

# Formalization of Generic Authenticated Data Structures

Matthias Brun      Dmitriy Traytel

March 8, 2026

## Abstract

Authenticated data structures are a technique for outsourcing data storage and maintenance to an untrusted server. The server is required to produce an efficiently checkable and cryptographically secure proof that it carried out precisely the requested computation. Miller et al. [2] introduced  $\lambda\bullet$  (pronounced *lambda auth*)—a functional programming language with a built-in primitive authentication construct, which supports a wide range of user-specified authenticated data structures while guaranteeing certain correctness and security properties for all well-typed programs. We formalize  $\lambda\bullet$  and prove its correctness and security properties. With Isabelle’s help, we uncover and repair several mistakes in the informal proofs and lemma statements. Our findings are summarized in an ITP’19 paper [1].

## Contents

<b>1</b>	<b>Preliminaries</b>	<b>2</b>
<b>2</b>	<b>Syntax of <math>\lambda\bullet</math></b>	<b>4</b>
<b>3</b>	<b>Semantics of <math>\lambda\bullet</math></b>	<b>6</b>
3.1	Equivariant Hash Function	6
3.2	Substitution	7
3.3	Weak Typing Judgement	9
3.4	Erasure of Authenticated Types	11
3.5	Strong Typing Judgement	11
3.6	Shallow Projection	12
3.7	Small-step Semantics	13
3.8	Type Progress	17
3.9	Weak Type Preservation	17
3.10	Corrected Lemma 1 from Miller et al. [2]: Weak Type Soundness	18
<b>4</b>	<b>Agreement Relation</b>	<b>18</b>
<b>5</b>	<b>Formalization of Miller et al.’s [2] Main Results</b>	<b>21</b>
5.1	Lemma 2.1	21
5.2	Counterexample to Lemma 2.2	21
5.3	Lemma 2.3	21
5.4	Lemma 2.4	22
5.5	Lemma 3	22
5.6	Lemma 4	22
5.7	Lemma 5: Single-Step Correctness	22
5.8	Lemma 6: Single-Step Security	22
5.9	Theorem 1: Correctness	23
5.10	Counterexamples to Theorem 1: Security	23
5.11	Corrected Theorem 1: Security	23
5.12	Remark 1	23

# 1 Preliminaries

Auxiliary freshness lemmas and simplifier setup.

**declare**

*fresh\_star\_Pair*[simp] *fresh\_star\_insert*[simp] *fresh\_Nil*[simp]  
*pure\_supp*[simp] *pure\_fresh*[simp]

**lemma** *fresh\_star\_Nil*[simp]: {} #\* t  
⟨proof⟩

**lemma** *supp\_flip*[simp]:  
**fixes** a b :: \_ :: at  
**shows** *supp* (a ↔ b) = (if a = b then {} else {atom a, atom b})  
⟨proof⟩

**lemma** *Abs\_lst\_eq\_flipI*:  
**fixes** a b :: \_ :: at **and** t :: \_ :: fs  
**assumes** atom b # t  
**shows** [[atom a]]lst. t = [[atom b]]lst. (a ↔ b) · t  
⟨proof⟩

**lemma** *atom\_not\_fresh\_eq*:  
**assumes** ¬ atom a # x  
**shows** a = x  
⟨proof⟩

**lemma** *fresh\_set\_fresh\_forall*:  
**shows** atom y # xs = (∀ x ∈ set xs. atom y # x)  
⟨proof⟩

**lemma** *finite\_fresh\_set\_fresh\_all*[simp]:  
**fixes** S :: (\_ :: fs) set  
**shows** finite S ⇒ atom a # S ↔ (∀ x ∈ S. atom a # x)  
⟨proof⟩

**lemma** *case\_option\_eqvt*[eqvt]:  
p · case\_option a b opt = case\_option (p · a) (p · b) (p · opt)  
⟨proof⟩

Nominal setup for finite maps.

**abbreviation** *fmap\_update* (⟨\_ '(\_ \$\$:= \_)⟩ [1000,0,0] 1000) **where** *fmap\_update* Γ x τ ≡ *fmupd* x τ Γ

**notation** *fmlookup* (infixl <\$\$> 999)

**notation** *fmempty* (⟨{\$\$}>)

**instantiation** *fmap* :: (pt, pt) pt  
**begin**

**unbundle** *fmap.lifting*

**lift\_definition**  
*permute\_fmap* :: perm ⇒ ('a, 'b) fmap ⇒ ('a, 'b) fmap  
**is**  
*permute* :: perm ⇒ ('a ↦ 'b) ⇒ ('a ↦ 'b)  
⟨proof⟩

**instance**  
*<proof>*

**end**

**lemma** *fmempty\_eqvt*[*eqvt*]:  
**shows**  $(p \cdot \{\$\$ \}) = \{\$\$ \}$   
*<proof>*

**lemma** *fmap\_update\_eqvt*[*eqvt*]:  
**shows**  $(p \cdot f(a \ \$\$ = b)) = (p \cdot f)((p \cdot a) \ \$\$ = (p \cdot b))$   
*<proof>*

**lemma** *fmap\_apply\_eqvt*[*eqvt*]:  
**shows**  $(p \cdot (f \ \$\$ b)) = (p \cdot f) \ \$\$ (p \cdot b)$   
*<proof>*

**lemma** *fresh\_fmempty*[*simp*]:  
**shows**  $a \ \#\ \{\$\$ \}$   
*<proof>*

**lemma** *fresh\_fmap\_update*:  
**shows**  $\llbracket a \ \#\ f; a \ \#\ x; a \ \#\ y \rrbracket \implies a \ \#\ f(x \ \$\$ = y)$   
*<proof>*

**lemma** *supp\_fmempty*[*simp*]:  
**shows**  $\text{supp } \{\$\$ \} = \{\}$   
*<proof>*

**lemma** *supp\_fmap\_update*:  
**shows**  $\text{supp } (f(x \ \$\$ = y)) \subseteq \text{supp}(f, x, y)$   
*<proof>*

**instance** *fmap* :: (*fs*, *fs*) *fs*  
*<proof>*

**lemma** *fresh\_transfer*[*transfer\_rule*]:  
 $((=) \implies \text{pcr\_fmap } (=) \implies (=) \implies \text{fresh fresh})$   
*<proof>*

