv-Var-right-inv-auto'*[*elim*]: $\Gamma \vdash s \leftrightarrow Var\ x : T'$
inductive-cases2 *alg-path-equiv-Const-left-inv-auto*[*elim*]: $\Gamma \vdash Const\ n \leftrightarrow t : T$
inductive-cases2 *alg-path-equiv-Const-right-inv-auto*[*elim*]: $\Gamma \vdash s \leftrightarrow Const\ n : T$
inductive-cases2 *alg-path-equiv-App-left-inv-auto*[*elim*]: $\Gamma \vdash App\ p\ s \leftrightarrow t : T$
inductive-cases2 *alg-path-equiv-App-right-inv-auto*[*elim*]: $\Gamma \vdash s \leftrightarrow App\ q\ t : T$
inductive-cases2 *alg-path-equiv-Lam-left-inv-auto*[*elim*]: $\Gamma \vdash Lam[x].s \leftrightarrow t : T$
inductive-cases2 *alg-path-equiv-Lam-right-inv-auto*[*elim*]: $\Gamma \vdash t \leftrightarrow Lam[x].s : T$

declare *trm.inject* [*simp del*]
declare *ty.inject* [*simp del*]

lemma *Q-Arrow-strong-inversion*:
  assumes *fs*: $x\#\Gamma\ x\#t\ x\#u$
  and *h*: $\Gamma \vdash t \Leftrightarrow u : T_1 \to T_2$
  shows $(x,T_1)\#\Gamma \vdash App\ t\ (Var\ x) \Leftrightarrow App\ u\ (Var\ x) : T_2$
proof −
  obtain y **where** *fs2*: $y\#(\Gamma,t,u)$ **and** $(y,T_1)\#\Gamma \vdash App\ t\ (Var\ y) \Leftrightarrow App\ u\ (Var\ y) : T_2$
    using *h* **by** *auto*
  then have $([(x,y)]\cdot((y,T_1)\#\Gamma)) \vdash [(x,y)]\cdot\ App\ t\ (Var\ y) \Leftrightarrow [(x,y)]\cdot\ App\ u\ (Var\ y) : T_2$
    using *alg-equiv.eqvt*[*simplified*] **by** *blast*
  then show *?thesis* **using** *fs fs2* **by** (*perm-simp*)
qed

For the `algorithmic_transitivity` lemma we need a unicity property. But one has to be cautious, because this unicity property is true only for algorithmic path. Indeed the following lemma is **false**:

$$[\![\Gamma \vdash s \Leftrightarrow t : T; \Gamma \vdash s \Leftrightarrow u : T']\!] \Longrightarrow T = T'$$

Here is the counter example :

$$\Gamma \vdash Const\ n \Leftrightarrow Const\ n : Tbase \text{ and } \Gamma \vdash Const\ n \Leftrightarrow Const\ n : TUnit$$

lemma *algorithmic-path-type-unicity*:
shows $\Gamma \vdash s \leftrightarrow t : T \Longrightarrow \Gamma \vdash s \leftrightarrow u : T' \Longrightarrow T = T'$
proof (*induct arbitrary*: u T'
rule: *alg-equiv-alg-path-equiv.inducts*(*2*) [*of* - - - - - %a b c d . *True*])
case (*QAP-Var* Γ x T u T')
have $\Gamma \vdash$ *Var* $x \leftrightarrow u : T'$ **by** *fact*
then have u=*Var* x **and** $(x,T') \in$ *set* Γ **by** *auto*
moreover have *valid* Γ $(x,T) \in$ *set* Γ **by** *fact*
ultimately show T=T' **using** *type-unicity-in-context* **by** *auto*
next
case (*QAP-App* Γ p q T_1 T_2 s t u T_2')
have *ih*:$\bigwedge u\ T.\ \Gamma \vdash p \leftrightarrow u : T \Longrightarrow T_1 \rightarrow T_2 = T$ **by** *fact*
have $\Gamma \vdash$ *App* p $s \leftrightarrow u : T_2'$ **by** *fact*
then obtain r t T_1' **where** u = *App* r t $\Gamma \vdash p \leftrightarrow r : T_1' \rightarrow T_2'$ **by** *auto*
then have $T_1 \rightarrow T_2 = T_1' \rightarrow T_2'$ **by** *auto*
then show T_2=T_2' **using** *ty.inject* **by** *auto*
qed (*auto*)

lemma *alg-path-equiv-implies-valid*:
shows $\Gamma \vdash s \Leftrightarrow t : T \Longrightarrow$ *valid* Γ
and $\Gamma \vdash s \leftrightarrow t : T \Longrightarrow$ *valid* Γ
by (*induct rule* : *alg-equiv-alg-path-equiv.inducts*, *auto*)

lemma *algorithmic-symmetry*:
shows $\Gamma \vdash s \Leftrightarrow t : T \Longrightarrow \Gamma \vdash t \Leftrightarrow s : T$
and $\Gamma \vdash s \leftrightarrow t : T \Longrightarrow \Gamma \vdash t \leftrightarrow s : T$
by (*induct rule*: *alg-equiv-alg-path-equiv.inducts*)
(*auto simp add*: *fresh-prod*)

