Nominal Formalisations of Typical SOS Proofs
The documented proofs

Julien Narboux and Christian Urban

February 26, 2007

Abstract

Structural operational semantics (SOS) provides a framework for ascribing semantics to programming languages. This is typically done by stating rules for typing judgements, small-step transitions and rules for evaluating an expression of the language. Structural inductions over expressions and inductions over inference rules are thus the most fundamental reasoning techniques employed in SOS. While the SOS-techniques are characterised in Plotkin's seminal notes as "symbol-pushing", programming languages nearly always contain binders and then reasoning is in fact rather subtle. We describe in this paper formalisations of typical proofs in SOS using the nominal datatype package. We show how this package eases the subtleties when reasoning about binders.

Contents

1 Definition of the language **3**
- 1.1 Definition of the terms and types . . . 3
- 1.2 Size functions . . . 4

2 Typing **5**
- 2.1 Typing contexts . . . 5
  - 2.1.1 Definition of the validity of contexts . . . 5
  - 2.1.2 Definition of sub contexts . . . 6
  - 2.1.3 Lemmas about contexts . . . 6
- 2.2 Definition of the typing relation . . . 6
- 2.3 Inversion lemmas for the typing relation . . . 7
- 2.4 Strong induction principle . . . 8
- 2.5 Strong inversion lemmas . . . 11

3 Substitutions **12**
- 3.1 Lookup function . . . 12
- 3.2 Parallel substitution . . . 12
- 3.3 Substitution . . . 13
- 3.4 Lemmas about freshness and substitution . . . 14

4 Big step semantic **16**
- 4.1 Inversion lemmas for the big step relation . . . 17
- 4.2 Strong induction principle for the big step relation . . . 17
- 4.3 Strong inversion lemmas for big step relation . . . 21

5 Values. **22**
- 5.1 Definition of values. . . . 22
- 5.2 Inversion lemmas for values . . . 22

6 Weakening lemma **22**

7 Substitution lemma **23**

8 Subject reduction **25**

9 Examples **25**
- 9.1 A solution to Challenge 5 . . . 26
- 9.2 A solution to Challenge 6 . . . 26

10 Unicity of evaluation **26**

11 Termination of evaluation **28**
- 11.1 Definition of the logical relations . . . 28
- 11.2 Inversion lemmas for logical relations . . . 29
- 11.3 Monotonicity . . . 30
- 11.4 The termination proof . . . 31

1 Definition of the language

1.1 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 *data* = *DNat*
  | *DProd data data*
  | *DSum data data*

nominal-datatype *ty* = *Data data*
  | *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* = *Var name*
  | *Lam* ≪*name*≫*trm* (*Lam* [-].- [*100*,*100*] *100*)
  | *App trm trm*
  | *Const nat*
  | *Pr trm trm*
  | *Fst trm*
  | *Snd trm*
  | *InL trm*
  | *InR trm*
  | *Case trm* ≪*name*≫*trm* ≪*name*≫*trm*
    (*Case* - *of inl* - → - | *inr* - → - [*100*,*100*,*100*,*100*,*100*] *100*)

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

lemma *perm-data*[*simp*]:
 fixes *D*::*data*
 and π::*name prm*

shows $\pi \cdot D = D$
by (*induct D rule*: *data.induct-weak*) (*simp-all*)

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

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

lemma *data-cases*:
fixes x::*data*
shows $x = DNat \vee (\exists\ T_1\ T_2.\ x = DProd\ T_1\ T_2) \vee (\exists\ T_1\ T_2.\ x = DSum\ T_1\ T_2)$
by (*induct x rule*:*data.induct-weak*, *auto*)

lemma *ty-cases*:
fixes x::*ty*
shows $(\exists\ T_1\ T_2.\ x = T_1 \rightarrow T_2) \vee (\exists\ T.\ x = Data\ T)$
by (*induct x rule*:*ty.induct-weak*, *auto*)

1.2 Size functions

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

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

instance *data* :: *size* **..**

nominal-primrec
size DNat = *1*
$size\ (DProd\ S_1\ S_2) = size\ S_1 + size\ S_2$
$size\ (DSum\ S_1\ S_2) = size\ S_1 + size\ S_2$
by (*rule TrueI*)+

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

nominal-primrec
$size\ (Data\ S) = size\ S$
$size\ (T_1 \rightarrow T_2) = size\ T_1 + size\ T_2$
by (*rule TrueI*)+

lemma *data-size-greater-zero*[*simp*]:
fixes S::*data*
shows $size\ S > 0$
by (*nominal-induct rule*: *data.induct*) (*simp-all*)

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

2 Typing

2.1 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.

2.1.1 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:

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

We generate the equivariance lemma for the relation `valid`.

nominal-inductive *valid*

We obtain a lemma called `valid_eqvt`:

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

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

The generated theorem is the following:

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

2.1.2 Definition of sub contexts

abbreviation
  sub :: (*name*×*ty*) *list* ⇒ (*name*×*ty*) *list* ⇒ *bool* (- ≪ - [*55*,*55*] *55*)
where
  $\Gamma_1 \ll \Gamma_2 \equiv \forall x\ T.\ (x,T) \in set\ \Gamma_1 \longrightarrow (x,T) \in set\ \Gamma_2$

2.1.3 Lemmas about contexts

lemma *type-unicity-in-context*:
  assumes *asm1*: $(x,t_2) \in set\ ((x,t_1)\#\Gamma)$
  and *asm2*: *valid* $((x,t_1)\#\Gamma)$
  shows $t_1=t_2$
proof −
  from *asm2* **have** $x\#\Gamma$ **by** (*cases*, *auto*)
  then have $(x,t_2) \notin set\ \Gamma$
    by (*induct* Γ) (*auto simp add*: *fresh-list-cons fresh-prod fresh-atm*)
  then have $(x,t_2) = (x,t_1)$ **using** *asm1* **by** *auto*
  then show $t_1 = t_2$ **by** *auto*
qed

lemma *case-distinction-on-context*:
  fixes Γ::(*name* × *ty*) *list*
  assumes *asm1*: *valid* $((m,t)\#\Gamma)$
  and *asm2*: $(n,U) \in set\ ((m,T)\#\Gamma)$
  shows $(n,U) = (m,T) \lor ((n,U) \in set\ \Gamma \land n \neq m)$
proof −
from *asm2* **have** $(n,U) \in set\ [(m,T)] \lor (n,U) \in set\ \Gamma$ **by** *auto*
moreover
{ assume *eq*: $m=n$
  assume $(n,U) \in set\ \Gamma$
  then have $\neg\ n\#\Gamma$
    by (*induct* Γ) (*auto simp add*: *fresh-list-cons fresh-prod fresh-atm*)
  moreover have $m\#\Gamma$ **using** *asm1* **by** *auto*
  ultimately have *False* **using** *eq* **by** *auto*
}
ultimately show *?thesis* **by** *auto*
qed

2.2 Definition of the typing relation

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

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

nominal-inductive *typing*

$$\frac{\textit{valid}\ \Gamma \qquad (x,\ T) \in \textit{set}\ \Gamma}{\Gamma \vdash \textit{Var}\ x : T}\text{t\_Var} \qquad \frac{\Gamma \vdash e_1 : T_1 \rightarrow T_2 \qquad \Gamma \vdash e_2 : T_1}{\Gamma \vdash \textit{App}\ e_1\ e_2 : T_2}\text{t\_App}$$

$$\frac{x \ \#\ \Gamma \qquad (x,\ T_1)\ \#\ \Gamma \vdash e : T_2}{\Gamma \vdash \textit{Lam}\ [x].e : T_1 \rightarrow T_2}\text{t\_Lam}$$

$$\frac{\textit{valid}\ \Gamma}{\Gamma \vdash \textit{Const}\ n : \textit{Data}\ \textit{DNat}}\text{t\_Const} \qquad \frac{\Gamma \vdash e : \textit{Data}\ (\textit{DProd}\ S_1\ S_2)}{\Gamma \vdash \textit{Fst}\ e : \textit{Data}\ S_1}\text{t\_Fst}$$

$$\frac{\Gamma \vdash e : \textit{Data}\ (\textit{DProd}\ S_1\ S_2)}{\Gamma \vdash \textit{Snd}\ e : \textit{Data}\ S_2}\text{t\_Snd}$$

$$\frac{\Gamma \vdash e : \textit{Data}\ S_1}{\Gamma \vdash \textit{InL}\ e : \textit{Data}\ (\textit{DSum}\ S_1\ S_2)}\text{t\_InL} \qquad \frac{\Gamma \vdash e : \textit{Data}\ S_2}{\Gamma \vdash \textit{InR}\ e : \textit{Data}\ (\textit{DSum}\ S_1\ S_2)}\text{t\_InR}$$

$$\frac{\begin{array}{c} x_1\ \#\ (\Gamma,\ e,\ e_2,\ x_2) \qquad x_2\ \#\ (\Gamma,\ e,\ e_1,\ x_1) \\ \Gamma \vdash e : \textit{Data}\ (\textit{DSum}\ S_1\ S_2) \qquad (x_1,\ \textit{Data}\ S_1)\ \#\ \Gamma \vdash e_1 : T \qquad (x_2,\ \textit{Data}\ S_2)\ \#\ \Gamma \vdash e_2 : T \end{array}}{\Gamma \vdash \textit{Case}\ e\ \textit{of}\ \textit{inl}\ x_1 \rightarrow e_1 \mid \textit{inr}\ x_2 \rightarrow e_2 : T}\text{t\_Case}$$

Figure 1: Typing rules

2.3 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*]
declare *data.inject* [*simp add*]

inductive-cases2 *t-Lam-inv-auto*[*elim*]: $\Gamma \vdash \textit{Lam}\ [x].t : T$
inductive-cases2 *t-Var-inv-auto*[*elim*]: $\Gamma \vdash \textit{Var}\ x : T$
inductive-cases2 *t-App-inv-auto*[*elim*]: $\Gamma \vdash \textit{App}\ x\ y : T$
inductive-cases2 *t-Const-inv-auto*[*elim*]: $\Gamma \vdash \textit{Const}\ n : T$
inductive-cases2 *t-Fst-inv-auto*[*elim*]: $\Gamma \vdash \textit{Fst}\ x : T$
inductive-cases2 *t-Snd-inv-auto*[*elim*]: $\Gamma \vdash \textit{Snd}\ x : T$
inductive-cases2 *t-InL-inv-auto*[*elim*]: $\Gamma \vdash \textit{InL}\ x : T$
inductive-cases2 *t-InL-inv-auto'*[*elim*]: $\Gamma \vdash \textit{InL}\ x : \textit{Data}\ (\textit{DSum}\ T_1\ \textit{T2})$
inductive-cases2 *t-InR-inv-auto*[*elim*]: $\Gamma \vdash \textit{InR}\ x : T$
inductive-cases2 *t-InR-inv-auto'*[*elim*]: $\Gamma \vdash \textit{InR}\ x : \textit{Data}\ (\textit{DSum}\ T_1\ \textit{T2})$
inductive-cases2 *t-Pr-inv-auto*[*elim*]: $\Gamma \vdash \textit{Pr}\ x\ y : T$
inductive-cases2 *t-Pr-inv-auto'*[*elim*]: $\Gamma \vdash \textit{Pr}\ e_1\ e_2 : \textit{Data}\ (\textit{DProd}\ \sigma\textit{1}\ \sigma_2)$
inductive-cases2 *t-Case-inv-auto*[*elim*]: $\Gamma \vdash \textit{Case}\ e\ \textit{of}\ \textit{inl}\ x_1 \rightarrow e_1 \mid \textit{inr}\ x_2 \rightarrow e_2 : T$

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

2.4 Strong induction principle

Now, we define a strong induction principle. This induction principle allow us to avoid some terms during the induction. The bound variable The avoided terms (c)