**lemma** *fmmap\_eqvt*[*eqvt*]:  $p \cdot (\text{fmmap } f F) = \text{fmmap } (p \cdot f) (p \cdot F)$   
*<proof>*

**lemma** *fmap\_freshness\_lemma*:  
**fixes**  $h :: ('a::\text{at}, 'b::\text{pt}) \text{ fmap}$   
**assumes**  $a: \exists a. \text{atom } a \ \#\ (h, h \ \$\$ a)$   
**shows**  $\exists x. \forall a. \text{atom } a \ \#\ h \longrightarrow h \ \$\$ a = x$   
*<proof>*

**lemma** *fmap\_freshness\_lemma\_unique*:  
**fixes**  $h :: ('a::\text{at}, 'b::\text{pt}) \text{ fmap}$   
**assumes**  $\exists a. \text{atom } a \ \#\ (h, h \ \$\$ a)$   
**shows**  $\exists! x. \forall a. \text{atom } a \ \#\ h \longrightarrow h \ \$\$ a = x$   
*<proof>*

**lemma** *fmdrop\_fset\_fmupd*[*simp*]:  
 $(\text{fmdrop\_fset } A f)(x \ \$\$ = y) = \text{fmdrop\_fset } (A \ |-\ | \{|x|\}) f(x \ \$\$ = y)$

**including** *fmap.lifting* **and** *fset.lifting*  
*<proof>*

**lemma** *fresh\_fset\_fminus*:  
**assumes** *atom x # A*  
**shows**  $A \setminus \{|x|\} = A$   
*<proof>*

**lemma** *fresh\_fun\_app*:  
**shows**  $\text{atom } x \# F \implies x \neq y \implies F y = \text{Some } a \implies \text{atom } x \# a$   
*<proof>*

**lemma** *fresh\_fmap\_fresh\_Some*:  
 $\text{atom } x \# F \implies x \neq y \implies F \$\$ y = \text{Some } a \implies \text{atom } x \# a$   
**including** *fmap.lifting*  
*<proof>*

**lemma** *fmdrop\_eqvt*:  $p \cdot \text{fmdrop } x F = \text{fmdrop } (p \cdot x) (p \cdot F)$   
*<proof>*

**lemma** *fmfilter\_eqvt*:  $p \cdot \text{fmfilter } Q F = \text{fmfilter } (p \cdot Q) (p \cdot F)$   
*<proof>*

**lemma** *fmdrop\_eq\_iff*:  
 $\text{fmdrop } x B = \text{fmdrop } y B \iff x = y \vee (x \notin \text{fmdom}' B \wedge y \notin \text{fmdom}' B)$   
*<proof>*

**lemma** *fresh\_fun\_upd*:  
**shows**  $\llbracket a \# f; a \# x; a \# y \rrbracket \implies a \# f(x := y)$   
*<proof>*

**lemma** *supp\_fun\_upd*:  
**shows**  $\text{supp } (f(x := y)) \subseteq \text{supp}(f, x, y)$   
*<proof>*

**lemma** *map\_drop\_fun\_upd*:  $\text{map\_drop } x F = F(x := \text{None})$   
*<proof>*

**lemma** *fresh\_fmdrop\_in\_fndom*:  $\llbracket x \in \text{fmdom}' B; y \# B; y \# x \rrbracket \implies y \# \text{fmdrop } x B$   
*<proof>*

**lemma** *fresh\_fmdrop*:  
**assumes**  $x \# B \ x \# y$   
**shows**  $x \# \text{fmdrop } y B$   
*<proof>*

**lemma** *fresh\_fmdrop\_fset*:  
**fixes**  $x :: \text{atom}$  **and**  $A :: (\_ :: \text{at\_base}) \text{ fset}$   
**assumes**  $x \# A \ x \# B$   
**shows**  $x \# \text{fmdrop\_fset } A B$   
*<proof>*

## 2 Syntax of $\lambda\bullet$

**typeddecl** *hash*  
**instantiation** *hash* :: *pure*  
**begin**

**definition** *permute\_hash* :: *perm*  $\Rightarrow$  *hash*  $\Rightarrow$  *hash* **where**  
*permute\_hash*  $\pi$  *h* = *h*  
**instance** *<proof>*  
**end**

**atom\_decl** *var*

**nominal\_datatype** *term* =  
*Unit* |  
*Var* *var* |  
*Lam* *x::var t::term binds x in t* |  
*Rec* *x::var t::term binds x in t* |  
*Inj1* *term* |  
*Inj2* *term* |  
*Pair* *term term* |  
*Let* *term x::var t::term binds x in t* |  
*App* *term term* |  
*Case* *term term term* |  
*Prj1* *term* |  
*Prj2* *term* |  
*Roll* *term* |  
*Unroll* *term* |  
*Auth* *term* |  
*Unauth* *term* |  
*Hash* *hash* |  
*Hashed* *hash term*

**atom\_decl** *tvar*

**nominal\_datatype** *ty* =  
*One* |  
*Fun* *ty ty* |  
*Sum* *ty ty* |  
*Prod* *ty ty* |  
*Mu*  $\alpha::tvar \tau::ty$  **binds**  $\alpha$  **in**  $\tau$  |  
*Alpha* *tvar* |  
*AuthT* *ty*

**lemma** *no\_tvars\_in\_term*[*simp*]: *atom* (*x :: tvar*)  $\#$  (*t :: term*)  
*<proof>*

**lemma** *no\_vars\_in\_ty*[*simp*]: *atom* (*x :: var*)  $\#$  ( $\tau :: ty$ )  
*<proof>*

**inductive** *value* :: *term*  $\Rightarrow$  *bool* **where**  
*value* *Unit* |  
*value* (*Var*  $\_$ ) |  
*value* (*Lam*  $\_ \_$ ) |  
*value* (*Rec*  $\_ \_$ ) |  
*value* *v*  $\Rightarrow$  *value* (*Inj1* *v*) |  
*value* *v*  $\Rightarrow$  *value* (*Inj2* *v*) |  
 $\llbracket$  *value* *v*<sub>1</sub>; *value* *v*<sub>2</sub>  $\rrbracket$   $\Rightarrow$  *value* (*Pair* *v*<sub>1</sub> *v*<sub>2</sub>) |  
*value* *v*  $\Rightarrow$  *value* (*Roll* *v*) |  
*value* (*Hash*  $\_$ ) |  
*value* *v*  $\Rightarrow$  *value* (*Hashed*  $\_$  *v*)