lemma *algorithmic-transitivity*:
shows $\Gamma \vdash s \Leftrightarrow t : T \Longrightarrow \Gamma \vdash t \Leftrightarrow u : T \Longrightarrow \Gamma \vdash s \Leftrightarrow u : T$
and $\Gamma \vdash s \leftrightarrow t : T \Longrightarrow \Gamma \vdash t \leftrightarrow u : T \Longrightarrow \Gamma \vdash s \leftrightarrow u : T$
proof (*nominal-induct* Γ s t T **and** Γ s t T *avoiding*: u *rule*: *alg-equiv-alg-path-equiv.strong-inducts*)
case (*QAT-Base* s p t q Γ u)
have $\Gamma \vdash t \Leftrightarrow u$: *TBase* **by** *fact*
then obtain r' q' **where** *b1*: $t \Downarrow q'$ **and** *b2*: $u \Downarrow r'$ **and** *b3*: $\Gamma \vdash q' \leftrightarrow r'$: *TBase* **by** *auto*
have *ih*: $\Gamma \vdash q \leftrightarrow r'$: *TBase* $\Longrightarrow \Gamma \vdash p \leftrightarrow r'$: *TBase* **by** *fact*
have $t \Downarrow q$ **by** *fact*
with *b1* **have** *eq*: q=q' **by** (*simp add*: *nf-unicity*)
with *ih b3* **have** $\Gamma \vdash p \leftrightarrow r'$: *TBase* **by** *simp*
moreover
have $s \Downarrow p$ **by** *fact*
ultimately show $\Gamma \vdash s \Leftrightarrow u$: *TBase* **using** *b2* **by** *auto*
next
case (*QAT-Arrow* x Γ s t T_1 T_2 u)
have *ih*:$(x,T_1)\#\Gamma \vdash$ *App* t (*Var* x) $\Leftrightarrow$ *App* u (*Var* x) : T_2
$\Longrightarrow (x,T_1)\#\Gamma \vdash$ *App* s (*Var* x) $\Leftrightarrow$ *App* u (*Var* x) : T_2 **by** *fact*
have *fs*: $x\#\Gamma$ $x\#s$ $x\#t$ $x\#u$ **by** *fact*
have $\Gamma \vdash t \Leftrightarrow u : T_1 \rightarrow T_2$ **by** *fact*
then have $(x,T_1)\#\Gamma \vdash$ *App* t (*Var* x) $\Leftrightarrow$ *App* u (*Var* x) : T_2 **using** *fs*
by (*simp add*: *Q-Arrow-strong-inversion*)
with *ih* **have** $(x,T_1)\#\Gamma \vdash$ *App* s (*Var* x) $\Leftrightarrow$ *App* u (*Var* x) : T_2 **by** *simp*
then show $\Gamma \vdash s \Leftrightarrow u : T_1 \rightarrow T_2$ **using** *fs* **by** (*auto simp add*: *fresh-prod*)

next
  case $(QAP\text{-}App\ \Gamma\ p\ q\ T_1\ T_2\ s\ t\ u)$
  have $\Gamma \vdash App\ q\ t \leftrightarrow u : T_2$ **by** *fact*
  then obtain $r\ T_1{}'\ v$ **where** *ha*: $\Gamma \vdash q \leftrightarrow r : T_1{}' \rightarrow T_2$ **and** *hb*: $\Gamma \vdash t \Leftrightarrow v : T_1{}'$ **and** *eq*: $u = App\ r\ v$
    by *auto*
  have *ih1*: $\Gamma \vdash q \leftrightarrow r : T_1 \rightarrow T_2 \Longrightarrow \Gamma \vdash p \leftrightarrow r : T_1 \rightarrow T_2$ **by** *fact*
  have *ih2*:$\Gamma \vdash t \Leftrightarrow v : T_1 \Longrightarrow \Gamma \vdash s \Leftrightarrow v : T_1$ **by** *fact*
  have $\Gamma \vdash p \leftrightarrow q : T_1 \rightarrow T_2$ **by** *fact*
  then have $\Gamma \vdash q \leftrightarrow p : T_1 \rightarrow T_2$ **by** (*simp add*: *algorithmic-symmetry*)
  with *ha* **have** $T_1{}' \rightarrow T_2 = T_1 \rightarrow T_2$ **using** *algorithmic-path-type-unicity* **by** *simp*
  then have $T_1{}' = T_1$ **by** (*simp add*: *ty.inject*)
  then have $\Gamma \vdash s \Leftrightarrow v : T_1\ \Gamma \vdash p \leftrightarrow r : T_1 \rightarrow T_2$ **using** *ih1 ih2 ha hb* **by** *auto*
  then show $\Gamma \vdash App\ p\ s \leftrightarrow u : T_2$ **using** *eq* **by** *auto*
qed (*auto*)

lemma *algorithmic-weak-head-closure*:
  shows $\Gamma \vdash s \Leftrightarrow t : T \Longrightarrow s' \rightsquigarrow s \Longrightarrow t' \rightsquigarrow t \Longrightarrow \Gamma \vdash s' \Leftrightarrow t' : T$
apply (*nominal-induct* $\Gamma\ s\ t\ T$ *avoiding*: $s'\ t'$
    rule: *alg-equiv-alg-path-equiv.strong-inducts*(*1*) [*of* - - - - %*a b c d e*. *True*])
apply(*auto intro*!: *QAT-Arrow*)
done