lemma *typing-induct-strong*
  [*consumes 1*, *case-names t-Var t-App t-Lam t-Const t-Pr t-Fst t-Snd t-InL t-InR t-Case*]:
    fixes P::$'a$::*fs-name* ⇒ (*name*×*ty*) *list* ⇒ *trm* ⇒ *ty* ⇒*bool*
    and x :: $'a$::*fs-name*
    assumes *a*: $\Gamma \vdash e : T$
    and *a1*: $\bigwedge\Gamma\ x\ T\ c.\ [\![valid\ \Gamma;\ (x,T)\in set\ \Gamma]\!] \Longrightarrow P\ c\ \Gamma\ (Var\ x)\ T$
    and *a2*: $\bigwedge\Gamma\ e_1\ T_1\ T_2\ e_2\ c.$
        $[\![\Gamma \vdash e_1 : T_1 \to T_2\ ;\ \bigwedge c.\ P\ c\ \Gamma\ e_1\ (T_1 \to T_2);\ \Gamma \vdash e_2 : T_1\ ;\ \bigwedge c.\ P\ c\ \Gamma\ e_2\ T_1]\!]$
        $\Longrightarrow P\ c\ \Gamma\ (App\ e_1\ e_2)\ T_2$
    and *a3*: $\bigwedge x\ \Gamma\ T_1\ t\ T_2\ c.$
        $[\![x\#(\Gamma,c);\ (x,T_1)\#\Gamma \vdash t : T_2\ ;\ \bigwedge c.\ P\ c\ ((x,T_1)\#\Gamma)\ t\ T_2]\!]$
        $\Longrightarrow P\ c\ \Gamma\ (Lam\ [x].t)\ (T_1 \to T_2)$
    and *a4*: $\bigwedge\Gamma\ n\ c.\ valid\ \Gamma \Longrightarrow P\ c\ \Gamma\ (Const\ n)\ (Data\ DNat)$
    and *a5*: $\bigwedge\Gamma\ e_1\ S_1\ e_2\ S_2\ c.$
        $[\![\Gamma \vdash e_1 : Data\ S_1;\ \bigwedge c.\ P\ c\ \Gamma\ e_1\ (Data\ S_1);\ \Gamma \vdash e_2 : Data\ S_2;\ \bigwedge c.\ P\ c\ \Gamma\ e_2\ (Data\ S_2)]\!]$
        $\Longrightarrow P\ c\ \Gamma\ (Pr\ e_1\ e_2)\ (Data\ (DProd\ S_1\ S_2))$
    and *a6*: $\bigwedge\Gamma\ e\ S_1\ S_2\ c.$
        $[\![\Gamma \vdash e : Data\ (DProd\ S_1\ S_2)\ ;\ \bigwedge c.\ P\ c\ \Gamma\ e\ (Data\ (DProd\ S_1\ S_2))]\!]$
        $\Longrightarrow P\ c\ \Gamma\ (Fst\ e)\ (Data\ S_1)$
    and *a7*: $\bigwedge\Gamma\ e\ S_1\ S_2\ c.$
        $[\![\Gamma \vdash e : Data\ (DProd\ S_1\ S_2)\ ;\ \bigwedge c.\ P\ c\ \Gamma\ e\ (Data\ (DProd\ S_1\ S_2))]\!]$
        $\Longrightarrow P\ c\ \Gamma\ (Snd\ e)\ (Data\ S_2)$
    and *a8*: $\bigwedge\Gamma\ e\ S_1\ S_2\ c.$
        $[\![\Gamma \vdash e : Data\ S_1\ ;\ \bigwedge c.\ P\ c\ \Gamma\ e\ (Data\ S_1)]\!] \Longrightarrow P\ c\ \Gamma\ (InL\ e)\ (Data\ (DSum\ S_1\ S_2))$
    and *a9*: $\bigwedge\Gamma\ e\ S_2\ S_1\ c.$
        $[\![\Gamma \vdash e : Data\ S_2\ ;\ \bigwedge c.\ P\ c\ \Gamma\ e\ (Data\ S_2)]\!] \Longrightarrow P\ c\ \Gamma\ (InR\ e)\ (Data\ (DSum\ S_1\ S_2))$
    and *a10*: $\bigwedge x_1\ \Gamma\ e\ e_2\ x_2\ e_1\ S_1\ S_2\ T\ c.$
        $[\![x_1\#(\Gamma,e,e_2,x_2,c);\ x_2\#(\Gamma,e,e_1,x_1,c);\ \ \Gamma \vdash e : Data\ (DSum\ S_1\ S_2);$
        $\bigwedge c.\ P\ c\ \Gamma\ e\ (Data\ (DSum\ S_1\ S_2));\ (x_1,Data\ S_1)\#\Gamma \vdash e_1 : T;$
        $\bigwedge c.\ P\ c\ ((x_1,Data\ S_1)\#\Gamma)\ e_1\ T;\ ((x_2,Data\ S_2)\#\Gamma) \vdash e_2 : T\ ;$
        $\bigwedge c.\ P\ c\ ((x_2,Data\ S_2)\#\Gamma)\ e_2\ T]\!]$
        $\Longrightarrow P\ c\ \Gamma\ (Case\ e\ of\ inl\ x_1 \to e_1 \mid inr\ x_2 \to e_2)\ T$
    shows $P\ c\ \Gamma\ e\ T$
proof −
  from *a* **have** $\bigwedge(\pi::name\ prm)\ c.\ P\ c\ (\pi\cdot\Gamma)\ (\pi\cdot e)\ T$
  proof (*induct*)
    case (*t-Var* $\Gamma\ x\ T\ \pi\ c$)
    have *valid* Γ **by** *fact*
    then have *valid* $(\pi\cdot\Gamma)$ **by** (*simp only*: *eqvt*)
    moreover
    have $(x,T)\in set\ \Gamma$ **by** *fact*
    then have $\pi\cdot(x,T)\in\pi\cdot(set\ \Gamma)$ **by** (*simp only*: *pt-set-bij*[*OF pt-name-inst*, *OF at-name-inst*])
    then have $(\pi\cdot x,T)\in set\ (\pi\cdot\Gamma)$ **by** (*simp add*: *eqvt*)
    ultimately show $P\ c\ (\pi\cdot\Gamma)\ (\pi\cdot(Var\ x))\ T$ **using** *a1* **by** *simp*
  next
    case (*t-App* $\Gamma\ e_1\ T_1\ T_2\ e_2\ \pi\ c$)
    thus $P\ c\ (\pi\cdot\Gamma)\ (\pi\cdot(App\ e_1\ e_2))\ T_2$ **using** *a2* **by** (*simp*, *blast intro*: *eqvt*[*simplified perm-ty*])
  next

case (*t-Lam* $x\ \Gamma\ T_1\ t\ T_2\ \pi\ c$)
obtain y::*name* **where** *fs*: $y\#(\pi\bullet x,\pi\bullet\Gamma,\pi\bullet t,c)$ **by** (*erule exists-fresh*[*OF fs-name1*])
let *?sw* = $[(\pi\bullet x,y)]$
let *?pi′* = *?sw*@π
have *f0*: $x\#\Gamma$ **by** *fact*
have *f1*: $(\pi\bullet x)\#(\pi\bullet\Gamma)$ **using** *f0* **by** (*simp add*: *fresh-bij*)
have *f2*: $y\#$*?pi′*$\bullet\Gamma$ **by** (*auto simp add*: *pt-name2 fresh-left calc-atm perm-pi-simp*)
have *pr1*: $(x,T_1)\#\Gamma \vdash t : T_2$ **by** *fact*
then have (*?pi′*$\bullet((x,T_1)\#\Gamma)) \vdash$ (*?pi′*$\bullet t$) : T_2 **by** (*simp only*: *typing-eqvt*[*simplified perm-ty*])
moreover
have *ih1*: $\bigwedge c.\ P\ c$ (*?pi′*$\bullet((x,T_1)\#\Gamma)$) (*?pi′*$\bullet t$) T_2 **by** *fact*
ultimately have $P\ c$ (*?pi′*$\bullet\Gamma$) (*Lam* $[y]$.(*?pi′*$\bullet t$)) $(T_1\to T_2)$ **using** *fs f2 a3*
  by (*simp add*: *calc-atm*)
then have $P\ c$ (*?sw*$\bullet\pi\bullet\Gamma$) (*?sw*$\bullet$(*Lam* $[(\pi\bullet x)].(\pi\bullet t)$)) $(T_1\to T_2)$
  by (*simp del*: *append-Cons add*: *calc-atm pt-name2*)
moreover have (*?sw*$\bullet\pi\bullet\Gamma$) = $(\pi\bullet\Gamma)$
  by (*rule perm-fresh-fresh*) (*simp-all add*: *fs f1*)
moreover have (*?sw*$\bullet$(*Lam* $[(\pi\bullet x)].(\pi\bullet t)$)) = *Lam* $[(\pi\bullet x)].(\pi\bullet t)$
  by (*rule perm-fresh-fresh*) (*simp-all add*: *fs f1 fresh-atm abs-fresh*)
ultimately show $P\ c\ (\pi\bullet\Gamma)$ ($\pi\bullet$(*Lam* $[x].t$)) $(T_1\to T_2)$
  by *simp*
next
case (*t-Const* $\Gamma\ n\ \pi\ c$)
thus $P\ c\ (\pi\bullet\Gamma)$ ($\pi\bullet$(*Const* n)) (*Data DNat*) **using** *a4* **by** (*simp*, *blast intro*: *eqvt*)
next
case (*t-Pr* $\Gamma\ e_1\ S_1\ e_2\ S_2\ \pi\ c$)
thus $P\ c\ (\pi\bullet\Gamma)$ ($\pi\bullet$(*Pr* $e_1\ e_2$)) (*Data* (*DProd* $S_1\ S_2$)) **using** *a5* **by** (*simp*, *blast intro*: *eqvt*[*simplified perm-ty*])
next
case (*t-Fst* $\Gamma\ e\ S_1\ S_2\ \pi\ c$)
thus $P\ c\ (\pi\bullet\Gamma)$ ($\pi\bullet$(*Fst* e)) (*Data* S_1) **using** *a6* **by** (*simp*, *blast intro*: *eqvt*[*simplified perm-ty*])
next
case (*t-Snd* $\Gamma\ e\ S_1\ S_2\ \pi\ c$)
thus $P\ c\ (\pi\bullet\Gamma)$ ($\pi\bullet$(*Snd* e)) (*Data* S_2) **using** *a7* **by** (*simp*, *blast intro*: *eqvt*[*simplified perm-ty*])
next
case (*t-InL* $\Gamma\ e\ S_1\ S_2\ \pi\ c$)
thus $P\ c\ (\pi\bullet\Gamma)$ ($\pi\bullet$(*InL* e)) (*Data* (*DSum* $S_1\ S_2$)) **using** *a8* **by** (*simp*, *blast intro*: *eqvt*[*simplified perm-ty*])
next
case (*t-InR* $\Gamma\ e\ S_2\ S_1\ \pi\ c$)
thus $P\ c\ (\pi\bullet\Gamma)$ ($\pi\bullet$(*InR* e)) (*Data* (*DSum* $S_1\ S_2$)) **using** *a9* **by** (*simp*, *blast intro*: *eqvt*[*simplified perm-ty*])
next
case (*t-Case* $x_1\ \Gamma\ e\ e_2\ x_2\ e_1\ S_1\ S_2\ T\ \pi\ c$)
obtain y_1::*name* **where** *fs1*: $y_1\#(\pi\bullet x_1,\pi\bullet x_2,\pi\bullet e,\pi\bullet e_1,\pi\bullet e_2,\pi\bullet\Gamma,c)$
  by (*erule exists-fresh*[*OF fs-name1*])
obtain y_2::*name* **where** *fs2*: $y_2\#(\pi\bullet x_1,\pi\bullet x_2,\pi\bullet e,\pi\bullet e_1,\pi\bullet e_2,\pi\bullet\Gamma,c,y_1)$
  by (*erule exists-fresh*[*OF fs-name1*])
let *?sw1* = $[(\pi\bullet x_1,y_1)]$
let *?sw2* = $[(\pi\bullet x_2,y_2)]$
let *?pi′* = *?sw2*@*?sw1*@π
have *f01*: $x_1\#(\Gamma,e,e_2,x_2)$ **by** *fact*
have *f11*: $(\pi\bullet x_1)\#(\pi\bullet\Gamma,\pi\bullet e,\pi\bullet e_2,\pi\bullet x_2)$ **using** *f01* **by** (*simp add*: *fresh-bij*)

```
    have f21: y₁#(?pi'•Γ,?pi'•e,?pi'•e₂) using f01 fs1 fs2
      by (simp add: fresh-atm fresh-prod fresh-left calc-atm pt-name2 perm-pi-simp)
    have f02: x₂#(Γ,e,e₁,x₁) by fact
    have f12: (π•x₂)#(π•Γ,π•e,π•e₁,π•x₁) using f02 by (simp add: fresh-bij)
    have f22: y₂#(?pi'•Γ,?pi'•e,?pi'•e₁) using f02 fs1 fs2
      by (auto simp add: fresh-atm fresh-prod fresh-left calc-atm pt-name2 perm-pi-simp)
    have pr1: Γ ⊢ e : Data (DSum S₁ S₂) by fact
    then have (?pi'•Γ) ⊢ (?pi'•e) : Data (DSum S₁ S₂) by (simp only: eqvt[simplified perm-ty])
    moreover
    have pr2: (x₁,Data S₁)#Γ ⊢ e₁ : T by fact
   then have (?pi'•((x₁,Data S₁)#Γ)) ⊢ (?pi'•e₁) : T by (simp only: typing-eqvt[simplified perm-ty])
    then have (y₁,Data S₁)#(?pi'•Γ) ⊢ (?pi'•e₁) : T using fs1 fs2
      by (auto simp add: calc-atm fresh-prod fresh-atm)
    moreover
    have pr2: (x₂,Data S₂)#Γ ⊢ e₂ : T by fact
   then have (?pi'•((x₂,Data S₂)#Γ)) ⊢ (?pi'•e₂) : T by (simp only: typing-eqvt[simplified perm-ty])
    then have (y₂,Data S₂)#(?pi'•Γ) ⊢ (?pi'•e₂) : T using fs1 fs2 f11 f12
      by (simp add: calc-atm fresh-prod fresh-atm)
    moreover
    have ih1: ⋀c. P c (?pi'•Γ) (?pi'•e) (Data (DSum S₁ S₂)) by fact
    moreover
    have ih2: ⋀c. P c (?pi'•((x₁,Data S₁)#Γ)) (?pi'•e₁) T by fact
    then have ⋀c. P c ((y₁,Data S₁)#(?pi'•Γ)) (?pi'•e₁) T using fs1 fs2
      by (auto simp add: calc-atm fresh-prod fresh-atm)
    moreover
    have ih3: ⋀c. P c (?pi'•((x₂,Data S₂)#Γ)) (?pi'•e₂) T by fact
    then have ⋀c. P c ((y₂,Data S₂)#(?pi'•Γ)) (?pi'•e₂) T using fs1 fs2 f11 f12
      by (simp add: calc-atm fresh-prod fresh-atm)
    ultimately have P c (?pi'•Γ) (Case (?pi'•e) of inl y₁ → (?pi'•e₁) | inr y₂ → (?pi'•e₂)) T
      using f21 f22 fs1 fs2 a10 by (auto simp add: fresh-atm fresh-prod)
    then have P c (?sw2•?sw1•π•Γ)
                  (?sw2•?sw1•(Case (π•e) of inl (π•x₁) → (π•e₁) | inr (π•x₂) → (π•e₂))) T
      using fs1 fs2 f01 f02 f11 f12
      by (auto simp del: append-Cons simp add: pt-name2 fresh-atm fresh-prod calc-atm)
    moreover have (?sw1•π•Γ) = (π•Γ)
      by (rule perm-fresh-fresh) (simp-all add: fs1 f11)
    moreover have (?sw2•π•Γ) = (π•Γ)
      by (rule perm-fresh-fresh) (simp-all add: fs2 f12)
    moreover have ?sw1•(Case (π•e) of inl (π•x₁) → (π•e₁) | inr (π•x₂) → (π•e₂))
                  = (Case (π•e) of inl (π•x₁) → (π•e₁) | inr (π•x₂) → (π•e₂))
      by (rule perm-fresh-fresh) (simp-all add: fs1 fs2 f11 f12 abs-fresh)
    moreover have ?sw2•(Case (π•e) of inl (π•x₁) → (π•e₁) | inr (π•x₂) → (π•e₂))
                  = (Case (π•e) of inl (π•x₁) → (π•e₁) | inr (π•x₂) → (π•e₂))
      by (rule perm-fresh-fresh) (simp-all add: fs1 fs2 f11 f12 abs-fresh)
    ultimately show P c (π•Γ) (π•(Case e of inl x₁ → e₁ | inr x₂ → e₂)) T
      by (simp only:, simp)
  qed
  then have P c (([]::name prm)•Γ) (([]::name prm)•e) T by blast
  then show P c Γ e T by simp
qed
```

2.5 Strong inversion lemmas

Now, we derive strong inversion lemmas for the `t_Lam` and `t_Case` rule.