**declare** *value.intros*[*simp*]  
**declare** *value.intros*[*intro*]

**equivariance** *value*

**lemma** *value\_inv[simp]*:

¬ *value* (*Let*  $e_1$   $x$   $e_2$ )  
¬ *value* (*App*  $v$   $v'$ )  
¬ *value* (*Case*  $v$   $v_1$   $v_2$ )  
¬ *value* (*Prj1*  $v$ )  
¬ *value* (*Prj2*  $v$ )  
¬ *value* (*Unroll*  $v$ )  
¬ *value* (*Auth*  $v$ )  
¬ *value* (*Unauth*  $v$ )  
{*proof*}

**inductive\_cases** *value\_Inj1\_inv[elim]*: *value* (*Inj1*  $e$ )

**inductive\_cases** *value\_Inj2\_inv[elim]*: *value* (*Inj2*  $e$ )

**inductive\_cases** *value\_Pair\_inv[elim]*: *value* (*Pair*  $e_1$   $e_2$ )

**inductive\_cases** *value\_Roll\_inv[elim]*: *value* (*Roll*  $e$ )

**inductive\_cases** *value\_Hashed\_inv[elim]*: *value* (*Hashed*  $h$   $e$ )

**abbreviation** *closed* :: *term*  $\Rightarrow$  *bool* **where**

*closed*  $t \equiv (\forall x::\text{var. atom } x \# t)$

### 3 Semantics of $\lambda\bullet$

Avoid clash with substitution notation.

**no\_notation** *inverse\_divide* (**infixl**  $\langle' / \rangle$  70)

Help automated provers with smallsteps.

**declare** *One\_nat\_def[simp del]*

#### 3.1 Equivariant Hash Function

**consts** *hash\_real* :: *term*  $\Rightarrow$  *hash*

**nominal\_function** *map\_fixed* :: *var*  $\Rightarrow$  *var list*  $\Rightarrow$  *term*  $\Rightarrow$  *term* **where**

*map\_fixed*  $fp$   $l$  *Unit* = *Unit* |  
*map\_fixed*  $fp$   $l$  (*Var*  $y$ ) = (if  $y \in \text{set } l$  then (*Var*  $y$ ) else (*Var*  $fp$ )) |  
*atom*  $y \# (fp, l) \Rightarrow \text{map\_fixed } fp$   $l$  (*Lam*  $y$   $t$ ) = (*Lam*  $y$  ((*map\_fixed*  $fp$  ( $y \# l$ )  $t$ ))) |  
*atom*  $y \# (fp, l) \Rightarrow \text{map\_fixed } fp$   $l$  (*Rec*  $y$   $t$ ) = (*Rec*  $y$  ((*map\_fixed*  $fp$  ( $y \# l$ )  $t$ ))) |  
*map\_fixed*  $fp$   $l$  (*Inj1*  $t$ ) = (*Inj1* ((*map\_fixed*  $fp$   $l$   $t$ ))) |  
*map\_fixed*  $fp$   $l$  (*Inj2*  $t$ ) = (*Inj2* ((*map\_fixed*  $fp$   $l$   $t$ ))) |  
*map\_fixed*  $fp$   $l$  (*Pair*  $t_1$   $t_2$ ) = (*Pair* ((*map\_fixed*  $fp$   $l$   $t_1$ )) ((*map\_fixed*  $fp$   $l$   $t_2$ ))) |  
*map\_fixed*  $fp$   $l$  (*Roll*  $t$ ) = (*Roll* ((*map\_fixed*  $fp$   $l$   $t$ ))) |  
*atom*  $y \# (fp, l) \Rightarrow \text{map\_fixed } fp$   $l$  (*Let*  $t_1$   $y$   $t_2$ ) = (*Let* ((*map\_fixed*  $fp$   $l$   $t_1$ ))  $y$  ((*map\_fixed*  $fp$  ( $y \# l$ )  $t_2$ ))) |  
*map\_fixed*  $fp$   $l$  (*App*  $t_1$   $t_2$ ) = (*App* ((*map\_fixed*  $fp$   $l$   $t_1$ )) ((*map\_fixed*  $fp$   $l$   $t_2$ ))) |  
*map\_fixed*  $fp$   $l$  (*Case*  $t_1$   $t_2$   $t_3$ ) = (*Case* ((*map\_fixed*  $fp$   $l$   $t_1$ )) ((*map\_fixed*  $fp$   $l$   $t_2$ )) ((*map\_fixed*  $fp$   $l$   $t_3$ ))) |  
*map\_fixed*  $fp$   $l$  (*Prj1*  $t$ ) = (*Prj1* ((*map\_fixed*  $fp$   $l$   $t$ ))) |  
*map\_fixed*  $fp$   $l$  (*Prj2*  $t$ ) = (*Prj2* ((*map\_fixed*  $fp$   $l$   $t$ ))) |  
*map\_fixed*  $fp$   $l$  (*Unroll*  $t$ ) = (*Unroll* ((*map\_fixed*  $fp$   $l$   $t$ ))) |  
*map\_fixed*  $fp$   $l$  (*Auth*  $t$ ) = (*Auth* ((*map\_fixed*  $fp$   $l$   $t$ ))) |  
*map\_fixed*  $fp$   $l$  (*Unauth*  $t$ ) = (*Unauth* ((*map\_fixed*  $fp$   $l$   $t$ ))) |  
*map\_fixed*  $fp$   $l$  (*Hash*  $h$ ) = (*Hash*  $h$ ) |  
*map\_fixed*  $fp$   $l$  (*Hashed*  $h$   $t$ ) = (*Hashed*  $h$  ((*map\_fixed*  $fp$   $l$   $t$ )))