lemma *algorithmic-monotonicity*:
  shows $\Gamma \vdash s \Leftrightarrow t : T \Longrightarrow \Gamma \subseteq \Gamma' \Longrightarrow valid\ \Gamma' \Longrightarrow \Gamma' \vdash s \Leftrightarrow t : T$
  and $\Gamma \vdash s \leftrightarrow t : T \Longrightarrow \Gamma \subseteq \Gamma' \Longrightarrow valid\ \Gamma' \Longrightarrow \Gamma' \vdash s \leftrightarrow t : T$
proof (*nominal-induct* $\Gamma\ s\ t\ T$ **and** $\Gamma\ s\ t\ T$ *avoiding*: Γ' *rule*: *alg-equiv-alg-path-equiv.strong-inducts*)
 case $(QAT\text{-}Arrow\ x\ \Gamma\ s\ t\ T_1\ T_2\ \Gamma')$
  have *fs*:$x\#\Gamma\ x\#s\ x\#t\ x\#\Gamma'$ **by** *fact*
  have *h2*:$\Gamma \subseteq \Gamma'$ **by** *fact*
  have *ih*:$\bigwedge\Gamma'.\ [\![(x,T_1)\#\Gamma \subseteq \Gamma';\ valid\ \Gamma']\!] \Longrightarrow \Gamma' \vdash App\ s\ (Var\ x) \Leftrightarrow App\ t\ (Var\ x) : T_2$ **by** *fact*
  have $valid\ \Gamma'$ **by** *fact*
  then have $valid\ ((x,T_1)\#\Gamma')$ **using** *fs* **by** *auto*
  moreover
  have *sub*: $(x,T_1)\#\Gamma \subseteq (x,T_1)\#\Gamma'$ **using** *h2* **by** *auto*
  ultimately have $(x,T_1)\#\Gamma' \vdash App\ s\ (Var\ x) \Leftrightarrow App\ t\ (Var\ x) : T_2$ **using** *ih* **by** *simp*
  then show $\Gamma' \vdash s \Leftrightarrow t : T_1 \rightarrow T_2$ **using** *fs* **by** (*auto simp add*: *fresh-prod*)
qed (*auto*)

lemma *path-equiv-implies-nf*:
  assumes $\Gamma \vdash s \leftrightarrow t : T$
  shows $s \rightsquigarrow|$ **and** $t \rightsquigarrow|$
using *assms*
by (*induct rule*: *alg-equiv-alg-path-equiv.inducts*(*2*)) (*simp*, *auto*)

6.4 Definition of the logical relation

We define the logical equivalence as a function. Note that here we can not use an inductive definition because of the negative occurence in the arrow case.

function *log-equiv* :: (*Ctxt* ⇒ *trm* ⇒ *trm* ⇒ *ty* ⇒ *bool*) (- ⊢ - *is* - : - [*60*,*60*,*60*,*60*] *60*)

```
where
  Γ ⊢ s is t : TUnit = True
| Γ ⊢ s is t : TBase = Γ ⊢ s ⇔ t : TBase
| Γ ⊢ s is t : (T1 → T2) =
   (∀Γ' s' t'. Γ⊆Γ' ⟶ valid Γ' ⟶ Γ' ⊢ s' is t' : T1 ⟶ (Γ' ⊢ (App s s') is (App t t') : T2))
apply (auto simp add: ty.inject)
apply (subgoal-tac (∃ T1 T2. b=T1 → T2) ∨ b=TUnit ∨ b=TBase )
apply (force)
apply (rule ty-cases)
done

termination
apply(relation measure (λ(-,-,-,T). size T))
apply(auto)
done
```

Monotonicity of the logical equivalence relation.

```
lemma logical-monotonicity :
 assumes a1: Γ ⊢ s is t : T
 and     a2: Γ ⊆ Γ'
 and     a3: valid Γ'
 shows Γ' ⊢ s is t : T
using a1 a2 a3
proof (induct arbitrary: Γ' rule: log-equiv.induct)
  case (2 Γ s t Γ')
  then show Γ' ⊢ s is t : TBase using algorithmic-monotonicity by auto
next
  case (3 Γ s t T1 T2 Γ')
  have Γ ⊢ s is t : T1→T2
  and  Γ ⊆ Γ'
  and  valid Γ' by fact
  then show Γ' ⊢ s is t : T1→T2 by simp
qed (auto)

lemma main-lemma:
  shows Γ ⊢ s is t : T ⟹ valid Γ ⟹ Γ ⊢ s ⇔ t : T
    and Γ ⊢ p ↔ q : T ⟹ Γ ⊢ p is q : T
proof (nominal-induct T arbitrary: Γ s t p q rule: ty.induct)
  case (Arrow T1 T2)
  {
    case (1 Γ s t)
    have ih1:⋀Γ s t. ⟦Γ ⊢ s is t : T2; valid Γ⟧ ⟹ Γ ⊢ s ⇔ t : T2 by fact
    have ih2:⋀Γ s t. Γ ⊢ s ↔ t : T1 ⟹ Γ ⊢ s is t : T1 by fact
    have h:Γ ⊢ s is t : T1→T2 by fact
    obtain x::name where fs:x♯(Γ,s,t) by (erule exists-fresh[OF fs-name1])
    have valid Γ by fact
    then have v: valid ((x,T1)#Γ) using fs by auto
    then have (x,T1)#Γ ⊢ Var x ↔ Var x : T1 by auto
    then have (x,T1)#Γ ⊢ Var x is Var x : T1 using ih2 by auto
    then have (x,T1)#Γ ⊢ App s (Var x) is App t (Var x) : T2 using h v by auto
    then have (x,T1)#Γ ⊢ App s (Var x) ⇔ App t (Var x) : T2 using ih1 v by auto
    then show Γ ⊢ s ⇔ t : T1→T2 using fs by (auto simp add: fresh-prod)
  next
```