lemma *t-Lam-elim*[*elim*] :
  assumes *a1*:$\Gamma \vdash$ *Lam* [x].t : T
  and *a2*: $x\#\Gamma$
  obtains T_1 **and** T_2 **where** $(x,T_1)\#\Gamma \vdash t : T_2$ **and** $T{=}T_1{\to}T_2$
proof −
  from *a1* **obtain** x' t' T_1 T_2
    where *b1*: $x'\#\Gamma$ **and** *b2*: $(x',T_1)\#\Gamma \vdash t' : T_2$ **and** *b3*: $[x'].t' = [x].t$ **and** *b4*: $T{=}T_1{\to}T_2$
    by *auto*
  obtain c::*name* **where** $c\#(\Gamma,x,x',t,t')$ **by** (*erule exists-fresh*[*OF fs-name1*])
  then have *fs*: $c\#\Gamma$ $c{\neq}x$ $c{\neq}x'$ $c\#t$ $c\#t'$ **by** (*simp-all add*: *fresh-atm*[*symmetric*])
  then have *b5*: $[(x',c)]{\bullet}t'{=}[(x,c)]{\bullet}t$ **using** *b3 fs* **by** (*simp add*: *alpha'*)
  have $([(x,c)]{\bullet}[(x',c)]{\bullet}((x',T_1)\#\Gamma)) \vdash ([(x,c)]{\bullet}[(x',c)]{\bullet}t') : T_2$ **using** *b2*
    by (*simp only*: *typing-eqvt*[*simplified perm-ty*])
  then have $(x,T_1)\#\Gamma \vdash t : T_2$ **using** *fs b1 a2 b5* **by** (*perm-simp add*: *calc-atm*)
  then show *?thesis* **using** *prems b4* **by** *simp*
qed

lemma *t-Case-elim*[*elim*] :
  assumes $\Gamma \vdash$ *Case e of inl* $x_1 \to e_1$ | *inr* $x_2 \to e_2$: T **and** $x_1\#\Gamma$ **and** $x_2\#\Gamma$
  obtains σ_1 σ_2 **where** $\Gamma \vdash e$: *Data* (*DSum* σ_1 σ_2) **and** (x_1, *Data* σ_1)$\#\Gamma \vdash e_1 : T$ **and** (x_2, *Data* σ_2)$\#\Gamma \vdash e_2 : T$
  proof −
    have *f*:$x_1\#\Gamma$ $x_2\#\Gamma$ **by** *fact*
    have $\Gamma \vdash$ *Case e of inl* $x_1 \to e_1$ | *inr* $x_2 \to e_2$: T **by** *fact*
    then obtain σ_1 σ_2 x_1' x_2' e_1' e_2' **where**
      h:$\Gamma \vdash e$: *Data* (*DSum* σ_1 σ_2) **and**
      h1:(x_1',*Data* σ_1)$\#\Gamma \vdash e_1' : T$ **and**
      h2:(x_2',*Data* σ_2)$\#\Gamma \vdash e_2' : T$ **and**
      e1:$[x_1].e_1{=}[x_1'].e_1'$ **and** *e2*:$[x_2].e_2{=}[x_2'].e_2'$
       by *auto*
     obtain c::*name* **where** *f'*:c # (x_1,x_1',e_1,e_1',Γ) **by** (*erule exists-fresh*[*OF fs-name1*])
    have *e1'*:$[(x_1,c)] \cdot e_1 = [(x_1',c)] \cdot e_1'$ **using** *e1 f'* **by** (*auto simp add*: *alpha' fresh-prod fresh-atm*)
     have $[(x_1',c)] \cdot ((x_1',$*Data* $\sigma_1)\#\ \Gamma) \vdash [(x_1',c)] \cdot e_1' : T$ **using** *h1 typing-eqvt*[*simplified perm-ty*] **by** *blast*
      then have *x*:(c,*Data* σ_1)#($[(x_1',c)] \cdot \Gamma) \vdash [(x_1',c)] \cdot e_1'$: T **using** *f'* **by** (*auto simp add*: *fresh-atm calc-atm*)
     have x_1' # Γ **using** *h1 typing-valid* **by** *auto*
    then have (c,*Data* σ_1)$\#\Gamma \vdash [(x_1\ ,c)] \cdot e_1 : T$ **using** *f' x e1'* **by** (*auto simp add*: *perm-fresh-fresh*)
     then have $[(x_1,c)] \cdot ((c,$*Data* $\sigma_1)\#\Gamma) \vdash [(x_1,c)] \cdot [(x_1\ ,c)] \cdot e_1 : T$ **using** *typing-eqvt*[*simplified perm-ty*] **by** *blast*
     then have $([(x_1,c)] \cdot (c,$*Data* $\sigma_1))\ \#\Gamma \vdash [(x_1,c)] \cdot [(x_1\ ,c)] \cdot e_1 : T$ **using** *f f'* **by** (*auto simp add*: *perm-fresh-fresh*)
     then have $([(x_1,c)] \cdot (c,$*Data* $\sigma_1))\ \#\Gamma \vdash e_1 : T$ **by** *perm-simp*
      then have *g1*:(x_1, *Data* σ_1)$\#\Gamma \vdash e_1 : T$ **using** *f'* **by** (*auto simp add*: *fresh-atm calc-atm fresh-prod*)

     obtain c::*name* **where** *f'*:c # (x_2,x_2',e_2,e_2',Γ) **by** (*erule exists-fresh*[*OF fs-name1*])
    have *e2'*:$[(x_2,c)] \cdot e_2 = [(x_2',c)] \cdot e_2'$ **using** *e2 f'* **by** (*auto simp add*: *alpha' fresh-prod fresh-atm*)
     have $[(x_2',c)] \cdot ((x_2',$*Data* $\sigma_2)\#\ \Gamma) \vdash [(x_2',c)] \cdot e_2' : T$ **using** *h2 typing-eqvt*[*simplified perm-ty*]

by *blast*
then have x:(c,*Data* σ_2)#([($x_2{}'$,c)] $\cdot$ Γ) ⊢ [($x_2{}'$,c)] $\cdot$ $e_2{}'$: T **using** f' **by** (*auto simp add*: *fresh-atm calc-atm*)
have $x_2{}'$ # Γ **using** *h2 typing-valid* **by** *auto*
then have (c,*Data* σ_2)#Γ ⊢ [(x_2 ,c)] $\cdot$ e_2 : T **using** f' x *e2'* **by** (*auto simp add*: *perm-fresh-fresh*)
then have [(x_2,c)] $\cdot$ ((c,*Data* σ_2)#Γ) ⊢ [(x_2,c)] $\cdot$ [(x_2 ,c)] $\cdot$ e_2 : T **using** *typing-eqvt*[*simplified perm-ty*] **by** *blast*
then have ([(x_2,c)] $\cdot$ (c,*Data* σ_2))#Γ ⊢ [(x_2,c)] $\cdot$ [(x_2 ,c)] $\cdot$ e_2 : T **using** f f' **by** (*auto simp add*: *perm-fresh-fresh*)
then have ([(x_2,c)] $\cdot$ (c,*Data* σ_2)) #Γ ⊢ e_2 : T **by** *perm-simp*
then have *g2*:(x_2,*Data* σ_2)#Γ ⊢ e_2 : T **using** f' **by** (*auto simp add*: *fresh-atm calc-atm fresh-prod*)
show *?thesis* **using** *g1 g2 prems* **by** *auto*
qed

3 Substitutions

Capture-avoiding substitution

3.1 Lookup function

fun
lookup :: (*name*×*trm*) *list* ⇒ *name* ⇒ *trm*
where
lookup [] x = *Var* x
lookup ((y,e)#θ) x = (*if* x=y *then* e *else* *lookup* θ x)

lemma *lookup-eqvt*:
fixes π::*name prm*
and θ::(*name*×*trm*) *list*
and X::*name*
shows $\pi\cdot$(*lookup* θ X) = *lookup* ($\pi\cdot\theta$) ($\pi\cdot X$)
by (*induct* θ, *auto simp add*: *perm-bij*)

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

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

3.2 Parallel substitution

consts

```
  psubst :: (name×trm) list ⇒ trm ⇒ trm  (-<-> [100,100] 100)

nominal-primrec
  θ<(Var x)> = (lookup θ x)
  θ<(App e1 e2)> = App (θ<e1>) (θ<e2>)
  x#θ ⟹ θ<(Lam [x].e)> = Lam [x].(θ<e>)
  θ<(Const n)> = Const n
  θ<(Pr e1 e2)> = Pr (θ<e1>) (θ<e2>)
  θ<(Fst e)> = Fst (θ<e>)
  θ<(Snd e)> = Snd (θ<e>)
  θ<(InL e)> = InL (θ<e>)
  θ<(InR e)> = InR (θ<e>)
  ⟦y≠x; x#(e,e2,θ); y#(e,e1,θ)⟧
   ⟹ θ<(Case e of inl x → e1 | inr y → e2)> =
                            (Case (θ<e>) of inl x → (θ<e1>) | inr y → (θ<e2>))
  apply(finite-guess add: fs-name1 lookup-eqvt)+
  apply(perm-full-simp)
  apply(simp add: fs-name1)
  apply(rule TrueI)+
  apply(simp add: abs-fresh)+
  apply(fresh-guess add: fs-name1 lookup-eqvt)+
  apply(perm-full-simp)
  apply(fresh-guess add: fs-name1 lookup-eqvt)+
  apply(perm-full-simp)
  apply(fresh-guess add: fs-name1 lookup-eqvt)
  apply(perm-full-simp)
  apply(simp-all)
  done

lemma psubst-eqvt[eqvt]:
  fixes π::name prm
  and   t::trm
  shows π•(θ<t>) = (π•θ)<(π•t)>
  by (nominal-induct t avoiding: θ rule: trm.induct)
     (perm-simp add: fresh-bij lookup-eqvt)+

lemma fresh-psubst:
  fixes z::name
  and   t::trm
  assumes z#t and z#θ
  shows z#(θ<t>)
using assms
by (nominal-induct t avoiding: z θ t rule: trm.induct)
   (auto simp add: abs-fresh lookup-fresh)
```

3.3 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 $x\#(y,t') \Longrightarrow (Lam\ [x].t)[y{::=}t'] = Lam\ [x].(t[y{::=}t'])$
and $(Const\ n)[y{::=}t'] = Const\ n$
and $(Pr\ e_1\ e_2)[y{::=}t'] = Pr\ (e_1[y{::=}t'])\ (e_2[y{::=}t'])$
and $(Fst\ e)[y{::=}t'] = Fst\ (e[y{::=}t'])$
and $(Snd\ e)[y{::=}t'] = Snd\ (e[y{::=}t'])$
and $(InL\ e)[y{::=}t'] = InL\ (e[y{::=}t'])$
and $(InR\ e)[y{::=}t'] = InR\ (e[y{::=}t'])$
and $[\![z{\neq}x;\ x\#(y,e,e_2,t');\ z\#(y,e,e_1,t')]\!]$
$\Longrightarrow (Case\ e\ of\ inl\ x \rightarrow e_1 \mid inr\ z \rightarrow e_2)[y{::=}t'] =$
$(Case\ (e[y{::=}t'])\ of\ inl\ x \rightarrow (e_1[y{::=}t']) \mid inr\ z \rightarrow (e_2[y{::=}t']))$
by (*simp-all add*: *fresh-list-cons fresh-list-nil*)

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

3.4 Lemmas about freshness and substitution

lemma *subst-rename*:
fixes $c{::}name$
and $t_1{::}trm$
assumes $c\#t_1$
shows $t_1[a{::=}t_2] = ([(c,a)]{\bullet}t_1)[c{::=}t_2]$
using *assms*
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*)
apply(*simp add*: *trm.inject calc-atm fresh-atm abs-fresh perm-nat-def*)
apply(*simp add*: *trm.inject calc-atm fresh-atm abs-fresh perm-nat-def*)
apply(*simp add*: *trm.inject calc-atm fresh-atm abs-fresh perm-nat-def*)
apply(*simp add*: *trm.inject calc-atm fresh-atm abs-fresh perm-nat-def*)
apply(*simp add*: *trm.inject calc-atm fresh-atm abs-fresh perm-nat-def*)
apply(*simp add*: *trm.inject calc-atm fresh-atm abs-fresh perm-nat-def*)
apply(*simp add*: *trm.inject calc-atm fresh-atm abs-fresh perm-nat-def*)
apply(*simp add*: *trm.inject calc-atm fresh-atm abs-fresh perm-nat-def*)
apply(*simp* (*no-asm-use*))
apply(*rule sym*)
apply(*rule trans*)
apply(*rule subst*)
apply(*simp add*: *perm-bij*)
apply(*simp add*: *fresh-prod*)
apply(*simp add*: *fresh-bij*)
apply(*simp add*: *calc-atm fresh-atm*)
apply(*simp add*: *fresh-prod*)
apply(*simp add*: *fresh-bij*)
apply(*simp add*: *calc-atm fresh-atm*)
apply(*rule sym*)
apply(*rule trans*)

apply(*rule subst*)
apply(*simp add*: *fresh-atm*)
apply(*simp*)
apply(*simp*)
apply(*simp* (*no-asm-use*) *add*: *trm.inject*)
apply(*rule conjI*)
apply(*blast*)
apply(*rule conjI*)
apply(*rotate-tac 12*)
apply(*drule-tac x=a* **in** *meta-spec*)
apply(*rotate-tac 14*)
apply(*drule-tac x=c* **in** *meta-spec*)
apply(*rotate-tac 14*)
apply(*drule-tac* $x{=}t_2$ **in** *meta-spec*)
apply(*simp add*: *calc-atm fresh-atm alpha abs-fresh*)
apply(*rotate-tac 13*)
apply(*drule-tac x=a* **in** *meta-spec*)
apply(*rotate-tac 14*)
apply(*drule-tac x=c* **in** *meta-spec*)
apply(*rotate-tac 14*)
apply(*drule-tac* $x{=}t_2$ **in** *meta-spec*)
apply(*simp add*: *calc-atm fresh-atm alpha abs-fresh*)
done

lemma *fresh-subst*:
fixes *z*::*name*
and t_1::*trm*
and *t2*::*trm*
assumes $z\#t_1$ **and** $z\#t_2$
shows $z\#t_1[y{::=}t_2]$
using *assms*
by (*nominal-induct* t_1 *avoiding*: *z y* t_2 *rule*: *trm.induct*)
(*auto simp add*: *abs-fresh fresh-atm*)

lemma *fresh-subst'*:
fixes *z*::*name*
and t_1::*trm*
and *t2*::*trm*
assumes $z\#[y].t_1$ **and** $z\#t_2$
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 *forget*:
fixes *x*::*name*
and *L*::*trm*
assumes $x\#L$
shows $L[x{::=}P] = L$
using *assms*
by (*nominal-induct L avoiding*: *x P rule*: *trm.induct*)
(*auto simp add*: *fresh-atm abs-fresh*)

```
lemma subst-fun-eq:
  fixes u::trm
  assumes [x].t1 = [y].t2
  shows t1[x::=u] = t2[y::=u]
proof -
  {
    assume x=y and t1=t2
    then have ?thesis using assms 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 assms 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
apply(nominal-induct t avoiding: θ c s rule: trm.induct)
apply(auto simp add: fresh-list-cons fresh-atm forget lookup-fresh lookup-fresh' fresh-psubst)
done

lemma fresh-subst-fresh:
    assumes a#e
    shows a#t[a::=e]
using assms
by (nominal-induct t avoiding: a e rule: trm.induct)
   (auto simp add: fresh-atm abs-fresh fresh-nat)
```