$\langle \text{proof} \rangle$   
**nominal\_termination** (*eqvt*)  
 $\langle \text{proof} \rangle$

**definition** *hash where*  
 $\text{hash } t = \text{hash\_real } (\text{map\_fixed } \text{undefined } [] t)$

**lemma** *permute\_map\_list*:  $p \cdot l = \text{map } (\lambda x. p \cdot x) l$   
 $\langle \text{proof} \rangle$

**lemma** *map\_fixed\_eqvt*:  $p \cdot l = l \implies \text{map\_fixed } v l (p \cdot t) = \text{map\_fixed } v l t$   
 $\langle \text{proof} \rangle$

**lemma** *hash\_eqvt[eqvt]*:  $p \cdot \text{hash } t = \text{hash } (p \cdot t)$   
 $\langle \text{proof} \rangle$

**lemma** *map\_fixed\_idle*:  $\{x. \neg \text{atom } x \# t\} \subseteq \text{set } l \implies \text{map\_fixed } v l t = t$   
 $\langle \text{proof} \rangle$

**lemma** *map\_fixed\_idle\_closed*:  
 $\text{closed } t \implies \text{map\_fixed } \text{undefined } [] t = t$   
 $\langle \text{proof} \rangle$

**lemma** *map\_fixed\_inj\_closed*:  
 $\text{closed } t \implies \text{closed } u \implies \text{map\_fixed } \text{undefined } [] t = \text{map\_fixed } \text{undefined } [] u \implies t = u$   
 $\langle \text{proof} \rangle$

**lemma** *hash\_eq\_hash\_real\_closed*:  
**assumes** *closed t*  
**shows**  $\text{hash } t = \text{hash\_real } t$   
 $\langle \text{proof} \rangle$

## 3.2 Substitution

**nominal\_function** *subst\_term* ::  $\text{term} \Rightarrow \text{term} \Rightarrow \text{var} \Rightarrow \text{term} \Rightarrow \text{term} \langle \_ \_ ' / \_ \rangle [250, 200, 200] 250$  **where**

$\text{Unit}[t' / x] = \text{Unit} \mid$   
 $(\text{Var } y)[t' / x] = (\text{if } x = y \text{ then } t' \text{ else } \text{Var } y) \mid$   
 $\text{atom } y \# (x, t') \implies (\text{Lam } y t)[t' / x] = \text{Lam } y (t[t' / x]) \mid$   
 $\text{atom } y \# (x, t') \implies (\text{Rec } y t)[t' / x] = \text{Rec } y (t[t' / x]) \mid$   
 $(\text{Inj1 } t)[t' / x] = \text{Inj1 } (t[t' / x]) \mid$   
 $(\text{Inj2 } t)[t' / x] = \text{Inj2 } (t[t' / x]) \mid$   
 $(\text{Pair } t1 t2)[t' / x] = \text{Pair } (t1[t' / x]) (t2[t' / x]) \mid$   
 $(\text{Roll } t)[t' / x] = \text{Roll } (t[t' / x]) \mid$   
 $\text{atom } y \# (x, t') \implies (\text{Let } t1 y t2)[t' / x] = \text{Let } (t1[t' / x]) y (t2[t' / x]) \mid$   
 $(\text{App } t1 t2)[t' / x] = \text{App } (t1[t' / x]) (t2[t' / x]) \mid$   
 $(\text{Case } t1 t2 t3)[t' / x] = \text{Case } (t1[t' / x]) (t2[t' / x]) (t3[t' / x]) \mid$   
 $(\text{Prj1 } t)[t' / x] = \text{Prj1 } (t[t' / x]) \mid$   
 $(\text{Prj2 } t)[t' / x] = \text{Prj2 } (t[t' / x]) \mid$   
 $(\text{Unroll } t)[t' / x] = \text{Unroll } (t[t' / x]) \mid$   
 $(\text{Auth } t)[t' / x] = \text{Auth } (t[t' / x]) \mid$   
 $(\text{Unauth } t)[t' / x] = \text{Unauth } (t[t' / x]) \mid$   
 $(\text{Hash } h)[t' / x] = \text{Hash } h \mid$   
 $(\text{Hashed } h t)[t' / x] = \text{Hashed } h (t[t' / x])$

$\langle \text{proof} \rangle$   
**nominal\_termination** (*eqvt*)  
 $\langle \text{proof} \rangle$

**type\_synonym** *tenv* =  $(\text{var}, \text{term}) \text{ fmap}$

**nominal\_function** *psubst\_term* :: *term*  $\Rightarrow$  *tenv*  $\Rightarrow$  *term* **where**  
*psubst\_term* *Unit* *f* = *Unit* |  
*psubst\_term* (*Var* *y*) *f* = (*case* *f* \$\$ *y* of *Some* *t*  $\Rightarrow$  *t* | *None*  $\Rightarrow$  *Var* *y*) |  
*atom* *y* # *f*  $\Longrightarrow$  *psubst\_term* (*Lam* *y* *t*) *f* = *Lam* *y* (*psubst\_term* *t* *f*) |  
*atom* *y* # *f*  $\Longrightarrow$  *psubst\_term* (*Rec* *y* *t*) *f* = *Rec* *y* (*psubst\_term* *t* *f*) |  
*psubst\_term* (*Inj1* *t*) *f* = *Inj1* (*psubst\_term* *t* *f*) |  
*psubst\_term* (*Inj2* *t*) *f* = *Inj2* (*psubst\_term* *t* *f*) |  
*psubst\_term* (*Pair* *t1* *t2*) *f* = *Pair* (*psubst\_term* *t1* *f*) (*psubst\_term* *t2* *f*) |  
*psubst\_term* (*Roll* *t*) *f* = *Roll* (*psubst\_term* *t* *f*) |  
*atom* *y* # *f*  $\Longrightarrow$  *psubst\_term* (*Let* *t1* *y* *t2*) *f* = *Let* (*psubst\_term* *t1* *f*) *y* (*psubst\_term* *t2* *f*) |  
*psubst\_term* (*App* *t1* *t2*) *f* = *App* (*psubst\_term* *t1* *f*) (*psubst\_term* *t2* *f*) |  
*psubst\_term* (*Case* *t1* *t2* *t3*) *f* = *Case* (*psubst\_term* *t1* *f*) (*psubst\_term* *t2* *f*) (*psubst\_term* *t3* *f*) |  
*psubst\_term* (*Prj1* *t*) *f* = *Prj1* (*psubst\_term* *t* *f*) |  
*psubst\_term* (*Prj2* *t*) *f* = *Prj2* (*psubst\_term* *t* *f*) |  
*psubst\_term* (*Unroll* *t*) *f* = *Unroll* (*psubst\_term* *t* *f*) |  
*psubst\_term* (*Auth* *t*) *f* = *Auth* (*psubst\_term* *t* *f*) |  
*psubst\_term* (*Unauth* *t*) *f* = *Unauth* (*psubst\_term* *t* *f*) |  
*psubst\_term* (*Hash* *h*) *f* = *Hash* *h* |  
*psubst\_term* (*Hashed* *h* *t*) *f* = *Hashed* *h* (*psubst\_term* *t* *f*)  
<proof>