case (2 Γ p q)
have *h*: $\Gamma \vdash p \leftrightarrow q : T_1 \rightarrow T_2$ **by** *fact*
have *ih1*:$\bigwedge\Gamma\ s\ t.\ \Gamma \vdash s \leftrightarrow t : T_2 \Longrightarrow \Gamma \vdash s\ is\ t : T_2$ **by** *fact*
have *ih2*:$\bigwedge\Gamma\ s\ t.\ [\![\Gamma \vdash s\ is\ t : T_1;\ valid\ \Gamma]\!] \Longrightarrow \Gamma \vdash s \Leftrightarrow t : T_1$ **by** *fact*
{
fix Γ' s t
assume $\Gamma \subseteq \Gamma'$ **and** *hl*:$\Gamma' \vdash s\ is\ t : T_1$ **and** *hk*: *valid* Γ'
then have $\Gamma' \vdash p \leftrightarrow q : T_1 \rightarrow T_2$ **using** *h algorithmic-monotonicity* **by** *auto*
moreover have $\Gamma' \vdash s \Leftrightarrow t : T_1$ **using** *ih2 hl hk* **by** *auto*
ultimately have $\Gamma' \vdash App\ p\ s \leftrightarrow App\ q\ t : T_2$ **by** *auto*
then have $\Gamma' \vdash App\ p\ s\ is\ App\ q\ t : T_2$ **using** *ih1* **by** *auto*
}
then show $\Gamma \vdash p\ is\ q : T_1 \rightarrow T_2$ **by** *simp*
}
next
case *TBase*
{ case *2*
have *h*:$\Gamma \vdash s \leftrightarrow t : TBase$ **by** *fact*
then have $s \rightsquigarrow|$ **and** $t \rightsquigarrow|$ **using** *path-equiv-implies-nf* **by** *auto*
then have $s \Downarrow s$ **and** $t \Downarrow t$ **by** *auto*
then have $\Gamma \vdash s \Leftrightarrow t : TBase$ **using** *h* **by** *auto*
then show $\Gamma \vdash s\ is\ t : TBase$ **by** *auto*
}
qed (*auto elim*: *alg-path-equiv-implies-valid*)

corollary *corollary-main*:
assumes *a*: $\Gamma \vdash s \leftrightarrow t : T$
shows $\Gamma \vdash s \Leftrightarrow t : T$
using *a main-lemma alg-path-equiv-implies-valid* **by** *blast*

lemma *logical-symmetry*:
assumes *a*: $\Gamma \vdash s\ is\ t : T$
shows $\Gamma \vdash t\ is\ s : T$
using *a*
by (*nominal-induct arbitrary*: Γ s t *rule*: *ty.induct*)
(*auto simp add*: *algorithmic-symmetry*)

lemma *logical-transitivity*:
assumes $\Gamma \vdash s\ is\ t : T$ $\Gamma \vdash t\ is\ u : T$
shows $\Gamma \vdash s\ is\ u : T$
using *assms*
proof (*nominal-induct arbitrary*: Γ s t u *rule*:*ty.induct*)
case *TBase*
then show $\Gamma \vdash s\ is\ u : TBase$ **by** (*auto elim*: *algorithmic-transitivity*)
next
case (*Arrow* T_1 T_2 Γ s t u)
have *h1*:$\Gamma \vdash s\ is\ t : T_1 \rightarrow T_2$ **by** *fact*
have *h2*:$\Gamma \vdash t\ is\ u : T_1 \rightarrow T_2$ **by** *fact*
have *ih1*:$\bigwedge\Gamma\ s\ t\ u.\ [\![\Gamma \vdash s\ is\ t : T_1;\ \Gamma \vdash t\ is\ u : T_1]\!] \Longrightarrow \Gamma \vdash s\ is\ u : T_1$ **by** *fact*
have *ih2*:$\bigwedge\Gamma\ s\ t\ u.\ [\![\Gamma \vdash s\ is\ t : T_2;\ \Gamma \vdash t\ is\ u : T_2]\!] \Longrightarrow \Gamma \vdash s\ is\ u : T_2$ **by** *fact*
{
fix Γ' s' u'
assume *hsub*:$\Gamma \subseteq \Gamma'$ **and** *hl*:$\Gamma' \vdash s'\ is\ u' : T_1$ **and** *hk*: *valid* Γ'

then have $\Gamma' \vdash u'$ *is* $s' : T_1$ **using** *logical-symmetry* **by** *blast*
then have $\Gamma' \vdash u'$ *is* $u' : T_1$ **using** *ih1 hl* **by** *blast*
then have $\Gamma' \vdash$ *App t u'* *is* *App u u'* $: T_2$ **using** *h2 hsub hk* **by** *auto*
moreover have $\Gamma' \vdash$ *App s s'* *is* *App t u'* $: T_2$ **using** *h1 hsub hl hk* **by** *auto*
ultimately have $\Gamma' \vdash$ *App s s'* *is* *App u u'* $: T_2$ **using** *ih2* **by** *blast*
}
then show $\Gamma \vdash s$ *is* $u : T_1 \rightarrow T_2$ **by** *auto*
qed (*auto*)

To simplify the formal proof, here we derive two lemmas which are weaker than the lemma in the paper version. We omit the reflexive and transitive closure of the relation $s' \rightsquigarrow s$ in the assumptions.

lemma *logical-weak-head-closure*:
assumes *a*: $\Gamma \vdash s$ *is* $t : T$
and *b*: $s' \rightsquigarrow s$
and *c*: $t' \rightsquigarrow t$
shows $\Gamma \vdash s'$ *is* $t' : T$
using *a b c algorithmic-weak-head-closure*
by (*nominal-induct arbitrary*: Γ *s t s' t'* *rule*: *ty.induct*)
(*auto*, *blast*)