4 Big step semantic

```
inductive2
  big :: trm⇒trm⇒bool (- ⇓ - [80,80] 80)
where
  b-Lam[intro]:   Lam [x].e ⇓ Lam [x].e
| b-App[intro]:   ⟦x#(e1,e2,e'); e1⇓Lam [x].e; e2⇓e2'; e[x::=e2']⇓e'⟧ ⟹ App e1 e2 ⇓ e'
| b-Const[intro]: Const n ⇓ Const n
| b-Pr[intro]:    ⟦e1⇓e1'; e2⇓e2'⟧ ⟹ Pr e1 e2 ⇓ Pr e1' e2'
| b-Fst[intro]:   e⇓Pr e1 e2 ⟹ Fst e⇓e1
| b-Snd[intro]:   e⇓Pr e1 e2 ⟹ Snd e⇓e2
| b-InL[intro]:   e⇓e' ⟹ InL e ⇓ InL e'
| b-InR[intro]:   e⇓e' ⟹ InR e ⇓ InR e'
| b-CaseL[intro]: ⟦x1#(e,e2,e'',x2); x2#(e,e1,e'',x1) ; e⇓InL e'; e1[x1::=e']⇓e''⟧
                  ⟹ Case e of inl x1 → e1 | inr x2 → e2 ⇓ e''
```

| *b-CaseR*[*intro*]: $[\![x_1\#(e,e_2,e'',x_2);\ x_2\#(e,e_1,e'',x_1)\ ;\ e\Downarrow InR\ e';\ e_2[x_2{::=}e']\Downarrow e'']\!]$
$\Longrightarrow Case\ e\ of\ inl\ x_1 \rightarrow e_1 \mid inr\ x_2 \rightarrow e_2 \Downarrow e''$

nominal-inductive *big*

lemma *fresh-preserved*:
fixes $x{::}name$
fixes $t{::}trm$
fixes $t'{::}trm$
assumes $t \Downarrow t'$ **and** $x\#t$
shows $x\#t'$
using *assms* **by** (*induct*) (*auto simp add*:*fresh-subst'*)

4.1 Inversion lemmas for the big step relation

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

inductive-cases2 *b-App-inv-auto*[*elim*]: $App\ e_1\ e_2 \Downarrow t$
inductive-cases2 *b-Case-inv-auto*[*elim*]: $Case\ e\ of\ inl\ x_1 \rightarrow e_1 \mid inr\ x_2 \rightarrow e_2 \Downarrow t$
inductive-cases2 *b-Lam-inv-auto*[*elim*]: $Lam[x].t \Downarrow t$
inductive-cases2 *b-Const-inv-auto*[*elim*]: $Const\ n \Downarrow t$
inductive-cases2 *b-Fst-inv-auto*[*elim*]: $Fst\ e \Downarrow t$
inductive-cases2 *b-Snd-inv-auto*[*elim*]: $Snd\ e \Downarrow t$
inductive-cases2 *b-InL-inv-auto*[*elim*]: $InL\ e \Downarrow t$
inductive-cases2 *b-InR-inv-auto*[*elim*]: $InR\ e \Downarrow t$
inductive-cases2 *b-Pr-inv-auto*[*elim*]: $Pr\ e_1\ e_2 \Downarrow t$

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

4.2 Strong induction principle for the big step relation

lemma *big-induct-strong*
[*consumes 1*, *case-names b-Lam b-App b-Const b-Pr b-Fst b-Snd b-InL b-InR b-CaseL b-CaseR*]:
fixes $P{::}'a{::}fs\text{-}name \Rightarrow trm \Rightarrow trm \Rightarrow bool$
and $x :: 'a{::}fs\text{-}name$
assumes a: $t \Downarrow t'$
and $a1$: $\bigwedge x\ e\ c.\ P\ c\ (Lam\ [x].e)\ (Lam\ [x].e)$
and $a2$: $\bigwedge x\ e_1\ e_2\ e_2'\ e'\ e_1'\ c.$
$[\![x\#(e_1,e_2,e',c);\ e_1\Downarrow Lam\ [x].e_1';\ (\bigwedge c.\ P\ c\ e_1\ (Lam\ [x].e_1'));$
$e_2\Downarrow e_2';\ (\bigwedge c.\ P\ c\ e_2\ e_2');\ \ e_1'[x{::=}e_2']\Downarrow e';\ (\bigwedge c.\ P\ c\ (e_1'[x{::=}e_2'])\ e')]\!]$
$\Longrightarrow P\ c\ (App\ e_1\ e_2)\ e'$
and $a3$: $\bigwedge n\ c.\ P\ c\ (Const\ n)\ (Const\ n)$
and $a4$: $\bigwedge e_1\ e_1'\ e_2\ e_2'\ c.$
$[\![e_1 \Downarrow e_1';\ (\bigwedge c.\ P\ c\ e_1\ e_1');\ e_2 \Downarrow e_2';\ (\bigwedge c.\ P\ c\ e_2\ e_2')]\!]$
$\Longrightarrow P\ c\ (Pr\ e_1\ e_2)\ (Pr\ e_1'\ e_2')$
and $a5$: $\bigwedge e\ e_1\ e_2\ c.\ [\![e \Downarrow Pr\ e_1\ e_2;\ (\bigwedge c.\ P\ c\ e\ (Pr\ e_1\ e_2))]\!] \Longrightarrow P\ c\ (Fst\ e)\ e_1$
and $a6$: $\bigwedge e\ e_1\ e_2\ c.\ [\![e \Downarrow Pr\ e_1\ e_2;\ (\bigwedge c.\ P\ c\ e\ (Pr\ e_1\ e_2))]\!] \Longrightarrow P\ c\ (Snd\ e)\ e_2$
and $a7$: $\bigwedge e\ e'\ c.\ [\![e \Downarrow e';\ (\bigwedge c.\ P\ c\ e\ e')]\!] \Longrightarrow P\ c\ (InL\ e)\ (InL\ e')$
and $a8$: $\bigwedge e\ e'\ c.\ [\![e \Downarrow e';\ (\bigwedge c.\ P\ c\ e\ e')]\!] \Longrightarrow P\ c\ (InR\ e)\ (InR\ e')$

```
  and a9: ⋀x1 e e2 e'' x2 e1 e' c.
          ⟦x1#(e,e2,e'',x2,c); x2#(e,e1,e'',x1,c);  e ⇓ InL e'; (⋀c. P c e (InL e'));
          e1[x1::=e'] ⇓ e''; (⋀c. P c (e1[x1::=e']) e'')⟧
          ⟹ P c (Case e of inl x1 → e1 | inr x2 → e2) e''
  and a10:⋀x1 e e2 e'' x2 e1 e' c.
          ⟦x1#(e,e2,e'',x2,c); x2#(e,e1,e'',x1,c); e ⇓ InR e'; (⋀c. P c e (InR e'));
          e2[x2::=e'] ⇓ e''; (⋀c. P c (e2[x2::=e']) e'')⟧
          ⟹ P c (Case e of inl x1 → e1 | inr x2 → e2) e''
  shows P c t t'
proof −
 from a have ⋀(π::name prm) c. P c (π•t) (π•t')
 proof (induct)
  case (b-Lam x e π c)
  show P c (π•(Lam [x].e)) (π•(Lam [x].e)) using a1 by simp
 next
  case (b-App x e1 e2 e' e1' e2' π c)
  obtain y::name where fs: y#(π•x,π•e1,π•e2,π•e2',π•e',π•e1,c) by (erule exists-fresh[OF fs-name1])
  let ?sw = [(π•x,y)]
  let ?pi' = ?sw@π
  have f0: x#(e1,e2,e') by fact
  have f1: (π•x)#(π•e1,π•e2,π•e') using f0 by (simp add: fresh-bij)
  have f2: y#(?pi'•e1,?pi'•e2,?pi'•e') using f0
   by (auto simp add: pt-name2 fresh-left calc-atm perm-pi-simp fresh-prod)
  have p1: e1 ⇓ Lam [x].e1' by fact
  then have (?pi'•e1)⇓(?pi'•Lam [x].e1') by (simp only: big-eqvt)
  moreover
  have p2: e2 ⇓ e2' by fact
  then have (?pi'•e2) ⇓ (?pi'•e2') by (simp only: big-eqvt)
  moreover
  have p3: e1'[x::=e2'] ⇓ e' by fact
  then have (?pi'•(e1'[x::=e2'])) ⇓ (?pi'•e') by (simp only: big-eqvt)
  then have (?pi'•e1')[y::=(?pi'•e2')] ⇓ (?pi'•e') by (simp add: subst-eqvt calc-atm)
  moreover
  have ih1: ⋀c. P c (?pi'•e1) (?pi'•(Lam [x].e1')) by fact
  then have ⋀c. P c (?pi'•e1) (Lam [y].(?pi'•e1')) by (simp add: calc-atm)
  moreover
  have ih2: ⋀c. P c (?pi'•e2) (?pi'•e2') by fact
  moreover
  have ih3: ⋀c. P c (?pi'•(e1'[x::=e2'])) (?pi'•e') by fact
  then have ⋀c. P c ((?pi'•e1')[y::=(?pi'•e2')]) (?pi'•e') by (simp add: calc-atm subst-eqvt)
  ultimately have P c (App (?pi'•e1) (?pi'•e2)) (?pi'•e') using fs f2
   by (auto intro!: a2 simp add: calc-atm)
  then have P c (?sw•(App (π•e1) (π•e2))) (?sw•(π•e'))
   by (simp del: append-Cons add: pt-name2)
  moreover have (?sw•(App (π•e1) (π•e2))) = App (π•e1) (π•e2)
   by (rule perm-fresh-fresh) (simp-all add: fs f1)
  moreover have (?sw•(π•e')) = π•e'
   by (rule perm-fresh-fresh) (simp-all add: fs f1)
  ultimately show P c (π•(App e1 e2)) (π•e')
   by simp
 next
  case (b-Const n π c)
  show P c (π•(Const n)) (π•(Const n)) using a3 by simp
```

next
  case $(b\text{-}Pr\ e_1\ e_1{}'\ e_2\ e_2{}'\ \pi\ c)$
  then show $P\ c\ (\pi{\bullet}(Pr\ e_1\ e_2))\ (\pi{\bullet}(Pr\ e_1{}'\ e_2{}'))$ **using** *a4*
    by (*simp*, *blast intro*: *big-eqvt*)
next
  case $(b\text{-}Fst\ e\ e_1\ e_2\ \pi\ c)$
  have *p1*: $e \Downarrow Pr\ e_1\ e_2$ **by** *fact*
  then have $(\pi{\bullet}e){\Downarrow}(\pi{\bullet}(Pr\ e_1\ e_2))$ **by** (*simp only*: *big-eqvt*)
  moreover
  have *ih1*: $\bigwedge c.\ P\ c\ (\pi{\bullet}e)\ (\pi{\bullet}(Pr\ e_1\ e_2))$ **by** *fact*
  ultimately show $P\ c\ (\pi{\bullet}(Fst\ e))\ (\pi{\bullet}e_1)$ **using** *a5* **by** *simp*
next
  case $(b\text{-}Snd\ e\ e_1\ e_2\ \pi\ c)$
  have *p1*: $e \Downarrow Pr\ e_1\ e_2$ **by** *fact*
  then have $(\pi{\bullet}e){\Downarrow}(\pi{\bullet}(Pr\ e_1\ e_2))$ **by** (*simp only*: *big-eqvt*)
  moreover
  have *ih1*: $\bigwedge c.\ P\ c\ (\pi{\bullet}e)\ (\pi{\bullet}(Pr\ e_1\ e_2))$ **by** *fact*
  ultimately show $P\ c\ (\pi{\bullet}(Snd\ e))\ (\pi{\bullet}e_2)$ **using** *a6* **by** *simp*
next
  case $(b\text{-}InL\ e\ e'\ \pi\ c)$
  then show $P\ c\ (\pi{\bullet}(InL\ e))\ (\pi{\bullet}(InL\ e'))$ **using** *a7*
    by (*simp*, *blast intro*: *big-eqvt*)
next
  case $(b\text{-}InR\ e\ e'\ \pi\ c)$
  then show $P\ c\ (\pi{\bullet}(InR\ e))\ (\pi{\bullet}(InR\ e'))$ **using** *a8*
    by (*simp*, *blast intro*: *big-eqvt*)
next
  case $(b\text{-}CaseL\ x_1\ e\ e_2\ e''\ x_2\ e_1\ e'\ \pi\ c)$
  obtain y_1::*name* **where** *fs1*: $y_1\#(\pi{\bullet}x_1,\pi{\bullet}e,\pi{\bullet}e_1,\pi{\bullet}e_2,\pi{\bullet}e'',\pi{\bullet}x_2,c)$
    by (*rule exists-fresh*[*OF fs-name1*])
  obtain y_2::*name* **where** *fs2*: $y_2\#(\pi{\bullet}x_2,\pi{\bullet}e,\pi{\bullet}e_1,\pi{\bullet}e_2,\pi{\bullet}e'',\pi{\bullet}x_1,c,y_1)$
    by (*rule exists-fresh*[*OF fs-name1*])
  let *?sw1* = $[(\pi{\bullet}x_1,y_1)]$
  let *?sw2* = $[(\pi{\bullet}x_2,y_2)]$
  let *?pi'* = *?sw2*@*?sw1*@π
  have *f01*: $x_1\#(e,e_2,e'',x_2)$ **by** *fact*
  have *f11*: $(\pi{\bullet}x_1)\#(\pi{\bullet}e,\pi{\bullet}e_2,\pi{\bullet}e'',\pi{\bullet}x_2)$ **using** *f01* **by** (*simp add*: *fresh-bij*)
  have *f21*: $y_1\#(?pi'{\bullet}e,?pi'{\bullet}e_2,?pi'{\bullet}e'')$ **using** *f01 fs1 fs2*
    by (*simp add*: *fresh-atm fresh-prod fresh-left calc-atm pt-name2 perm-pi-simp*)
  have *f02*: $x_2\#(e,e_1,e'',x_1)$ **by** *fact*
  have *f12*: $(\pi{\bullet}x_2)\#(\pi{\bullet}e,\pi{\bullet}e_1,\pi{\bullet}e'',\pi{\bullet}x_1)$ **using** *f02* **by** (*simp add*: *fresh-bij*)
  have *f22*: $y_2\#(?pi'{\bullet}e,?pi'{\bullet}e_1,?pi'{\bullet}e'')$ **using** *f02 fs1 fs2*
    by (*auto simp add*: *fresh-atm fresh-prod fresh-left calc-atm pt-name2 perm-pi-simp*)
  have *p1*: $e \Downarrow InL\ e'$ **by** *fact*
  then have $(?pi'{\bullet}e) \Downarrow (?pi'{\bullet}(InL\ e'))$ **by** (*simp only*: *big-eqvt*)
  moreover
  have *p2*: $e_1[x_1::=e'] \Downarrow e''$ **by** *fact*
  then have $(?pi'{\bullet}(e_1[x_1::=e'])) \Downarrow (?pi'{\bullet}e'')$ **by** (*simp only*: *big-eqvt*)
  then have $(?pi'{\bullet}e_1)[y_1::=(?pi'{\bullet}e')] \Downarrow (?pi'{\bullet}e'')$ **using** *fs1 fs2*
    by (*auto simp add*: *calc-atm subst-eqvt fresh-prod fresh-atm del*: *append-Cons*)
  moreover
  have *ih1*: $\bigwedge c.\ P\ c\ (?pi'{\bullet}e)\ (?pi'{\bullet}(InL\ e'))$ **by** *fact*
  moreover