**nominal\_termination** (*eqvt*)  
<proof>

**nominal\_function** *subst\_type* :: *ty*  $\Rightarrow$  *ty*  $\Rightarrow$  *tvar*  $\Rightarrow$  *ty* **where**  
*subst\_type* *One* *t' x* = *One* |  
*subst\_type* (*Fun* *t1* *t2*) *t' x* = *Fun* (*subst\_type* *t1* *t' x*) (*subst\_type* *t2* *t' x*) |  
*subst\_type* (*Sum* *t1* *t2*) *t' x* = *Sum* (*subst\_type* *t1* *t' x*) (*subst\_type* *t2* *t' x*) |  
*subst\_type* (*Prod* *t1* *t2*) *t' x* = *Prod* (*subst\_type* *t1* *t' x*) (*subst\_type* *t2* *t' x*) |  
*atom* *y* # (*t', x*)  $\Longrightarrow$  *subst\_type* (*Mu* *y* *t*) *t' x* = *Mu* *y* (*subst\_type* *t* *t' x*) |  
*subst\_type* (*Alpha* *y*) *t' x* = (*if* *y* = *x* *then* *t'* *else* *Alpha* *y*) |  
*subst\_type* (*AuthT* *t*) *t' x* = *AuthT* (*subst\_type* *t* *t' x*)  
<proof>

**nominal\_termination** (*eqvt*)  
<proof>

**lemma** *fresh\_subst\_term*: *atom* *x* # *t*[*t' / x*]  $\longleftrightarrow$  (*x* = *x'*  $\vee$  *atom* *x* # *t*)  $\wedge$  (*atom* *x'* # *t*  $\vee$  *atom* *x* # *t'*)  
<proof>

**lemma** *term\_fresh\_subst[simp]*: *atom* *x* # *t*  $\Longrightarrow$  *atom* *x* # *s*  $\Longrightarrow$  (*atom* (*x::var*)) # *t*[*s / y*]  
<proof>

**lemma** *term\_subst\_idle[simp]*: *atom* *y* # *t*  $\Longrightarrow$  *t*[*s / y*] = *t*  
<proof>

**lemma** *term\_subst\_subst*: *atom* *y1*  $\neq$  *atom* *y2*  $\Longrightarrow$  *atom* *y1* # *s2*  $\Longrightarrow$  *t*[*s1 / y1*][*s2 / y2*] = *t*[*s2 / y2*][*s1*[*s2 / y2*] / *y1*]  
<proof>

**lemma** *fresh\_psubst*:  
**fixes** *x :: var*  
**assumes** *atom* *x* # *e* *atom* *x* # *vs*  
**shows** *atom* *x* # *psubst\_term* *e* *vs*  
<proof>

**lemma** *fresh\_subst\_type*:  
*atom*  $\alpha$  # *subst\_type*  $\tau$   $\tau'$   $\alpha'$   $\longleftrightarrow$  (( $\alpha$  =  $\alpha'$   $\vee$  *atom*  $\alpha$  #  $\tau$ )  $\wedge$  (*atom*  $\alpha'$  #  $\tau$   $\vee$  *atom*  $\alpha$  #  $\tau'$ ))  
<proof>

**lemma** *type\_fresh\_subst*[simp]:  $atom\ x \# t \implies atom\ x \# s \implies (atom\ (x::tvar)) \# subst\_type\ t\ s\ y$   
 ⟨proof⟩

**lemma** *type\_subst\_idle*[simp]:  $atom\ y \# t \implies subst\_type\ t\ s\ y = t$   
 ⟨proof⟩

**lemma** *type\_subst\_subst*:  $atom\ y1 \neq atom\ y2 \implies atom\ y1 \# s2 \implies$   
 $subst\_type\ (subst\_type\ t\ s1\ y1)\ s2\ y2 = subst\_type\ (subst\_type\ t\ s2\ y2)\ (subst\_type\ s1\ s2\ y2)\ y1$   
 ⟨proof⟩

### 3.3 Weak Typing Judgement

**type\_synonym** *tyenv* = (var, ty) fmap

**inductive** *judge\_weak* :: tyenv  $\Rightarrow$  term  $\Rightarrow$  ty  $\Rightarrow$  bool (⟨\_  $\vdash_W$  \_ : \_⟩ [150,0,150] 149) **where**