lemma *logical-weak-head-closure'*:
assumes $\Gamma \vdash s$ *is* $t : T$ **and** $s' \rightsquigarrow s$
shows $\Gamma \vdash s'$ *is* $t : T$
using *assms*
proof (*nominal-induct arbitrary*: Γ *s t s'* *rule*: *ty.induct*)
case (*TBase* Γ *s t s'*)
then show *?case* **by** *force*
next
case (*TUnit* Γ *s t s'*)
then show *?case* **by** *auto*
next
case (*Arrow* T_1 T_2 Γ *s t s'*)
have *h1*: $s' \rightsquigarrow s$ **by** *fact*
have *ih*: $\bigwedge \Gamma\ s\ t\ s'.\ [\![\Gamma \vdash s\ is\ t : T_2;\ s' \rightsquigarrow s]\!] \Longrightarrow \Gamma \vdash s'\ is\ t : T_2$ **by** *fact*
have *h2*: $\Gamma \vdash s$ *is* $t : T_1 \rightarrow T_2$ **by** *fact*
then
have *hb*: $\forall \Gamma'\ s'\ t'.\ \Gamma \subseteq \Gamma' \longrightarrow valid\ \Gamma' \longrightarrow \Gamma' \vdash s'\ is\ t' : T_1 \longrightarrow (\Gamma' \vdash (App\ s\ s')\ is\ (App\ t\ t') : T_2)$
by *auto*
{
fix Γ' s_2 t_2
assume $\Gamma \subseteq \Gamma'$ **and** $\Gamma' \vdash s_2$ *is* $t_2 : T_1$ **and** *valid* Γ'
then have $\Gamma' \vdash (App\ s\ s_2)$ *is* $(App\ t\ t_2) : T_2$ **using** *hb* **by** *auto*
moreover have $(App\ s'\ s_2) \rightsquigarrow (App\ s\ s_2)$ **using** *h1* **by** *auto*
ultimately have $\Gamma' \vdash App\ s'\ s_2$ *is* $App\ t\ t_2 : T_2$ **using** *ih* **by** *auto*
}
then show $\Gamma \vdash s'$ *is* $t : T_1 \rightarrow T_2$ **by** *auto*
qed

abbreviation
log-equiv-for-psubsts :: *Ctxt* ⇒ *Subst* ⇒ *Subst* ⇒ *Ctxt* ⇒ *bool* (- ⊢ - *is* - *over* - [*60*,*60*] *60*)
where

```
Γ′ ⊢ θ is θ′ over Γ ≡ ∀ x T. (x,T) ∈ set Γ ⟶ Γ′ ⊢ θ<Var x> is θ′<Var x> : T
```

Now, we can derive that the logical equivalence is almost reflexive.

```
lemma logical-pseudo-reflexivity:
  assumes Γ′ ⊢ t is s over Γ
  shows Γ′ ⊢ s is s over Γ
proof −
  have Γ′ ⊢ t is s over Γ by fact
  moreover then have Γ′ ⊢ s is t over Γ using logical-symmetry by blast
  ultimately show Γ′ ⊢ s is s over Γ using logical-transitivity by blast
qed

lemma logical-subst-monotonicity :
  assumes a: Γ′ ⊢ s is t over Γ
  and     b: Γ′ ⊆ Γ′′
  and     c: valid Γ′′
  shows Γ′′ ⊢ s is t over Γ
using a b c logical-monotonicity by blast

lemma equiv-subst-ext :
  assumes h1: Γ′ ⊢ θ is θ′ over Γ
  and     h2: Γ′ ⊢ s is t : T
  and     fs: x♯Γ
  shows Γ′ ⊢ (x,s)#θ is (x,t)#θ′ over (x,T)#Γ
using assms
proof −
  {
    fix y U
    assume (y,U) ∈ set ((x,T)#Γ)
    moreover
    {
      assume (y,U) ∈ set [(x,T)]
      then have Γ′ ⊢ (x,s)#θ<Var y> is (x,t)#θ′<Var y> : U by auto
    }
    moreover
    {
      assume hl:(y,U) ∈ set Γ
      then have ¬ y♯Γ by (induct Γ) (auto simp add: fresh-list-cons fresh-atm fresh-prod)
      then have hf:x♯ Var y using fs by (auto simp add: fresh-atm)
     then have (x,s)#θ<Var y> = θ<Var y> (x,t)#θ′<Var y> = θ′<Var y> using fresh-psubst-simp
by blast+
      moreover have  Γ′ ⊢ θ<Var y> is θ′<Var y> : U using h1 hl by auto
      ultimately have Γ′ ⊢ (x,s)#θ<Var y> is (x,t)#θ′<Var y> : U by auto
    }
    ultimately have Γ′ ⊢ (x,s)#θ<Var y> is (x,t)#θ′<Var y> : U by auto
  }
  then show Γ′ ⊢ (x,s)#θ is (x,t)#θ′ over (x,T)#Γ by auto
qed
```

6.5 Fundamental theorems

theorem *fundamental-theorem-1*:
  assumes *h1*: $\Gamma \vdash t : T$
  and *h2*: $\Gamma' \vdash \theta$ *is* θ' *over* Γ
  and *h3*: *valid* Γ'
  shows $\Gamma' \vdash \theta<t>$ *is* $\theta'<t> : T$
using *h1 h2 h3*
proof (*nominal-induct* Γ t T *avoiding*: Γ' θ θ' *rule*: *typing.strong-induct*)
  case (*t-Lam* x Γ T_1 t_2 T_2 Γ' θ θ')
  have *fs*:$x\#\theta$ $x\#\theta'$ $x\#\Gamma$ **by** *fact*
  have *h*:$\Gamma' \vdash \theta$ *is* θ' *over* Γ **by** *fact*
  have *ih*:$\bigwedge \Gamma'\ \theta\ \theta'.$ $[\![\Gamma' \vdash \theta$ *is* θ' *over* $(x,T_1)\#\Gamma$; *valid* $\Gamma']\!] \Longrightarrow \Gamma' \vdash \theta<t_2>$ *is* $\theta'<t_2> : T_2$ **by** *fact*
  {
    fix Γ'' s' t'
    assume $\Gamma' \subseteq \Gamma''$ **and** *hl*:$\Gamma'' \vdash s'$ *is* $t' : T_1$ **and** *v*: *valid* Γ''
    then have $\Gamma'' \vdash \theta$ *is* θ' *over* Γ **using** *logical-subst-monotonicity h* **by** *blast*
    then have $\Gamma'' \vdash (x,s')\#\theta$ *is* $(x,t')\#\theta'$ *over* $(x,T_1)\#\Gamma$ **using** *equiv-subst-ext hl fs* **by** *blast*
    then have $\Gamma'' \vdash (x,s')\#\theta<t_2>$ *is* $(x,t')\#\theta'<t_2> : T_2$ **using** *ih v* **by** *auto*
    then have $\Gamma''\vdash\theta<t_2>[x::=s']$ *is* $\theta'<t_2>[x::=t'] : T_2$ **using** *psubst-subst-psubst fs* **by** *simp*
    moreover have *App* (*Lam* $[x].\theta<t_2>$) $s' \rightsquigarrow \theta<t_2>[x::=s']$ **by** *auto*
    moreover have *App* (*Lam* $[x].\theta'<t_2>$) $t' \rightsquigarrow \theta'<t_2>[x::=t']$ **by** *auto*
    ultimately have $\Gamma''\vdash$ *App* (*Lam* $[x].\theta<t_2>$) s' *is App* (*Lam* $[x].\theta'<t_2>$) $t' : T_2$
      using *logical-weak-head-closure* **by** *auto*
  }
  then show $\Gamma' \vdash \theta<Lam\ [x].t_2>$ *is* $\theta'<Lam\ [x].t_2> : T_1 \rightarrow T_2$ **using** *fs* **by** *simp*
qed (*auto*)