have *ih2*: ⋀*c*. *P c* (*?pi′*•(*e*₁[*x*₁::=*e′*])) (*?pi′*•*e′′*) **by** *fact*
then have ⋀*c*. *P c* ((*?pi′*•*e*₁)[*y*₁::=(*?pi′*•*e′*)]) (*?pi′*•*e′′*) **using** *fs1 fs2*
by (*auto simp add*: *calc-atm subst-eqvt fresh-prod fresh-atm del*: *append-Cons*)
ultimately have *P c* (*Case* (*?pi′*•*e*) *of inl y*₁ → (*?pi′*•*e*₁) | *inr y*₂ → (*?pi′*•*e*₂)) (*?pi′*•*e′′*)
using *f21 f22 fs1 fs2* **by** (*auto intro*!: *a9 simp add*: *fresh-atm fresh-prod*)
then have *P c* (*?sw2*•*?sw1*•(*Case* (π•*e*) *of inl* (π•*x*₁) → (π•*e*₁) | *inr* (π•*x*₂) → (π•*e*₂)))
(*?sw2*•*?sw1*•(π•*e′′*))
using *fs1 fs2 f01 f02 f11 f12*
by (*auto simp del*: *append-Cons simp add*: *pt-name2 fresh-atm fresh-prod calc-atm*)
moreover have *?sw1*•(*Case* (π•*e*) *of inl* (π•*x*₁) → (π•*e*₁) | *inr* (π•*x*₂) → (π•*e*₂))
= (*Case* (π•*e*) *of inl* (π•*x*₁) → (π•*e*₁) | *inr* (π•*x*₂) → (π•*e*₂))
by (*rule perm-fresh-fresh*) (*simp-all add*: *fs1 fs2 f11 f12 abs-fresh*)
moreover have *?sw2*•(*Case* (π•*e*) *of inl* (π•*x*₁) → (π•*e*₁) | *inr* (π•*x*₂) → (π•*e*₂))
= (*Case* (π•*e*) *of inl* (π•*x*₁) → (π•*e*₁) | *inr* (π•*x*₂) → (π•*e*₂))
by (*rule perm-fresh-fresh*) (*simp-all add*: *fs1 fs2 f11 f12 abs-fresh*)
moreover have (*?sw1*•(π•*e′′*)) = (π•*e′′*)
by (*rule perm-fresh-fresh*) (*simp-all add*: *fs1 fs2 f11 f12*)
moreover have (*?sw2*•(π•*e′′*)) = (π•*e′′*)
by (*rule perm-fresh-fresh*) (*simp-all add*: *fs1 fs2 f11 f12*)
ultimately show *P c* (π•(*Case e of inl x*₁ → *e*₁ | *inr x*₂ → *e*₂)) (π•*e′′*)
by (*simp only*:, *simp*)
next
case (*b-CaseR x*₁ *e e*₂ *e′′ x*₂ *e*₁ *e′* π *c*)
obtain *y*₁::*name* **where** *fs1*: *y*₁#(π•*x*₁,π•*e*,π•*e*₁,π•*e*₂,π•*e′′*,π•*x*₂,*c*)
by (*rule exists-fresh*[*OF fs-name1*])
obtain *y*₂::*name* **where** *fs2*: *y*₂#(π•*x*₂,π•*e*,π•*e*₁,π•*e*₂,π•*e′′*,π•*x*₁,*c*,*y*₁)
by (*rule exists-fresh*[*OF fs-name1*])
let *?sw1* = [(π•*x*₁,*y*₁)]
let *?sw2* = [(π•*x*₂,*y*₂)]
let *?pi′* = *?sw2*@*?sw1*@π
have *f01*: *x*₁#(*e*,*e*₂,*e′′*,*x*₂) **by** *fact*
have *f11*: (π•*x*₁)#(π•*e*,π•*e*₂,π•*e′′*,π•*x*₂) **using** *f01* **by** (*simp add*: *fresh-bij*)
have *f21*: *y*₁#(*?pi′*•*e*,*?pi′*•*e*₂,*?pi′*•*e′′*) **using** *f01 fs1 fs2*
by (*simp add*: *fresh-atm fresh-prod fresh-left calc-atm pt-name2 perm-pi-simp*)
have *f02*: *x*₂#(*e*,*e*₁,*e′′*,*x*₁) **by** *fact*
have *f12*: (π•*x*₂)#(π•*e*,π•*e*₁,π•*e′′*,π•*x*₁) **using** *f02* **by** (*simp add*: *fresh-bij*)
have *f22*: *y*₂#(*?pi′*•*e*,*?pi′*•*e*₁,*?pi′*•*e′′*) **using** *f02 fs1 fs2*
by (*auto simp add*: *fresh-atm fresh-prod fresh-left calc-atm pt-name2 perm-pi-simp*)
have *p1*: *e* ⇓ *InR e′* **by** *fact*
then have (*?pi′*•*e*) ⇓ (*?pi′*•(*InR e′*)) **by** (*simp only*: *big-eqvt*)
moreover
have *p2*: *e*₂[*x*₂::=*e′*] ⇓ *e′′* **by** *fact*
then have (*?pi′*•(*e*₂[*x*₂::=*e′*])) ⇓ (*?pi′*•*e′′*) **by** (*simp only*: *big-eqvt*)
then have (*?pi′*•*e*₂)[*y*₂::=(*?pi′*•*e′*)] ⇓ (*?pi′*•*e′′*) **using** *fs1 fs2 f11 f12*
by (*auto simp add*: *calc-atm subst-eqvt fresh-prod fresh-atm del*: *append-Cons*)
moreover
have *ih1*: ⋀*c*. *P c* (*?pi′*•*e*) (*?pi′*•(*InR e′*)) **by** *fact*
moreover
have *ih2*: ⋀*c*. *P c* (*?pi′*•(*e*₂[*x*₂::=*e′*])) (*?pi′*•*e′′*) **by** *fact*
then have ⋀*c*. *P c* ((*?pi′*•*e*₂)[*y*₂::=(*?pi′*•*e′*)]) (*?pi′*•*e′′*) **using** *fs1 fs2 f11 f12*
by (*auto simp add*: *calc-atm subst-eqvt fresh-prod fresh-atm del*: *append-Cons*)
ultimately have *P c* (*Case* (*?pi′*•*e*) *of inl y*₁ → (*?pi′*•*e*₁) | *inr y*₂ → (*?pi′*•*e*₂)) (*?pi′*•*e′′*)
using *f21 f22 fs1 fs2* **by** (*auto intro*!: *a10 simp add*: *fresh-atm fresh-prod*)

then have $P\ c\ (?sw2 \cdot ?sw1 \cdot (Case\ (\pi \cdot e)\ of\ inl\ (\pi \cdot x_1) \rightarrow (\pi \cdot e_1)\ |\ inr\ (\pi \cdot x_2) \rightarrow (\pi \cdot e_2)))$
$(?sw2 \cdot ?sw1 \cdot (\pi \cdot e''))$
using *fs1 fs2 f01 f02 f11 f12*
by (*auto simp del*: *append-Cons simp add*: *pt-name2 fresh-atm fresh-prod calc-atm*)
moreover have $?sw1 \cdot (Case\ (\pi \cdot e)\ of\ inl\ (\pi \cdot x_1) \rightarrow (\pi \cdot e_1)\ |\ inr\ (\pi \cdot x_2) \rightarrow (\pi \cdot e_2))$
$= (Case\ (\pi \cdot e)\ of\ inl\ (\pi \cdot x_1) \rightarrow (\pi \cdot e_1)\ |\ inr\ (\pi \cdot x_2) \rightarrow (\pi \cdot e_2))$
by (*rule perm-fresh-fresh*) (*simp-all add*: *fs1 fs2 f11 f12 abs-fresh*)
moreover have $?sw2 \cdot (Case\ (\pi \cdot e)\ of\ inl\ (\pi \cdot x_1) \rightarrow (\pi \cdot e_1)\ |\ inr\ (\pi \cdot x_2) \rightarrow (\pi \cdot e_2))$
$= (Case\ (\pi \cdot e)\ of\ inl\ (\pi \cdot x_1) \rightarrow (\pi \cdot e_1)\ |\ inr\ (\pi \cdot x_2) \rightarrow (\pi \cdot e_2))$
by (*rule perm-fresh-fresh*) (*simp-all add*: *fs1 fs2 f11 f12 abs-fresh*)
moreover have $(?sw1 \cdot (\pi \cdot e'')) = (\pi \cdot e'')$
by (*rule perm-fresh-fresh*) (*simp-all add*: *fs1 fs2 f11 f12*)
moreover have $(?sw2 \cdot (\pi \cdot e'')) = (\pi \cdot e'')$
by (*rule perm-fresh-fresh*) (*simp-all add*: *fs1 fs2 f11 f12*)
ultimately show $P\ c\ (\pi \cdot (Case\ e\ of\ inl\ x_1 \rightarrow e_1\ |\ inr\ x_2 \rightarrow e_2))\ (\pi \cdot e'')$
by (*simp only*:, *simp*)
qed
then have $P\ c\ (([]::name\ prm) \cdot t)\ (([]::name\ prm) \cdot t')$ **by** *blast*
then show $P\ c\ t\ t'$ **by** *simp*
qed

4.3 Strong inversion lemmas for big step relation

lemma *b-App-elim*[*elim*]:
assumes $App\ e_1\ e_2 \Downarrow e'$ **and** $x\#(e_1,e_2,e')$
obtains f_1 **and** f_2 **where** $e_1 \Downarrow Lam\ [x].\ f_1\ \ e_2 \Downarrow f_2\ \ f_1[x::=f_2] \Downarrow e'$
using *assms*
apply –
apply(*erule b-App-inv-auto*)
apply(*drule-tac pi*=[(*xa*,*x*)] **in** *big-eqvt*)
apply(*drule-tac pi*=[(*xa*,*x*)] **in** *big-eqvt*)
apply(*drule-tac pi*=[(*xa*,*x*)] **in** *big-eqvt*)
apply(*perm-simp add*: *calc-atm eqvt*)
done

lemma *b-CaseL-elim*[*elim*]:
assumes $Case\ e\ of\ inl\ x_1 \rightarrow e_1\ |\ inr\ x_2 \rightarrow e_2 \Downarrow e''$ **and** $(\bigwedge\ t.\ \neg\ \ e \Downarrow InR\ t)$
obtains e' **where** $e \Downarrow InL\ e'$ **and** $e_1[x_1::=e'] \Downarrow e''$
using *assms*
apply –
apply (*rule b-Case-inv-auto*, *auto*)
apply(*drule-tac u*=e' **in** *subst-fun-eq*)
apply(*simp*)
done

lemma *b-CaseR-elim*[*elim*]:
assumes $Case\ e\ of\ inl\ x_1 \rightarrow e_1\ |\ inr\ x_2 \rightarrow e_2 \Downarrow e''$ **and** $\bigwedge\ t.\ \neg\ e \Downarrow InL\ t$
obtains e' **where** $e \Downarrow InR\ e'$ **and** $e_2[x_2::=e'] \Downarrow e''$
using *assms*
apply –
apply (*rule b-Case-inv-auto*, *auto*)
apply(*drule-tac u*=e' **in** *subst-fun-eq*)+
apply(*simp*)

done

5 Values.

5.1 Definition of values.

inductive2
val :: *trm*⇒*bool*
where
v-Lam[*intro*]: *val* (*Lam* [*x*].*e*)
| *v-Const*[*intro*]: *val* (*Const n*)
| *v-Pr*[*intro*]: ⟦*val* e_1; *val* e_2⟧ ⟹ *val* (*Pr* e_1 e_2)
| *v-InL*[*intro*]: *val e* ⟹ *val* (*InL e*)
| *v-InR*[*intro*]: *val e* ⟹ *val* (*InR e*)

5.2 Inversion lemmas for values

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

inductive-cases2 *v-Const-inv-auto*[*elim*]: *val* (*Const n*)
inductive-cases2 *v-Pr-inv-auto*[*elim*]: *val* (*Pr* e_1 e_2)
inductive-cases2 *v-InL-inv-auto*[*elim*]: *val* (*InL e*)
inductive-cases2 *v-InR-inv-auto*[*elim*]: *val* (*InR e*)
inductive-cases2 *v-Fst-inv-auto*[*elim*]: *val* (*Fst e*)
inductive-cases2 *v-Snd-inv-auto*[*elim*]: *val* (*Snd e*)
inductive-cases2 *v-Case-inv-auto*[*elim*]: *val* (*Case e of inl* x_1 → e_1 | *inr* x_2 → e_2)
inductive-cases2 *v-Var-inv-auto*[*elim*]: *val* (*Var x*)
inductive-cases2 *v-Lam-inv-auto*[*elim*]: *val* (*Lam* [*x*].*e*)
inductive-cases2 *v-App-inv-auto*[*elim*]: *val* (*App* e_1 e_2)

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

6 Weakening lemma

lemma *weakening*:
assumes $\Gamma_1 \vdash e\colon T$ **and** *valid* Γ_2 **and** $\Gamma_1 \ll \Gamma_2$
shows $\Gamma_2 \vdash e\colon T$
using *assms*
proof(*nominal-induct* Γ_1 *e T avoiding*: Γ_2 *rule*: *typing-induct-strong*)
case (*t-Lam x* Γ_1 T_1 *t* T_2 Γ_2)
have *ih*: ⟦*valid* $((x,T_1)\#\Gamma_2)$; $(x,T_1)\#\Gamma_1 \ll (x,T_1)\#\Gamma_2$⟧ ⟹ $(x,T_1)\#\Gamma_2 \vdash t : T_2$ **by** *fact*
have *H1*: *valid* Γ_2 **by** *fact*
have *H2*: $\Gamma_1 \ll \Gamma_2$ **by** *fact*
have *fs*: $x\#\Gamma_2$ **by** *fact*
then have *valid* $((x,T_1)\#\Gamma_2)$ **using** *H1* **by** *auto*
moreover have $(x,T_1)\#\Gamma_1 \ll (x,T_1)\#\Gamma_2$ **using** *H2* **by** *auto*
ultimately have $(x,T_1)\#\Gamma_2 \vdash t : T_2$ **using** *ih* **by** *simp*
thus $\Gamma_2 \vdash$ *Lam* [*x*].*t* : $T_1 \rightarrow T_2$ **using** *fs* **by** *auto*

next
  case ($t\text{-}Case\ x_1\ \Gamma_1\ e\ e_2\ x_2\ e_1\ S_1\ S_2\ T\ \Gamma_2$)
  then have ih_1: $valid\ ((x_1,Data\ S_1)\#\Gamma_2) \Longrightarrow (x_1,Data\ S_1)\#\Gamma_2 \vdash e_1 : T$
    and ih_2: $valid\ ((x_2,Data\ S_2)\#\Gamma_2) \Longrightarrow (x_2,Data\ S_2)\#\Gamma_2 \vdash e_2 : T$
    and ih_3: $\Gamma_2 \vdash e : Data\ (DSum\ S_1\ S_2)$ **by** *auto*
  have fs_1: $x_1\#\Gamma_2\ x_1\#e\ x_1\#e_2\ x_1\#x_2$ **by** *fact*
  have fs_2: $x_2\#\Gamma_2\ x_2\#e\ x_2\#e_1\ x_2\#x_1$ **by** *fact*
  have $valid\ \Gamma_2$ **by** *fact*
  then have $valid\ ((x_1,Data\ S_1)\#\Gamma_2)$ **and** $valid\ ((x_2,Data\ S_2)\#\Gamma_2)$ **using** $fs_1\ fs_2$ **by** *auto*
  then have $(x_1, Data\ S_1)\#\Gamma_2 \vdash e_1 : T$ **and** $(x_2, Data\ S_2)\#\Gamma_2 \vdash e_2 : T$ **using** $ih_1\ ih_2$ **by** *simp-all*
  with ih_3 **show** $\Gamma_2 \vdash Case\ e\ of\ inl\ x_1 \rightarrow e_1 \mid inr\ x_2 \rightarrow e_2 : T$ **using** $fs_1\ fs_2$ **by** *auto*
qed (*auto*)