*jw\_Unit*:  $\Gamma \vdash_W Unit : One$  |  
*jw\_Var*:  $\llbracket \Gamma \ \$\$ x = Some\ \tau \rrbracket$   
 $\implies \Gamma \vdash_W Var\ x : \tau$  |  
*jw\_Lam*:  $\llbracket atom\ x \# \Gamma; \Gamma(x\ \$\$ := \tau_1) \vdash_W e : \tau_2 \rrbracket$   
 $\implies \Gamma \vdash_W Lam\ x\ e : Fun\ \tau_1\ \tau_2$  |  
*jw\_App*:  $\llbracket \Gamma \vdash_W e : Fun\ \tau_1\ \tau_2; \Gamma \vdash_W e' : \tau_1 \rrbracket$   
 $\implies \Gamma \vdash_W App\ e\ e' : \tau_2$  |  
*jw\_Let*:  $\llbracket atom\ x \# (\Gamma, e_1); \Gamma \vdash_W e_1 : \tau_1; \Gamma(x\ \$\$ := \tau_1) \vdash_W e_2 : \tau_2 \rrbracket$   
 $\implies \Gamma \vdash_W Let\ e_1\ x\ e_2 : \tau_2$  |  
*jw\_Rec*:  $\llbracket atom\ x \# \Gamma; atom\ y \# (\Gamma, x); \Gamma(x\ \$\$ := Fun\ \tau_1\ \tau_2) \vdash_W Lam\ y\ e : Fun\ \tau_1\ \tau_2 \rrbracket$   
 $\implies \Gamma \vdash_W Rec\ x\ (Lam\ y\ e) : Fun\ \tau_1\ \tau_2$  |  
*jw\_Inj1*:  $\llbracket \Gamma \vdash_W e : \tau_1 \rrbracket$   
 $\implies \Gamma \vdash_W Inj1\ e : Sum\ \tau_1\ \tau_2$  |  
*jw\_Inj2*:  $\llbracket \Gamma \vdash_W e : \tau_2 \rrbracket$   
 $\implies \Gamma \vdash_W Inj2\ e : Sum\ \tau_1\ \tau_2$  |  
*jw\_Case*:  $\llbracket \Gamma \vdash_W e : Sum\ \tau_1\ \tau_2; \Gamma \vdash_W e_1 : Fun\ \tau_1\ \tau; \Gamma \vdash_W e_2 : Fun\ \tau_2\ \tau \rrbracket$   
 $\implies \Gamma \vdash_W Case\ e\ e_1\ e_2 : \tau$  |  
*jw\_Pair*:  $\llbracket \Gamma \vdash_W e_1 : \tau_1; \Gamma \vdash_W e_2 : \tau_2 \rrbracket$   
 $\implies \Gamma \vdash_W Pair\ e_1\ e_2 : Prod\ \tau_1\ \tau_2$  |  
*jw\_Prj1*:  $\llbracket \Gamma \vdash_W e : Prod\ \tau_1\ \tau_2 \rrbracket$   
 $\implies \Gamma \vdash_W Prj1\ e : \tau_1$  |  
*jw\_Prj2*:  $\llbracket \Gamma \vdash_W e : Prod\ \tau_1\ \tau_2 \rrbracket$   
 $\implies \Gamma \vdash_W Prj2\ e : \tau_2$  |  
*jw\_Roll*:  $\llbracket atom\ \alpha \# \Gamma; \Gamma \vdash_W e : subst\_type\ \tau\ (Mu\ \alpha\ \tau)\ \alpha \rrbracket$   
 $\implies \Gamma \vdash_W Roll\ e : Mu\ \alpha\ \tau$  |  
*jw\_Unroll*:  $\llbracket atom\ \alpha \# \Gamma; \Gamma \vdash_W e : Mu\ \alpha\ \tau \rrbracket$   
 $\implies \Gamma \vdash_W Unroll\ e : subst\_type\ \tau\ (Mu\ \alpha\ \tau)\ \alpha$  |  
*jw\_Auth*:  $\llbracket \Gamma \vdash_W e : \tau \rrbracket$   
 $\implies \Gamma \vdash_W Auth\ e : \tau$  |  
*jw\_Unauth*:  $\llbracket \Gamma \vdash_W e : \tau \rrbracket$   
 $\implies \Gamma \vdash_W Unauth\ e : \tau$

**declare** *judge\_weak.intros*[simp]

**declare** *judge\_weak.intros*[intro]

**equivariance** *judge\_weak*

**nominal\_inductive** *judge\_weak*

**avoids** *jw\_Lam*:  $x$   
 | *jw\_Rec*:  $x$  **and**  $y$   
 | *jw\_Let*:  $x$   
 | *jw\_Roll*:  $\alpha$   
 | *jw\_Unroll*:  $\alpha$

*<proof>*

Inversion rules for typing judgment.

**inductive\_cases** *jw\_Unit\_inv[elim]*:  $\Gamma \vdash_W \text{Unit} : \tau$

**inductive\_cases** *jw\_Var\_inv[elim]*:  $\Gamma \vdash_W \text{Var } x : \tau$

**lemma** *jw\_Lam\_inv[elim]*:

**assumes**  $\Gamma \vdash_W \text{Lam } x \ e : \tau$

**and**  $\text{atom } x \# \Gamma$

**obtains**  $\tau_1 \ \tau_2$  **where**  $\tau = \text{Fun } \tau_1 \ \tau_2 \ (\Gamma(x \ \$\$ := \tau_1)) \vdash_W \ e : \tau_2$

*<proof>*

**lemma** *swap\_permute\_swap*:  $\text{atom } x \# \pi \implies \text{atom } y \# \pi \implies (x \leftrightarrow y) \cdot \pi \cdot (x \leftrightarrow y) \cdot t = \pi \cdot t$

*<proof>*

**lemma** *jw\_Rec\_inv[elim]*:

**assumes**  $\Gamma \vdash_W \text{Rec } x \ t : \tau$

**and**  $\text{atom } x \# \Gamma$

**obtains**  $y \ e \ \tau_1 \ \tau_2$  **where**  $\text{atom } y \# (\Gamma, x) \ t = \text{Lam } y \ e \ \tau = \text{Fun } \tau_1 \ \tau_2 \ \Gamma(x \ \$\$ := \text{Fun } \tau_1 \ \tau_2) \vdash_W \text{Lam } y \ e : \text{Fun } \tau_1 \ \tau_2$