theorem *fundamental-theorem-2*:
  assumes *h1*: $\Gamma \vdash s \equiv t : T$
  and *h2*: $\Gamma' \vdash \theta$ *is* θ' *over* Γ
  and *h3*: *valid* Γ'
  shows $\Gamma' \vdash \theta<s>$ *is* $\theta'<t> : T$
using *h1 h2 h3*
proof (*nominal-induct* Γ s t T *avoiding*: Γ' θ θ' *rule*: *def-equiv.strong-induct*)
  case (*Q-Refl* Γ t T Γ' θ θ')
  have $\Gamma \vdash t : T$
  and *valid* Γ' **by** *fact*
  moreover
  have $\Gamma' \vdash \theta$ *is* θ' *over* Γ **by** *fact*
  ultimately show $\Gamma' \vdash \theta<t>$ *is* $\theta'<t> : T$ **using** *fundamental-theorem-1* **by** *blast*
next
  case (*Q-Symm* Γ t s T Γ' θ θ')
  have $\Gamma' \vdash \theta$ *is* θ' *over* Γ
  and *valid* Γ' **by** *fact*
  moreover
  have *ih*: $\bigwedge \Gamma'\ \theta\ \theta'.$ $[\![\Gamma' \vdash \theta$ *is* θ' *over* Γ; *valid* $\Gamma']\!] \Longrightarrow \Gamma' \vdash \theta<t>$ *is* $\theta'<s> : T$ **by** *fact*
  ultimately show $\Gamma' \vdash \theta<s>$ *is* $\theta'<t> : T$ **using** *logical-symmetry* **by** *blast*
next
  case (*Q-Trans* Γ s t T u Γ' θ θ')
  have *ih1*: $\bigwedge \Gamma'\ \theta\ \theta'.$ $[\![\Gamma' \vdash \theta$ *is* θ' *over* Γ; *valid* $\Gamma']\!] \Longrightarrow \Gamma' \vdash \theta<s>$ *is* $\theta'<t> : T$ **by** *fact*
  have *ih2*: $\bigwedge \Gamma'\ \theta\ \theta'.$ $[\![\Gamma' \vdash \theta$ *is* θ' *over* Γ; *valid* $\Gamma']\!] \Longrightarrow \Gamma' \vdash \theta<t>$ *is* $\theta'<u> : T$ **by** *fact*
  have *h*: $\Gamma' \vdash \theta$ *is* θ' *over* Γ