A corollary of the weakening lemma.

lemma *context-exchange*:
  assumes a: $(x_1,T_1)\#(x_2,T_2)\#\Gamma \vdash t : T$
  shows $(x_2,T_2)\#(x_1,T_1)\#\Gamma \vdash t : T$
proof −
  from a **have** $valid\ ((x_1,T_1)\#(x_2,T_2)\#\Gamma)$ **by** (*simp add*: *typing-valid*)
  then have $x_1{\neq}x_2\ x_1\#\Gamma\ x_2\#\Gamma\ valid\ \Gamma$
    by (*auto simp*: *fresh-list-cons fresh-atm*[*symmetric*])
  then have $valid\ ((x_2,T_2)\#(x_1,T_1)\#\Gamma)$
    by (*auto simp*: *fresh-list-cons fresh-atm*)
  moreover
  have $(x_1,T_1)\#(x_2,T_2)\#\Gamma \ll (x_2,T_2)\#(x_1,T_1)\#\Gamma$ **by** *auto*
  ultimately show $(x_2,T_2)\#(x_1,T_1)\#\Gamma \vdash t : T$ **using** a **by** (*auto intro*: *weakening*)
qed

lemma *typing-var-unicity*:
  assumes $(x,t_1)\#\Gamma \vdash Var\ x : t_2$
  shows $t_1{=}t_2$
proof −
  have $(x,t_2) \in set\ ((x,t_1)\#\Gamma)$ **and** $valid\ ((x,t_1)\#\Gamma)$ **using** *assms* **by** *auto*
  thus $t_1{=}t_2$ **by** (*simp only*: *type-unicity-in-context*)
qed

7 Substitution lemma

lemma *typing-substitution*:
  fixes $\Gamma{::}(name \times ty)\ list$
  assumes $(x,T')\#\Gamma \vdash e : T$
  and $\Gamma \vdash e' : T'$
  shows $\Gamma \vdash e[x{::=}e'] : T$
  using *assms*
proof (*nominal-induct* e *avoiding*: $\Gamma\ e'\ x$ *arbitrary*: T *rule*: *trm.induct*)
  case ($Var\ y\ \Gamma\ e'\ x\ T$)
  have $h1$: $(x,T')\#\Gamma \vdash Var\ y : T$ **by** *fact*
  have $h2$: $\Gamma \vdash e' : T'$ **by** *fact*
  show $\Gamma \vdash (Var\ y)[x{::=}e'] : T$
  proof (*cases* $x{=}y$)
    case *True*
    assume as: $x{=}y$

then have $T{=}T'$ **using** *h1 typing-var-unicity* **by** *auto*
then show $\Gamma \vdash (Var\ y)[x{::=}e'] : T$ **using** *as h2* **by** *simp*
next
case *False*
assume *as*: $x{\neq}y$
have $(y,T) \in set\ ((x,T')\#\Gamma)$ **using** *h1* **by** *auto*
then have $(y,T) \in set\ \Gamma$ **using** *as* **by** *auto*
moreover
have *valid* Γ **using** *h2* **by** (*simp only*: *typing-valid*)
ultimately show $\Gamma \vdash (Var\ y)[x{::=}e'] : T$ **using** *as* **by** (*simp add*: *t-Var*)
qed
next
case $(Lam\ y\ t\ \Gamma\ e'\ x\ T)$
have *vc*: $y\#\Gamma\ y\#x\ y\#e'$ **by** *fact*
have *pr1*: $\Gamma \vdash e' : T'$ **by** *fact*
have *pr2*: $(x,T')\#\Gamma \vdash Lam\ [y].t : T$ **by** *fact*
then obtain $T_1\ T_2$ **where** *pr2'*: $(y,T_1)\#(x,T')\#\Gamma \vdash t : T_2$ **and** *eq*: $T = T_1 \rightarrow T_2$
using *vc* **by** (*auto simp add*: *fresh-list-cons*)
then have *pr2''*:$(x,T')\#(y,T_1)\#\Gamma \vdash t : T_2$ **by** (*simp add*: *context-exchange*)
have *ih*: $[\![(x,T')\#(y,T_1)\#\Gamma \vdash t : T_2;\ (y,T_1)\#\Gamma \vdash e' : T']\!] \Longrightarrow (y,T_1)\#\Gamma \vdash t[x{::=}e'] : T_2$ **by** *fact*
have *valid* Γ **using** *pr1* **by** (*simp add*: *typing-valid*)
then have *valid* $((y,T_1)\#\Gamma)$ **using** *vc* **by** *auto*
then have $(y,T_1)\#\Gamma \vdash e' : T'$ **using** *pr1* **by** (*auto intro*: *weakening*)
then have $(y,T_1)\#\Gamma \vdash t[x{::=}e'] : T_2$ **using** *ih pr2''* **by** *simp*
then have $\Gamma \vdash Lam\ [y].(t[x{::=}e']) : T_1 \rightarrow T_2$ **using** *vc* **by** (*auto intro*: *t-Lam*)
thus $\Gamma \vdash (Lam\ [y].t)[x{::=}e'] : T$ **using** *vc eq* **by** *simp*
next
case $(Case\ t_1\ x_1\ t_2\ x_2\ t3\ \Gamma\ e'\ x\ T)$
have *vc*: $x_1\#\Gamma\ x_1\#e'\ x_1\#x x_1\#t_1\ x_1\#t3\ x_2\#\Gamma$
$x_2\#e'\ x_2\#x\ \ x_2\#t_1\ x_2\#t_2\ x_2{\neq}x_1$ **by** *fact*
have *as1*: $\Gamma \vdash e' : T'$ **by** *fact*
have *as2*: $(x,T')\#\Gamma \vdash Case\ t_1\ of\ inl\ x_1 \rightarrow t_2 \mid inr\ x_2 \rightarrow t3 : T$ **by** *fact*
then obtain $S_1\ S_2$ **where**
h1:$(x,T')\#\Gamma \vdash t_1 : Data\ (DSum\ S_1\ S_2)$ **and**
h2:$(x_1,Data\ S_1)\#(x,T')\#\Gamma \vdash t_2 : T$ **and**
h3:$(x_2,Data\ S_2)\#(x,T')\#\Gamma \vdash t3 : T$
using *vc* **by** (*auto simp add*: *fresh-list-cons*)
have *ih1*: $[\![(x,T')\#\Gamma \vdash t_1 : T;\ \Gamma \vdash e' : T']\!] \Longrightarrow \Gamma \vdash t_1[x{::=}e'] : T$
and *ih2*: $[\![(x,T')\#(x_1,Data\ S_1)\#\Gamma \vdash t_2{:}T;\ (x_1,Data\ S_1)\#\Gamma \vdash e'{:}T']\!] \Longrightarrow (x_1,Data\ S_1)\#\Gamma \vdash t_2[x{::=}e']{:}T$
and *ih3*: $[\![(x,T')\#(x_2,Data\ S_2)\#\Gamma \vdash t3{:}T;\ (x_2,Data\ S_2)\#\Gamma \vdash e'{:}T']\!] \Longrightarrow (x_2,Data\ S_2)\#\Gamma \vdash t3[x{::=}e']{:}T$
by *fact*
from *h2* **have** *h2'*: $(x,T')\#(x_1,Data\ S_1)\#\Gamma \vdash t_2 : T$ **by** (*rule context-exchange*)
from *h3* **have** *h3'*: $(x,T')\#(x_2,Data\ S_2)\#\Gamma \vdash t3 : T$ **by** (*rule context-exchange*)
have $\Gamma \vdash t_1[x{::=}e'] : Data\ (DSum\ S_1\ S_2)$ **using** *h1 ih1 as1* **by** *simp*
moreover
have *valid* $((x_1,Data\ S_1)\#\Gamma)$ **using** *h2'* **by** (*auto dest*: *typing-valid*)
then have $(x_1,Data\ S_1)\#\Gamma \vdash e' : T'$ **using** *as1* **by** (*auto simp add*: *weakening*)
then have $(x_1,Data\ S_1)\#\Gamma \vdash t_2[x{::=}e'] : T$ **using** *ih2 h2'* **by** *simp*
moreover
have *valid* $((x_2,Data\ S_2)\#\Gamma)$ **using** *h3'* **by** (*auto dest*: *typing-valid*)
then have $(x_2,Data\ S_2)\#\Gamma \vdash e' : T'$ **using** *as1* **by** (*auto simp add*: *weakening*)

then have $(x_2, Data\ S_2)\#\Gamma \vdash t3[x::=e'] : T$ **using** *ih3 h3'* **by** *simp*
ultimately have $\Gamma \vdash Case\ (t_1[x::=e'])\ of\ inl\ x_1 \rightarrow (t_2[x::=e'])\ |\ inr\ x_2 \rightarrow (t3[x::=e']) : T$
using *vc* **by** (*auto simp add*: *fresh-atm fresh-subst*)
thus $\Gamma \vdash (Case\ t_1\ of\ inl\ x_1 \rightarrow t_2\ |\ inr\ x_2 \rightarrow t3)[x::=e'] : T$ **using** *vc* **by** *simp*
qed (*simp*, *fast*)+

8 Subject reduction

lemma *subject-reduction*:
assumes $e \Downarrow e'$ **and** $\Gamma \vdash e : T$
shows $\Gamma \vdash e' : T$
using *assms*
proof (*nominal-induct avoiding*: Γ *arbitrary*: T *rule*: *big-induct-strong*)
case ($b\text{-}App\ x\ e_1\ e_2\ e_2'\ e'\ e\ \Gamma\ T$)
have *vc*: $x\#\Gamma$ **by** *fact*
have $\Gamma \vdash App\ e_1\ e_2 : T$ **by** *fact*
then obtain T' **where**
a1: $\Gamma \vdash e_1 : T' \rightarrow T$ **and**
a2: $\Gamma \vdash e_2 : T'$ **by** *auto*
have *ih1*: $\Gamma \vdash e_1 : T' \rightarrow T \Longrightarrow \Gamma \vdash Lam\ [x].e : T' \rightarrow T$ **by** *fact*
have *ih2*: $\Gamma \vdash e_2 : T' \Longrightarrow \Gamma \vdash e_2' : T'$ **by** *fact*
have *ih3*: $\Gamma \vdash e[x::=e_2'] : T \Longrightarrow \Gamma \vdash e' : T$ **by** *fact*
have $\Gamma \vdash Lam\ [x].e : T' \rightarrow T$ **using** *ih1 a1* **by** *simp*
then have $((x,T')\#\Gamma) \vdash e : T$ **using** *vc* **by** (*auto simp add*: *ty.inject*)
moreover
have $\Gamma \vdash e_2'$: T' **using** *ih2 a2* **by** *simp*
ultimately have $\Gamma \vdash e[x::=e_2'] : T$ **by** (*simp add*: *typing-substitution*)
thus $\Gamma \vdash e' : T$ **using** *ih3* **by** *simp*
next
case ($b\text{-}CaseL\ x_1\ e\ e_2\ e''\ x_2\ e_1\ e'\ \Gamma$)
have *vc*: $x_1\#\Gamma\ x_2\#\Gamma$ **by** *fact*
have $\Gamma \vdash Case\ e\ of\ inl\ x_1 \rightarrow e_1\ |\ inr\ x_2 \rightarrow e_2 : T$ **by** *fact*
then obtain $S_1\ S_2\ e_1'\ e_2'$ **where**
a1: $\Gamma \vdash e : Data\ (DSum\ S_1\ S_2)$ **and**
a2: $((x_1, Data\ S_1)\#\Gamma) \vdash e_1 : T$ **using** *vc* **by** *auto*
have *ih1*: $\Gamma \vdash e : Data\ (DSum\ S_1\ S_2) \Longrightarrow \Gamma \vdash InL\ e' : Data\ (DSum\ S_1\ S_2)$ **by** *fact*
have *ih2*: $\Gamma \vdash e_1[x_1::=e'] : T \Longrightarrow \Gamma \vdash e'' : T$ **by** *fact*
have $\Gamma \vdash InL\ e' : Data\ (DSum\ S_1\ S_2)$ **using** *ih1 a1* **by** *simp*
then have $\Gamma \vdash e' : Data\ S_1$ **by** *auto*
then have $\Gamma \vdash e_1[x_1::=e'] : T$ **using** *a2* **by** (*simp add*: *typing-substitution*)
then show $\Gamma \vdash e'' : T$ **using** *ih2* **by** *simp*
next
case ($b\text{-}CaseR\ x_1\ e\ e_2\ e''\ x_2\ e_1\ e'\ \Gamma\ T$)
then show $\Gamma \vdash e'' : T$ **by** (*blast intro*: *typing-substitution*)
qed (*blast*)+

9 Examples

In this section we follow the challenge and try the definition of our semantic on some concrete examples. There is no way to effectively execute inductive relation in Isabelle so we use the automatic tactics.