*<proof>*

**inductive\_cases** *jw\_Inj1\_inv[elim]*:  $\Gamma \vdash_W \text{Inj1 } e : \tau$

**inductive\_cases** *jw\_Inj2\_inv[elim]*:  $\Gamma \vdash_W \text{Inj2 } e : \tau$

**inductive\_cases** *jw\_Pair\_inv[elim]*:  $\Gamma \vdash_W \text{Pair } e_1 \ e_2 : \tau$

**lemma** *jw\_Let\_inv[elim]*:

**assumes**  $\Gamma \vdash_W \text{Let } e_1 \ x \ e_2 : \tau_2$

**and**  $\text{atom } x \# (e_1, \Gamma)$

**obtains**  $\tau_1$  **where**  $\Gamma \vdash_W \ e_1 : \tau_1 \ \Gamma(x \ \$\$ := \tau_1) \vdash_W \ e_2 : \tau_2$

*<proof>*

**inductive\_cases** *jw\_Prj1\_inv[elim]*:  $\Gamma \vdash_W \text{Prj1 } e : \tau_1$

**inductive\_cases** *jw\_Prj2\_inv[elim]*:  $\Gamma \vdash_W \text{Prj2 } e : \tau_2$

**inductive\_cases** *jw\_App\_inv[elim]*:  $\Gamma \vdash_W \text{App } e \ e' : \tau_2$

**inductive\_cases** *jw\_Case\_inv[elim]*:  $\Gamma \vdash_W \text{Case } e \ e_1 \ e_2 : \tau$

**inductive\_cases** *jw\_Auth\_inv[elim]*:  $\Gamma \vdash_W \text{Auth } e : \tau$

**inductive\_cases** *jw\_Unauth\_inv[elim]*:  $\Gamma \vdash_W \text{Unauth } e : \tau$

**lemma** *subst\_type\_perm\_eq*:

**assumes**  $\text{atom } b \# t$

**shows**  $\text{subst\_type } t \ (\text{Mu } a \ t) \ a = \text{subst\_type } ((a \leftrightarrow b) \cdot t) \ (\text{Mu } b \ ((a \leftrightarrow b) \cdot t)) \ b$

*<proof>*

**lemma** *jw\_Roll\_inv[elim]*:

**assumes**  $\Gamma \vdash_W \text{Roll } e : \tau$

**and**  $\text{atom } \alpha \# (\Gamma, \tau)$

**obtains**  $\tau'$  **where**  $\tau = \text{Mu } \alpha \ \tau' \ \Gamma \vdash_W \ e : \text{subst\_type } \tau' \ (\text{Mu } \alpha \ \tau') \ \alpha$

*<proof>*

**lemma** *jw\_Unroll\_inv[elim]*:

**assumes**  $\Gamma \vdash_W \text{Unroll } e : \tau$

**and**  $\text{atom } \alpha \# (\Gamma, \tau)$

**obtains**  $\tau'$  **where**  $\tau = \text{subst\_type } \tau' \ (\text{Mu } \alpha \ \tau') \ \alpha \ \Gamma \vdash_W \ e : \text{Mu } \alpha \ \tau'$

*<proof>*

Additional inversion rules based on type rather than term.

**inductive\_cases** *jw\_Prod\_inv[elim]*:  $\{\$\$ \} \vdash_W \ e : \text{Prod } \tau_1 \ \tau_2$

**inductive\_cases** *jw\_Sum\_inv[elim]*:  $\{\$\$\} \vdash_W e : \text{Sum } \tau_1 \tau_2$

**lemma** *jw\_Fun\_inv[elim]*:

**assumes**  $\{\$\$\} \vdash_W v : \text{Fun } \tau_1 \tau_2$  *value*  $v$

**obtains**  $e \ x$  **where**  $v = \text{Lam } x \ e \vee v = \text{Rec } x \ e \ \text{atom } x \ \# (c::\text{term})$   
 $\langle \text{proof} \rangle$

**lemma** *jw\_Mu\_inv[elim]*:

**assumes**  $\{\$\$\} \vdash_W v : \text{Mu } \alpha \ \tau$  *value*  $v$

**obtains**  $v'$  **where**  $v = \text{Roll } v'$   
 $\langle \text{proof} \rangle$

### 3.4 Erasure of Authenticated Types

**nominal\_function** *erase* ::  $ty \Rightarrow ty$  **where**

*erase*  $\text{One} = \text{One}$  |  
*erase*  $(\text{Fun } \tau_1 \tau_2) = \text{Fun } (\text{erase } \tau_1) (\text{erase } \tau_2)$  |  
*erase*  $(\text{Sum } \tau_1 \tau_2) = \text{Sum } (\text{erase } \tau_1) (\text{erase } \tau_2)$  |  
*erase*  $(\text{Prod } \tau_1 \tau_2) = \text{Prod } (\text{erase } \tau_1) (\text{erase } \tau_2)$  |  
*erase*  $(\text{Mu } \alpha \ \tau) = \text{Mu } \alpha \ (\text{erase } \tau)$  |  
*erase*  $(\text{Alpha } \alpha) = \text{Alpha } \alpha$  |  
*erase*  $(\text{AuthT } \tau) = \text{erase } \tau$   
 $\langle \text{proof} \rangle$

**nominal\_termination** (*eqvt*)

$\langle \text{proof} \rangle$

**lemma** *fresh\_erase\_fresh*:

**assumes**  $\text{atom } x \ \# \ \tau$

**shows**  $\text{atom } x \ \# \ \text{erase } \tau$   
 $\langle \text{proof} \rangle$

**lemma** *fresh\_fmmap\_erase\_fresh*:

**assumes**  $\text{atom } x \ \# \ \Gamma$

**shows**  $\text{atom } x \ \# \ \text{fmmap } \text{erase } \Gamma$   
 $\langle \text{proof} \rangle$

**lemma** *erase\_subst\_type\_shift[simp]*:

*erase*  $(\text{subst\_type } \tau \ \tau' \ \alpha) = \text{subst\_type } (\text{erase } \tau) (\text{erase } \tau') \ \alpha$   
 $\langle \text{proof} \rangle$

**definition** *erase\_env* ::  $tyenv \Rightarrow tyenv$  **where**

*erase\_env* = *fmmap erase*

### 3.5 Strong Typing Judgement

**inductive** *judge* ::  $tyenv \Rightarrow \text{term} \Rightarrow ty \Rightarrow \text{bool}$  ( $\langle \_ \vdash \_ : \_ \rangle$  [150,0,150] 149) **where**

*j\_Unit*:  $\Gamma \vdash \text{Unit} : \text{One}$  |

*j\_Var*:  $\llbracket \Gamma \ \$\$ \ x = \text{Some } \tau \rrbracket$   
 $\implies \Gamma \vdash \text{Var } x : \tau$  |