and v: *valid* Γ' **by** *fact*
then have $\Gamma' \vdash \theta'$ *is* θ' *over* Γ **using** *logical-pseudo-reflexivity* **by** *auto*
then have $\Gamma' \vdash \theta'{<}t{>}$ *is* $\theta'{<}u{>} : T$ **using** *ih2 v* **by** *auto*
moreover have $\Gamma' \vdash \theta{<}s{>}$ *is* $\theta'{<}t{>} : T$ **using** *ih1 h v* **by** *auto*
ultimately show $\Gamma' \vdash \theta{<}s{>}$ *is* $\theta'{<}u{>} : T$ **using** *logical-transitivity* **by** *blast*
next
case $(\mathit{Q\text{-}Abs}\ x\ \Gamma\ T_1\ s_2\ t_2\ T_2\ \Gamma'\ \theta\ \theta')$
have $\mathit{fs}{:}x\#\Gamma$ **by** *fact*
have $\mathit{fs2}$: $x\#\theta\ x\#\theta'$ **by** *fact*
have $\mathit{h2}$: $\Gamma' \vdash \theta$ *is* θ' *over* Γ
and $\mathit{h3}$: *valid* Γ' **by** *fact*
have $\mathit{ih}{:}\bigwedge\Gamma'\ \theta\ \theta'.\ [\![\Gamma' \vdash \theta\ \mathit{is}\ \theta'\ \mathit{over}\ (x,T_1)\#\Gamma;\ \mathit{valid}\ \Gamma']\!] \Longrightarrow \Gamma' \vdash \theta{<}s_2{>}\ \mathit{is}\ \theta'{<}t_2{>} : T_2$ **by** *fact*
{
fix $\Gamma''\ s'\ t'$
assume $\Gamma' \subseteq \Gamma''$ **and** $\mathit{hl}{:}\Gamma'' \vdash s'$ *is* $t' : T_1$ **and** hk: *valid* Γ''
then have $\Gamma'' \vdash \theta$ *is* θ' *over* Γ **using** *h2 logical-subst-monotonicity* **by** *blast*
then have $\Gamma'' \vdash (x,s')\#\theta$ *is* $(x,t')\#\theta'$ *over* $(x,T_1)\#\Gamma$ **using** *equiv-subst-ext hl fs* **by** *blast*
then have $\Gamma'' \vdash (x,s')\#\theta{<}s_2{>}$ *is* $(x,t')\#\theta'{<}t_2{>} : T_2$ **using** *ih hk* **by** *blast*
then have $\Gamma'' \vdash \theta{<}s_2{>}[x{::=}s']$ *is* $\theta'{<}t_2{>}[x{::=}t'] : T_2$ **using** *fs2 psubst-subst-psubst* **by** *auto*
moreover have $\mathit{App}\ (\mathit{Lam}\ [x].\ \theta{<}s_2{>})\ s' \leadsto \theta{<}s_2{>}[x{::=}s']$
and $\mathit{App}\ (\mathit{Lam}\ [x].\theta'{<}t_2{>})\ t' \leadsto \theta'{<}t_2{>}[x{::=}t']$ **by** *auto*
ultimately have $\Gamma'' \vdash \mathit{App}\ (\mathit{Lam}\ [x].\ \theta{<}s_2{>})\ s'$ *is* $\mathit{App}\ (\mathit{Lam}\ [x].\theta'{<}t_2{>})\ t' : T_2$
using *logical-weak-head-closure* **by** *auto*
}
moreover have *valid* Γ' **using** *h2* **by** *auto*
ultimately have $\Gamma' \vdash \mathit{Lam}\ [x].\theta{<}s_2{>}$ *is* $\mathit{Lam}\ [x].\theta'{<}t_2{>} : T_1{\to}T_2$ **by** *auto*
then show $\Gamma' \vdash \theta{<}\mathit{Lam}\ [x].s_2{>}$ *is* $\theta'{<}\mathit{Lam}\ [x].t_2{>} : T_1{\to}T_2$ **using** *fs2* **by** *auto*
next
case $(\mathit{Q\text{-}App}\ \Gamma\ s_1\ t_1\ T_1\ T_2\ s_2\ t_2\ \Gamma'\ \theta\ \theta')$
then show $\Gamma' \vdash \theta{<}\mathit{App}\ s_1\ s_2{>}$ *is* $\theta'{<}\mathit{App}\ t_1\ t_2{>} : T_2$ **by** *auto*
next
case $(\mathit{Q\text{-}Beta}\ x\ \Gamma\ s_2\ t_2\ T_1\ \mathit{s12}\ \mathit{t12}\ T_2\ \Gamma'\ \theta\ \theta')$
have h: $\Gamma' \vdash \theta$ *is* θ' *over* Γ
and h': *valid* Γ' **by** *fact*
have fs: $x\#\Gamma$ **by** *fact*
have $\mathit{fs2}$: $x\#\theta\ x\#\theta'$ **by** *fact*
have $\mathit{ih1}$: $\bigwedge\Gamma'\ \theta\ \theta'.\ [\![\Gamma' \vdash \theta\ \mathit{is}\ \theta'\ \mathit{over}\ \Gamma;\ \mathit{valid}\ \Gamma']\!] \Longrightarrow \Gamma' \vdash \theta{<}s_2{>}\ \mathit{is}\ \theta'{<}t_2{>} : T_1$ **by** *fact*
have $\mathit{ih2}$: $\bigwedge\Gamma'\ \theta\ \theta'.\ [\![\Gamma' \vdash \theta\ \mathit{is}\ \theta'\ \mathit{over}\ (x,T_1)\#\Gamma;\ \mathit{valid}\ \Gamma']\!] \Longrightarrow \Gamma' \vdash \theta{<}\mathit{s12}{>}\ \mathit{is}\ \theta'{<}\mathit{t12}{>} : T_2$ **by** *fact*
have $\Gamma' \vdash \theta{<}s_2{>}$ *is* $\theta'{<}t_2{>} : T_1$ **using** *ih1 h' h* **by** *auto*
then have $\Gamma' \vdash (x,\theta{<}s_2{>})\#\theta$ *is* $(x,\theta'{<}t_2{>})\#\theta'$ *over* $(x,T_1)\#\Gamma$ **using** *equiv-subst-ext h fs* **by** *blast*
then have $\Gamma' \vdash (x,\theta{<}s_2{>})\#\theta{<}\mathit{s12}{>}$ *is* $(x,\theta'{<}t_2{>})\#\theta'{<}\mathit{t12}{>} : T_2$ **using** *ih2 h'* **by** *auto*
then have $\Gamma' \vdash \theta{<}\mathit{s12}{>}[x{::=}\theta{<}s_2{>}]$ *is* $\theta'{<}\mathit{t12}{>}[x{::=}\theta'{<}t_2{>}] : T_2$ **using** *fs2 psubst-subst-psubst* **by** *auto*
then have $\Gamma' \vdash \theta{<}\mathit{s12}{>}[x{::=}\theta{<}s_2{>}]$ *is* $\theta'{<}\mathit{t12}[x{::=}t_2]{>} : T_2$ **using** *fs2 psubst-subst-propagate* **by** *auto*
moreover have $\mathit{App}\ (\mathit{Lam}\ [x].\theta{<}\mathit{s12}{>})\ (\theta{<}s_2{>}) \leadsto \theta{<}\mathit{s12}{>}[x{::=}\theta{<}s_2{>}]$ **by** *auto*
ultimately have $\Gamma' \vdash \mathit{App}\ (\mathit{Lam}\ [x].\theta{<}\mathit{s12}{>})\ (\theta{<}s_2{>})$ *is* $\theta'{<}\mathit{t12}[x{::=}t_2]{>} : T_2$
using *logical-weak-head-closure'* **by** *auto*
then show $\Gamma' \vdash \theta{<}\mathit{App}\ (\mathit{Lam}\ [x].\mathit{s12})\ s_2{>}$ *is* $\theta'{<}\mathit{t12}[x{::=}t_2]{>} : T_2$ **using** *fs2* **by** *simp*
next
case $(\mathit{Q\text{-}Ext}\ x\ \Gamma\ s\ t\ T_1\ T_2\ \Gamma'\ \theta\ \theta')$
have $\mathit{h2}$: $\Gamma' \vdash \theta$ *is* θ' *over* Γ