9.1 A solution to Challenge 5

```
lemma challenge-5:
  assumes x≠y
  shows App (App (Lam [x].(Lam [y].Var y)) (Const n₁)) (Const n₂) ⇓ (Const n₂)
  using assms
  by (auto intro!: big.intros simp add: forget abs-fresh fresh-atm fresh-nat)
```

9.2 A solution to Challenge 6

```
lemma challenge-6:
 shows Fst (App (Lam [x].Pr (Var x) (Var x)) (Const n)) ⇓ Const n
  by (auto intro!: big.intros) (simp add: fresh-nat abs-fresh)
```

10 Unicity of evaluation

```
lemma challenge-4-unicity:
  assumes e ⇓ e₁ and e ⇓ e₂
  shows e₁ = e₂
  using assms
proof (induct arbitrary: e₂)
  case (b-Lam x e t₂)
  have Lam [x].e ⇓ t₂ by fact
  thus Lam [x].e = t₂ by (cases, simp-all add: trm.inject)
next
  case (b-Fst e e₁ e₂ t₂)
  have Fst e ⇓ t₂ by fact
  then obtain e₁' e₂' where e ⇓ Pr e₁' e₂' and eq: t₂ = e₁' by auto
  then have Pr e₁ e₂ = Pr e₁' e₂' by auto
  thus e₁ = t₂ using eq by (simp add: trm.inject)
next
  case (b-Snd e e₁ e₂ t₂)
  thus ?case by (force simp add: trm.inject)
next
  case (b-App x e₁ e₂ e' e₁' e₂' t₂)
  have e₁ ⇓ Lam [x].e₁' by fact
  have ih1: ⋀t. e₁ ⇓ t ⟹ Lam [x].e₁' = t by fact
  have e₂ ⇓ e₂' by fact
  have ih2:⋀t. e₂ ⇓ t ⟹ e₂' = t by fact
  have e₁'[x::=e₂'] ⇓ e' by fact
  have ih3: ⋀t. e₁'[x::=e₂'] ⇓ t ⟹ e' = t by fact
  have f:x#(e₁,e₂,e') by fact
  then have x # App e₁ e₂ by auto
  moreover
  have app:App e₁ e₂ ⇓ t₂ by fact
  ultimately have x#t₂ using fresh-preserved by blast
  then have x#(e₁,e₂,t₂) using f by auto
  then obtain f₁'' f₂'' where x1:e₁ ⇓ Lam [x]. f₁'' and x2:e₂ ⇓ f₂'' and x3:f₁''[x::=f₂''] ⇓ t₂
    using app by auto
  then have Lam [x]. f₁'' = Lam [x]. e₁' using ih1 by simp
  then have f₁'' = e₁' by (auto simp add: trm.inject alpha)
  moreover have f₂''=e₂' using x2 ih2 by simp
```

ultimately have $e_1{}'[x::=e_2{}'] \Downarrow t_2$ **using** *x3* **by** *simp*
thus *?case* **using** *ih3* **by** *simp*
next
case (*b-CaseL* x_1 e e_2 e'' x_2 e_1 e' t_2)
have *ih1*:$\bigwedge t.\ e \Downarrow t \Longrightarrow InL\ e' = t$ **by** *fact*
have *ih2*:$\bigwedge t.\ e_1[x_1::=e'] \Downarrow t \Longrightarrow e'' = t$ **by** *fact*
have *ha*:$\bigwedge t.\ (e \Downarrow InR\ t) \Longrightarrow False$ **using** *ih1* **by** *force*
have $Case\ e\ of\ inl\ x_1 \rightarrow e_1 \mid inr\ x_2 \rightarrow e_2 \Downarrow t_2$ **by** *fact*
then obtain xe' **where** $e \Downarrow InL\ xe'$ **and** h:$e_1[x_1::=xe'] \Downarrow t_2$ **using** *ha* **by** *auto*
then have $InL\ xe' = InL\ e'$ **using** *ih1* **by** *simp*
then have $xe' = e'$ **by** (*simp add*: *trm.inject*)
then have $e_1[x_1::=e'] \Downarrow t_2$ **using** *h* **by** *simp*
then show $e'' = t_2$ **using** *ih2* **by** *simp*
next
case (*b-CaseR* x_1 e e_2 e'' x_2 e_1 e' t_2)
have *ih1*:$\bigwedge t.\ e \Downarrow t \Longrightarrow InR\ e' = t$ **by** *fact*
have *ih2*:$\bigwedge t.\ e_2[x_2::=e'] \Downarrow t \Longrightarrow e'' = t$ **by** *fact*
have *a*:$\bigwedge t.\ (e \Downarrow InL\ t \Longrightarrow False)$ **using** *ih1* **by** *force*
have $Case\ e\ of\ inl\ x_1 \rightarrow e_1 \mid inr\ x_2 \rightarrow e_2 \Downarrow t_2$ **by** *fact*
then obtain xe' **where** $e \Downarrow InR\ xe'$ **and** h:$e_2[x_2::=xe'] \Downarrow t_2$ **using** *a* **by** *auto*
then have $InR\ xe' = InR\ e'$ **using** *ih1* **by** *simp*
then have $e_2[x_2::=e'] \Downarrow t_2$ **using** *h* **by** (*simp add*: *trm.inject*)
thus $e'' = t_2$ **using** *ih2* **by** *simp*
qed (*fast*)+

lemma *not-val-App*[*simp*]:
shows
$\neg$ *val* ($App\ e_1\ e_2$)
$\neg$ *val* ($Fst\ e$)
$\neg$ *val* ($Snd\ e$)
$\neg$ *val* ($Var\ x$)
$\neg$ *val* ($Case\ e\ of\ inl\ x_1 \rightarrow e_1 \mid inr\ x_2 \rightarrow e_2$)
by *auto*

lemma *reduces-to-value*:
assumes h:$t \Downarrow t'$
shows *val* t'
using *h* **by** (*induct*, *auto*)

lemma *type-prod-down-pair*:
assumes $\Gamma \vdash t : Data\ (DProd\ S_1\ S_2)$ **and** $t \Downarrow t'$
obtains $t_1\ t_2$ **where** $t' = Pr\ t_1\ t_2$
proof –
have $\Gamma \vdash t' : Data\ (DProd\ S_1\ S_2)$ **using** *assms subject-reduction* **by** *simp*
moreover
have *val* t' **using** *reduces-to-value assms* **by** *simp*
ultimately obtain $t_1\ t_2$ **where** $t' = Pr\ t_1\ t_2$ **by** (*cases*, *auto simp add*:*ty.inject data.inject*)
thus *?thesis* **using** *prems* **by** *auto*
qed

lemma *type-sum-down-or*:
assumes $\Gamma \vdash t : Data\ (DSum\ \sigma_1\ \sigma_2)$ **and** $t \Downarrow t'$
shows $(\exists\ t''.\ t' = InL\ t'') \vee (\exists\ t''.\ t' = InR\ t'')$

proof –
  have $\Gamma \vdash t' : Data\ (DSum\ \sigma_1\ \sigma_2)$ **using** *assms subject-reduction* **by** *simp*
  moreover
  have *val* t' **using** *reduces-to-value assms* **by** *simp*
  ultimately obtain t'' **where** $t' = InL\ t'' \vee\ t' = InR\ t''$
    by (*cases*, *auto simp add*:*ty.inject data.inject*)
  thus *?thesis* **by** *auto*
qed

lemma *type-arrow-down-lam*:
  assumes $\Gamma \vdash t : \sigma \rightarrow \tau$ **and** $t \Downarrow t'$
  obtains $x\ t''$ **where** $t' = Lam\ [x].\ t''$
proof –
  have $\Gamma \vdash t' : \sigma \rightarrow \tau$ **using** *assms subject-reduction* **by** *simp*
  moreover
  have *val* t' **using** *reduces-to-value assms* **by** *simp*
  ultimately obtain $x\ t''$ **where** $t' = Lam\ [x].\ t''$ **by** (*cases*, *auto simp add*:*ty.inject data.inject*)
  thus *?thesis* **using** *prems* **by** *auto*
qed

lemma *type-nat-down-const*:
  assumes $\Gamma \vdash t : Data\ DNat$ **and** $t \Downarrow t'$
  obtains n **where** $t' = Const\ n$
proof –
  have $\Gamma \vdash t' : Data\ DNat$ **using** *assms subject-reduction* **by** *simp*
  moreover have *val* t' **using** *reduces-to-value assms* **by** *simp*
  ultimately obtain n **where** $t' = Const\ n$ **by** (*cases*, *auto simp add*:*ty.inject data.inject*)
  thus *?thesis* **using** *prems* **by** *auto*
qed

11 Termination of evaluation

11.1 Definition of the logical relations

function
  $V' :: data \Rightarrow trm\ set$
where
  $V'\ (DNat) = \{Const\ n \mid n.\ n \in (UNIV::nat\ set)\}$
| $V'\ (DProd\ S_1\ S_2) = \{Pr\ x\ y \mid x\ y.\ x \in V'\ S_1 \wedge y \in V'\ S_2\}$
| $V'\ (DSum\ S_1\ S_2) = \{InL\ x \mid x.\ x \in V'\ S_1\} \cup \{InR\ y \mid y.\ y \in V'\ S_2\}$
apply (*auto simp add*: *data.inject ty.inject*)
apply (*subgoal-tac* $x{=}DNat \vee (\exists S_1\ S_2.\ x{=}DProd\ S_1\ S_2) \vee (\exists\ S_1\ S_2.\ x{=}DSum\ S_1\ S_2)$)
apply (*force*)
apply (*rule data-cases*)
done

termination
apply(*relation measure size*)
apply(*auto*)
done

lemma *Vprimes-are-values* :

fixes *S*::*data*
assumes *h*: $e \in V'\ S$
shows *val e*
using *h*
by (*nominal-induct S arbitrary*: *e rule*:*data.induct*)
(*auto*)

function
$V :: ty \Rightarrow trm\ set$
where
$V\ (Data\ S) = V'\ S$
$|\ V\ (T_1 \rightarrow T_2) = \{Lam\ [x].e\ |\ x\ e.\ \forall\ v \in (V\ T_1).\ \exists\ v'.\ e[x{::=}v] \Downarrow v' \wedge v' \in V\ T_2\}$
apply (*auto simp add*: *data.inject ty.inject*)
apply (*subgoal-tac* $(\exists\ T_1\ T_2.\ x{=}T_1 \rightarrow T_2) \vee (\exists\ T.\ x{=}Data\ T)$)
apply (*force*)
apply (*rule ty-cases*)
done

termination
apply(*relation measure size*)
apply *auto*
done

11.2 Inversion lemmas for logical relations

lemma *V-arrow-elim-weak*[*elim*] :
assumes $h{:}u \in (V\ (T_1 \rightarrow T_2))$
obtains *a t* **where** $u = Lam[a].t$ **and** $\forall\ v \in (V\ T_1).\ \exists\ v'.\ t[a{::=}v] \Downarrow v' \wedge v' \in V\ T_2$
using *h* **by** (*auto simp add*: *V.cases*)

lemma *V-arrow-elim-strong*[*elim*]:
fixes $c{::}'a{::}fs\text{-}name$
assumes *h*: $u \in (V\ (T_1 \rightarrow T_2))$
obtains *a t* **where** $a\#c\ u = Lam[a].t\ \forall\ v \in (V\ T_1).\ \exists\ v'.\ t[a{::=}v] \Downarrow v' \wedge v' \in V\ T_2$
using *h*
apply −
apply(*erule V-arrow-elim-weak*)
apply(*subgoal-tac* $\exists\, a'{::}name.\ a'\#(a,t,c)$)
apply(*erule exE*)
apply(*drule-tac* $x{=}a'$ **in** *meta-spec*)
apply(*simp*)
apply(*drule-tac* $x{=}[(a,a')]{\bullet}t$ **in** *meta-spec*)
apply(*simp add*: *trm.inject alpha fresh-prod fresh-atm*)
apply(*perm-simp*)
apply(*simp add*: *fresh-left calc-atm*)
apply(*auto*)
apply(*simp add*: *subst-rename*)
apply(*subgoal-tac* $[(a',a)]{\bullet}t = [(a,a')]{\bullet}t$)
apply(*simp*)
apply(*rule pt-name3*)
apply(*rule at-ds5*[*OF at-name-inst*])
apply(*rule exists-fresh'*)
apply(*simp add*: *fs-name1*)

done

lemma *V-are-values* :
  fixes T::ty
  assumes h:$e \in V\ T$
  shows $val\ e$
using h **by** (*nominal-induct T arbitrary*: *e rule*:*ty.induct*, *auto simp add*: *Vprimes-are-values*)

lemma *values-reduce-to-themselves*:
  assumes h:$val\ v$
  shows $v \Downarrow v$
using h **by** (*induct*,*auto*)

lemma *Vs-reduce-to-themselves*[*simp*]:
  assumes h:$v \in V\ T$
  shows $v \Downarrow v$
using h **by** (*simp add*: *values-reduce-to-themselves V-are-values*)

lemma *V-sum*:
  assumes h:$x \in V\ (Data\ (DSum\ S_1\ S_2))$
  shows $(\exists\ y.\ x = InL\ y \wedge\ y \in V'\ S_1) \vee (\exists\ y.\ x = InR\ y \wedge y \in V'\ S_2)$
using h **by** *simp*

11.3 Monotonicity

abbreviation
 mapsto :: (*name*×*trm*) *list* ⇒ *name* ⇒ *trm* ⇒ *bool* (- *maps* - *to* - [*55*,*55*,*55*] *55*)
where
 θ *maps* x *to* $e \equiv (lookup\ \theta\ x) = e$

abbreviation
  v-closes :: (*name*×*trm*) *list* ⇒ (*name*×*ty*) *list* ⇒ *bool* (- *Vcloses* - [*55*,*55*] *55*)
where
  $\theta\ Vcloses\ \Gamma \equiv \forall x\ T.\ ((x,T) \in set\ \Gamma \longrightarrow (\exists e.\ \theta\ maps\ x\ to\ e \wedge e \in (V\ T)))$

lemma *monotonicity*:
  fixes m::$name$
  fixes θ::($name \times trm$) $list$
  assumes $h1$: $\theta\ Vcloses\ \Gamma$
  and $h2$: $e \in V\ T$
  and $h3$: $valid\ ((x,T)\#\Gamma)$
  shows $(x,e)\#\theta\ Vcloses\ (x,T)\#\Gamma$
proof(*intro strip*)
  fix $x'\ T'$
  assume $(x',T') \in set\ ((x,T)\#\Gamma)$
  then have $((x',T')=(x,T)) \vee ((x',T')\in set\ \Gamma \wedge x' \neq x)$ **using** $h3$
    by (*rule-tac case-distinction-on-context*)
  moreover
  {
    assume $(x',T') = (x,T)$
    then have $\exists e'.\ ((x,e)\#\theta)\ maps\ x\ to\ e' \wedge e' \in V\ T'$ **using** $h2$ **by** *auto*

```
  }
  moreover
  {
    assume (x',T') ∈ set Γ and neq:x' ≠ x
      then have ∃ e'. θ maps x' to e' ∧ e' ∈ V T' using h1 by auto
      then have ∃ e'. ((x,e)#θ) maps x' to e' ∧ e' ∈ V T' using neq by auto
  }
  ultimately show ∃ e'. ((x,e)#θ) maps x' to e' ∧ e' ∈ V T' by blast
qed
```