and $h2'$: *valid* Γ' **by** *fact*
have $fs{:}x\#\Gamma\ x\#s\ x\#t$ **by** *fact*
have $ih{:}\bigwedge\Gamma'\ \theta\ \theta'.\ [\![\Gamma' \vdash \theta\ is\ \theta'\ over\ (x,T_1)\#\Gamma;\ valid\ \Gamma']\!]$
$\Longrightarrow \Gamma' \vdash \theta<App\ s\ (Var\ x)>\ is\ \theta'<App\ t\ (Var\ x)> : T_2$ **by** *fact*
{
fix $\Gamma''\ s'\ t'$
assume *hsub*: $\Gamma' \subseteq \Gamma''$ **and** *hl*: $\Gamma'' \vdash s'\ is\ t' : T_1$ **and** *hk*: *valid* Γ''
then have $\Gamma'' \vdash \theta\ is\ \theta'\ over\ \Gamma$ **using** *h2 logical-subst-monotonicity* **by** *blast*
then have $\Gamma'' \vdash (x,s')\#\theta\ is\ (x,t')\#\theta'\ over\ (x,T_1)\#\Gamma$ **using** *equiv-subst-ext hl fs* **by** *blast*
then have $\Gamma'' \vdash (x,s')\#\theta<App\ s\ (Var\ x)>\ \ is\ (x,t')\#\theta'<App\ t\ (Var\ x)> : T_2$ **using** *ih hk* **by** *blast*
then
have $\Gamma'' \vdash App\ (((x,s')\#\theta)<s>)\ (((x,s')\#\theta)<(Var\ x)>)\ is\ App\ ((x,t')\#\theta'<t>)\ ((x,t')\#\theta'<(Var\ x)>) : T_2$
by *auto*
then have $\Gamma'' \vdash App\ ((x,s')\#\theta<s>)\ s'\ \ is\ App\ ((x,t')\#\theta'<t>)\ t' : T_2$ **by** *auto*
then have $\Gamma'' \vdash App\ (\theta<s>)\ s'\ is\ App\ (\theta'<t>)\ t' : T_2$ **using** *fs fresh-psubst-simp* **by** *auto*
}
moreover have *valid* Γ' **using** *h2* **by** *auto*
ultimately show $\Gamma' \vdash \theta<s>\ is\ \theta'<t> : T_1 \rightarrow T_2$ **by** *auto*
next
case $(Q\text{-}Unit\ \Gamma\ s\ t\ \Gamma'\ \theta\ \theta')$
then show $\Gamma' \vdash \theta<s>\ is\ \theta'<t> : TUnit$ **by** *auto*
qed

6.6 Completeness

theorem *completeness*:
assumes *asm*: $\Gamma \vdash s \equiv t : T$
shows $\Gamma \vdash s \Leftrightarrow t : T$
proof −
have *val*: *valid* Γ **using** *def-equiv-implies-valid asm* **by** *simp*
moreover
{
fix $x\ T$
assume $(x,T) \in set\ \Gamma\ valid\ \Gamma$
then have $\Gamma \vdash Var\ x\ is\ Var\ x : T$ **using** *main-lemma*(*2*) **by** *blast*
}
ultimately have $\Gamma \vdash []\ is\ []\ over\ \Gamma$ **by** *auto*
then have $\Gamma \vdash []<s>\ is\ []<t> : T$ **using** *fundamental-theorem-2 val asm* **by** *blast*
then have $\Gamma \vdash s\ is\ t : T$ **by** *simp*
then show $\Gamma \vdash s \Leftrightarrow t : T$ **using** *main-lemma*(*1*) *val* **by** *simp*
qed

7 About soundness

We leave soundness as an exercise - like in the book :-)
If $\Gamma \vdash s \Leftrightarrow t : T$ and $\Gamma \vdash t : T$ and $\Gamma \vdash s : T$ then $\Gamma \vdash s \equiv t : T$.
$[\![\Gamma \vdash s \leftrightarrow t : T;\ \Gamma \vdash t : T\ ;\ \Gamma \vdash s : T\ ]\!] \Longrightarrow \Gamma \vdash s \equiv t : T$

end

References

[1] S. Berghofer and C. Urban. *The Nominal Datatypes Package user manual.* Technische Universität München, 2006.

[2] K. Crary. Logical Relations and a Case Study in Equivalence Checking. In B. C. Pierce, editor, *Advanced Topics in Types and Programming Languages*, pages 223–244. MIT Press, 2005.

[3] L. C. Paulson. The Isabelle reference manual, 2006.

[4] C. Urban and S. Berghofer. A Recursion Combinator for Nominal Datatypes Implemented in Isabelle/HOL. In *Proc. of the 3rd International Joint Conference on Automated Reasoning (IJCAR)*, volume 4130 of *LNAI*, pages 498–512, 2006.

[5] C. Urban and C. Tasson. Nominal Techniques in Isabelle/HOL. In *Proc. of the 20th International Conference on Automated Deduction (CADE)*, volume 3632 of *LNCS*, pages 38–53, 2005.

[6] M. Wenzel. *Isabelle/Isar — a versatile environment for human-readable formal proof documents.* PhD thesis, Institut für Informatik, Technische Universität München, 2002.