11.4 The termination proof

```
lemma termination-aux:
  fixes T :: ty
  fixes Γ :: (name × ty) list
  fixes θ :: (name × trm) list
  fixes e :: trm
  assumes h1: Γ ⊢ e : T
  and     h2: θ Vcloses Γ
  shows ∃ v. θ<e> ⇓ v ∧ v ∈ V T
using h2 h1
proof(nominal-induct e avoiding: Γ θ arbitrary: T rule: trm.induct)
  case (App e1 e2 Γ θ T)
  have ih1:⋀θ Γ T. ⟦θ Vcloses Γ; Γ ⊢ e1 : T⟧ ⟹ ∃ v. θ<e1> ⇓ v ∧ v ∈ V T by fact
  have ih2:⋀θ Γ T. ⟦θ Vcloses Γ; Γ ⊢ e2 : T⟧ ⟹ ∃ v. θ<e2> ⇓ v ∧ v ∈ V T by fact
  have as1:θ Vcloses Γ by fact
  have as2: Γ ⊢ App e1 e2 : T by fact
  from as2 obtain T' where Γ ⊢ e1 : T' → T and Γ ⊢ e2 : T' by auto
  then obtain v1 v2 where (i): θ<e1> ⇓ v1 v1 ∈ V (T' → T)
                  and (ii):θ<e2> ⇓ v2 v2 ∈ V T' using ih1 ih2 as1 by blast
  from (i) obtain x e'
          where v1 = Lam[x].e'
          and (iii): (∀ v ∈ (V T').∃ v'. e'[x::=v] ⇓ v' ∧ v' ∈ V T)
          and (iv):  θ<e1> ⇓ Lam [x].e'
          and fr: x#(θ,e1,e2) by blast
  from fr have fr1: x#θ<e1> and fr2: x#θ<e2> by (simp-all add: fresh-psubst)
  from (ii) (iii) obtain v3 where (v): e'[x::=v2] ⇓ v3 v3 ∈ V T by auto
  from fr2 (ii) have x#v2 by (simp add: fresh-preserved)
  then have x#e'[x::=v2] by (simp add: fresh-subst-fresh)
  then have fr3: x#v3 using (v) by (simp add: fresh-preserved)
  from fr1 fr2 fr3 have x#(θ<e1>,θ<e2>,v3) by simp
  with (iv) (ii) (v) have App (θ<e1>) (θ<e2>) ⇓ v3 by auto
  then show ∃ v. θ<App e1 e2> ⇓ v ∧ v ∈ V T using (v) by auto
next
  case (Pr t1 t2 Γ θ T)
  have Γ ⊢ Pr t1 t2 : T by fact
  then obtain Ta Tb where ta:Γ ⊢ t1 : Data Ta and Γ ⊢ t2 : Data Tb
                  and eq:T=Data (DProd Ta Tb) by auto
  have h:θ Vcloses Γ by fact
  then obtain v1 v2 where θ<t1> ⇓ v1 ∧ v1 ∈ V (Data Ta) θ<t2> ⇓ v2 ∧ v2 ∈ V (Data Tb)
    using prems by blast
  thus ∃ v. θ<Pr t1 t2> ⇓ v ∧ v ∈ V T using eq by auto
next
```

case $(Lam\ x\ e\ \Gamma\ \theta\ T)$
have $ih{:}\bigwedge\theta\ \Gamma\ T.\ [\![\theta\ Vcloses\ \Gamma;\ \Gamma \vdash e : T]\!] \Longrightarrow \exists v.\ \theta{<}e{>} \Downarrow v \wedge v \in V\ T$ **by** *fact*
have as_1: $\theta\ Vcloses\ \Gamma$ **by** *fact*
have as_2: $\Gamma \vdash Lam\ [x].e : T$ **by** *fact*
have *fs*: $x\#\Gamma\ x\#\theta$ **by** *fact*
from as_2 *fs* **obtain** $T_1\ T_2$
where (i): $(x,T_1)\#\Gamma \vdash e{:}T_2$ **and** (ii): $T = T_1 \to T_2$ **by** *auto*
from (i) **have** (iii): $valid\ ((x,T_1)\#\Gamma)$ **by** $(simp\ add{:}\ typing\text{-}valid)$
have $\forall v \in (V\ T_1).\ \exists v'.\ (\theta{<}e{>})[x{::=}v] \Downarrow v' \wedge v' \in V\ T_2$
proof
fix v
assume $v \in (V\ T_1)$
with (iii) as_1 **have** $(x,v)\#\theta\ Vcloses\ (x,T_1)\#\Gamma$ **using** *monotonicity* **by** *auto*
with *ih* (i) **obtain** v' **where** $((x,v)\#\theta){<}e{>} \Downarrow v' \wedge v' \in V\ T_2$ **by** *blast*
then have $\theta{<}e{>}[x{::=}v] \Downarrow v' \wedge v' \in V\ T_2$ **using** *fs*
by $(simp\ add{:}\ psubst\text{-}subst\text{-}psubst)$
then show $\exists v'.\ \theta{<}e{>}[x{::=}v] \Downarrow v' \wedge v' \in V\ T_2$ **by** *auto*
qed
then have $Lam[x].\theta{<}e{>} \in V\ (T_1 \to T_2)$ **by** *auto*
then have $\theta{<}Lam\ [x].e{>} \Downarrow Lam[x].\theta{<}e{>} \wedge Lam[x].\theta{<}e{>} \in V\ (T_1{\to}T_2)$ **using** *fs* **by** *auto*
thus $\exists v.\ \theta{<}Lam\ [x].e{>} \Downarrow v \wedge v \in V\ T$ **using** (ii) **by** *auto*
next
case $(Case\ t'\ n_1\ t_1\ n_2\ t_2\ \Gamma\ \theta\ T)$
have f: $n_1\#\Gamma\ n_1\#\theta\ n_2\#\Gamma\ n_2\#\theta\ n_2{\neq}n_1\ n_1\#t'$
$n_1\#t_2\ n_2\#t'\ n_2\#t_1$ **by** *fact*
have $h{:}\theta\ Vcloses\ \Gamma$ **by** *fact*
have $th{:}\Gamma \vdash Case\ t'\ of\ inl\ n_1 \to t_1 \mid inr\ n_2 \to t_2 : T$ **by** *fact*
then obtain $S_1\ S_2$ **where**
$hm{:}\Gamma \vdash t' : Data\ (DSum\ S_1\ S_2)$ **and**
$hl{:}(n_1,Data\ S_1)\#\ \Gamma \vdash t_1 : T$ **and**
$hr{:}(n_2,Data\ S_2)\#\ \Gamma \vdash t_2 : T$ **using** f **by** *auto*
then obtain v_0 **where** $ht'{:}\theta{<}t'{>} \Downarrow v_0$ **and** $hS{:}v_0 \in V\ (Data\ (DSum\ S_1\ S_2))$ **using** *prems h* **by** *blast*

{
fix ${v_0}'$
assume $eqc{:}v_0 = InL\ {v_0}'$ **and** ${v_0}' \in V'\ S_1$
then have *inc*: ${v_0}' \in V\ (Data\ S_1)$ **by** *auto*
have $valid\ \Gamma$ **using** *th typing-valid* **by** *auto*
then moreover have $valid\ ((n_1,Data\ S_1)\#\Gamma)$ **using** f **by** *auto*
then moreover have $(n_1,{v_0}')\#\theta\ Vcloses\ (n_1,Data\ S_1)\#\Gamma$
using *inc h monotonicity* **by** *blast*
moreover have $ih{:}\bigwedge\Gamma\ \theta\ T.\ [\![\theta\ Vcloses\ \Gamma;\ \Gamma \vdash t_1 : T]\!] \Longrightarrow$
$\exists v.\ \theta{<}t_1{>} \Downarrow v \wedge v \in V\ T$ **by** *fact*
ultimately obtain v_1 **where** $ho{:}((n_1,{v_0}')\#\theta){<}t_1{>} \Downarrow v_1 \wedge v_1 \in V\ T$ **using** *hl* **by** *blast*
then have $r{:}\theta{<}t_1{>}[n_1{::=}{v_0}'] \Downarrow v_1 \wedge v_1 \in V\ T$ **using** *psubst-subst-psubst f* **by** *simp*
then moreover have $n_1\#(\theta{<}t'{>},\theta{<}t_2{>},v_1,n_2)$
proof −
have $n_1\#v_0$ **using** *ht′ fresh-preserved fresh-psubst f* **by** *auto*
then have $n_1\#{v_0}'$ **using** *eqc* **by** *auto*
then have $n_1\#v_1$ **using** *f r fresh-preserved fresh-subst-fresh* **by** *blast*
thus $n_1\#(\theta{<}t'{>},\theta{<}t_2{>},v_1,n_2)$ **using** f **by** $(simp\ add{:}\ fresh\text{-}atm\ fresh\text{-}psubst)$
qed

moreover have $n_2\#(\theta<t'>,\theta<t_1>,v_1,n_1)$
proof −
have $n_2\#v_0$ **using** *ht′ fresh-preserved fresh-psubst f* **by** *auto*
then have $n_2\#v_0{}'$ **using** *eqc* **by** *auto*
then have $n_2\#((n_1,v_0{}')\#\theta)$ **using** *f fresh-list-cons fresh-atm* **by** *force*
then have $n_2\#((n_1,v_0{}')\#\theta)<t_1>$ **using** *f fresh-psubst* **by** *auto*
moreover then have $n_2 \# v_1$ **using** *fresh-preserved ho* **by** *auto*
ultimately show $n_2\#(\theta<t'>,\theta<t_1>,v_1,n_1)$ **using** *f* **by** (*simp add*: *fresh-psubst fresh-atm*)
qed
ultimately have $\mathit{Case}\ \theta<t'>\ \mathit{of}\ \mathit{inl}\ n_1 \rightarrow \theta<t_1>\ |\ \mathit{inr}\ n_2 \rightarrow \theta<t_2> \Downarrow v_1 \wedge v_1 \in V\ T$ **using** *ht′*
eqc **by** *auto*
moreover
have $\mathit{Case}\ \theta<t'>\ \mathit{of}\ \mathit{inl}\ n_1 \rightarrow \theta<t_1>\ |\ \mathit{inr}\ n_2 \rightarrow \theta<t_2> = \theta<\mathit{Case}\ t'\ \mathit{of}\ \mathit{inl}\ n_1 \rightarrow t_1\ |\ \mathit{inr}\ n_2$
$\rightarrow t_2>$
using *f* **by** *auto*
ultimately have $\exists v.\ \theta<\mathit{Case}\ t'\ \mathit{of}\ \mathit{inl}\ n_1 \rightarrow t_1\ |\ \mathit{inr}\ n_2 \rightarrow t_2> \Downarrow v \wedge v \in V\ T$ **by** *auto*
}
moreover
{
fix $v_0{}'$
assume *eqc*:$v_0 = \mathit{InR}\ v_0{}'$ **and** $v_0{}' \in V'\ S_2$
then have *inc*:$v_0{}' \in V\ (\mathit{Data}\ S_2)$ **by** *auto*
have *valid* Γ **using** *th typing-valid* **by** *auto*
then moreover have *valid* $((n_2,\mathit{Data}\ S_2)\#\Gamma)$ **using** *f* **by** *auto*
then moreover have $(n_2,v_0{}')\#\theta$ *Vcloses* $(n_2,\mathit{Data}\ S_2)\#\Gamma$
using *inc h monotonicity* **by** *blast*
moreover have *ih*:$\bigwedge\Gamma\ \theta\ T.\ [\![\theta\ \mathit{Vcloses}\ \Gamma;\ \Gamma \vdash t_2 : T]\!] \Longrightarrow \exists v.\ \theta<t_2> \Downarrow v \wedge v \in V\ T$ **by** *fact*
ultimately obtain v_2 **where** *ho*:$((n_2,v_0{}')\#\theta)<t_2> \Downarrow v_2 \wedge v_2 \in V\ T$ **using** *hr* **by** *blast*
then have *r*:$\theta<t_2>[n_2{::=}v_0{}'] \Downarrow v_2 \wedge v_2 \in V\ T$ **using** *psubst-subst-psubst f* **by** *simp*
moreover have $n_1\#(\theta<t'>,\theta<t_2>,v_2,n_2)$
proof −
have $n_1\#\theta<t'>$ **using** *fresh-psubst f* **by** *simp*
then have $n_1\#v_0$ **using** *ht′ fresh-preserved* **by** *auto*
then have $n_1\#v_0{}'$ **using** *eqc* **by** *auto*
then have $n_1\#((n_2,v_0{}')\#\theta)$ **using** *f fresh-list-cons fresh-atm* **by** *force*
then have $n_1\#((n_2,v_0{}')\#\theta)<t_2>$ **using** *f fresh-psubst* **by** *auto*
moreover then have $n_1\#v_2$ **using** *fresh-preserved ho* **by** *auto*
ultimately show $n_1 \# (\theta<t'>,\theta<t_2>,v_2,n_2)$ **using** *f* **by** (*simp add*: *fresh-psubst fresh-atm*)
qed
moreover have $n_2 \# (\theta<t'>,\theta<t_1>,v_2,n_1)$
proof −
have $n_2\#\theta<t'>$ **using** *fresh-psubst f* **by** *simp*
then have $n_2\#v_0$ **using** *ht′ fresh-preserved* **by** *auto*
then have $n_2\#v_0{}'$ **using** *eqc* **by** *auto*
then have $n_2\#\theta<t_2>[n_2{::=}v_0{}']$ **using** *f fresh-subst-fresh* **by** *auto*
then have $n_2\#v_2$ **using** *f fresh-preserved r* **by** *blast*
then show $n_2\#(\theta<t'>,\theta<t_1>,v_2,n_1)$ **using** *f* **by** (*simp add*: *fresh-atm fresh-psubst*)
qed
ultimately have $\mathit{Case}\ \theta<t'>\ \mathit{of}\ \mathit{inl}\ n_1 \rightarrow \theta<t_1>\ |\ \mathit{inr}\ n_2 \rightarrow \theta<t_2> \Downarrow v_2 \wedge v_2 \in V\ T$ **using** *ht′*
eqc **by** *auto*
then have $\exists v.\ \theta<\mathit{Case}\ t'\ \mathit{of}\ \mathit{inl}\ n_1 \rightarrow t_1\ |\ \mathit{inr}\ n_2 \rightarrow t_2> \Downarrow v \wedge v \in V\ T$ **using** *f* **by** *auto*
}
ultimately show $\exists v.\ \theta<\mathit{Case}\ t'\ \mathit{of}\ \mathit{inl}\ n_1 \rightarrow t_1\ |\ \mathit{inr}\ n_2 \rightarrow t_2> \Downarrow v \wedge v \in V\ T$ **using** *hS V-sum*

by *blast*
qed (*force*)+

Well type closed terms reduce to a value

theorem *termination-of-evaluation*:
  assumes *a*: $[] \vdash e : T$
  shows $\exists v.\ e \Downarrow v \wedge val\ v$
proof −
  from *a* **have** $\exists v.\ (([]::(name \times trm)\ list)<e>) \Downarrow v \wedge v \in V\ T$
    by (*rule termination-aux*) (*auto*)
  thus $\exists v.\ e \Downarrow v \wedge val\ v$ **using** *V-are-values* **by** *auto*
qed

end