*j\_Lam*:  $\llbracket \text{atom } x \ \# \ \Gamma; \Gamma(x \ \$\$ := \tau_1) \vdash e : \tau_2 \rrbracket$   
 $\implies \Gamma \vdash \text{Lam } x \ e : \text{Fun } \tau_1 \tau_2$  |

*j\_App*:  $\llbracket \Gamma \vdash e : \text{Fun } \tau_1 \tau_2; \Gamma \vdash e' : \tau_1 \rrbracket$   
 $\implies \Gamma \vdash \text{App } e \ e' : \tau_2$  |

*j\_Let*:  $\llbracket \text{atom } x \ \# \ (\Gamma, e_1); \Gamma \vdash e_1 : \tau_1; \Gamma(x \ \$\$ := \tau_1) \vdash e_2 : \tau_2 \rrbracket$   
 $\implies \Gamma \vdash \text{Let } e_1 \ x \ e_2 : \tau_2$  |

*j\_Rec*:  $\llbracket \text{atom } x \ \# \ \Gamma; \text{atom } y \ \# \ (\Gamma, x); \Gamma(x \ \$\$ := \text{Fun } \tau_1 \tau_2) \vdash \text{Lam } y \ e' : \text{Fun } \tau_1 \tau_2 \rrbracket$   
 $\implies \Gamma \vdash \text{Rec } x \ (\text{Lam } y \ e') : \text{Fun } \tau_1 \tau_2$  |

*j\_Inj1*:  $\llbracket \Gamma \vdash e : \tau_1 \rrbracket$   
 $\implies \Gamma \vdash \text{Inj1 } e : \text{Sum } \tau_1 \tau_2$  |

```

j_Inj2:  [ Γ ⊢ e : τ2 ]
        ⇒ Γ ⊢ Inj2 e : Sum τ1 τ2 |
j_Case:  [ Γ ⊢ e : Sum τ1 τ2; Γ ⊢ e1 : Fun τ1 τ; Γ ⊢ e2 : Fun τ2 τ ]
        ⇒ Γ ⊢ Case e e1 e2 : τ |
j_Pair:  [ Γ ⊢ e1 : τ1; Γ ⊢ e2 : τ2 ]
        ⇒ Γ ⊢ Pair e1 e2 : Prod τ1 τ2 |
j_Prj1:  [ Γ ⊢ e : Prod τ1 τ2 ]
        ⇒ Γ ⊢ Prj1 e : τ1 |
j_Prj2:  [ Γ ⊢ e : Prod τ1 τ2 ]
        ⇒ Γ ⊢ Prj2 e : τ2 |
j_Roll:  [ atom α ‡ Γ; Γ ⊢ e : subst_type τ (Mu α τ) α ]
        ⇒ Γ ⊢ Roll e : Mu α τ |
j_Unroll: [ atom α ‡ Γ; Γ ⊢ e : Mu α τ ]
        ⇒ Γ ⊢ Unroll e : subst_type τ (Mu α τ) α |
j_Auth:  [ Γ ⊢ e : τ ]
        ⇒ Γ ⊢ Auth e : AuthT τ |
j_Unauth: [ Γ ⊢ e : AuthT τ ]
        ⇒ Γ ⊢ Unauth e : τ

```

**declare** *judge.intros*[*intro*]

**equivariance** *judge*

**nominal\_inductive** *judge*

```

avoids j_Lam: x
      | j_Rec: x and y
      | j_Let: x
      | j_Roll: α
      | j_Unroll: α
⟨proof⟩

```

**lemma** *judge\_imp\_judge\_weak*:

```

assumes Γ ⊢ e : τ
shows   erase_env Γ ⊢W e : erase τ
⟨proof⟩

```

### 3.6 Shallow Projection

**nominal\_function** *shallow* :: *term* ⇒ *term* (⟨⟨\_⟩⟩) **where**

```

⟨Unit⟩ = Unit |
⟨Var v⟩ = Var v |
⟨Lam x e⟩ = Lam x ⟨e⟩ |
⟨Rec x e⟩ = Rec x ⟨e⟩ |
⟨Inj1 e⟩ = Inj1 ⟨e⟩ |
⟨Inj2 e⟩ = Inj2 ⟨e⟩ |
⟨Pair e1 e2⟩ = Pair ⟨e1⟩ ⟨e2⟩ |
⟨Roll e⟩ = Roll ⟨e⟩ |
⟨Let e1 x e2⟩ = Let ⟨e1⟩ x ⟨e2⟩ |
⟨App e1 e2⟩ = App ⟨e1⟩ ⟨e2⟩ |
⟨Case e e1 e2⟩ = Case ⟨e⟩ ⟨e1⟩ ⟨e2⟩ |
⟨Prj1 e⟩ = Prj1 ⟨e⟩ |
⟨Prj2 e⟩ = Prj2 ⟨e⟩ |
⟨Unroll e⟩ = Unroll ⟨e⟩ |
⟨Auth e⟩ = Auth ⟨e⟩ |
⟨Unauth e⟩ = Unauth ⟨e⟩ |
— No rule is defined for Hash, but: "[..] preserving that structure in every case but that of <h, v> [..]"
⟨Hash h⟩ = Hash h |
⟨Hashed h e⟩ = Hash h
⟨proof⟩

```

**nominal\_termination** (*eqvt*)  
 ⟨*proof*⟩

**lemma** *fresh\_shallow*:  $\text{atom } x \# e \implies \text{atom } x \# (|e|)$   
 ⟨*proof*⟩

### 3.7 Small-step Semantics

**datatype** *mode* = *I* | *P* | *V* — Ideal, Prover and Verifier modes

**instantiation** *mode* :: *pure*

**begin**

**definition** *permute\_mode* :: *perm*  $\Rightarrow$  *mode*  $\Rightarrow$  *mode* **where**

*permute\_mode*  $\pi$  *h* = *h*

**instance** ⟨*proof*⟩

**end**

**type\_synonym** *proofstream* = *term list*

**inductive** *smallstep* :: *proofstream*  $\Rightarrow$  *term*  $\Rightarrow$  *mode*  $\Rightarrow$  *proofstream*  $\Rightarrow$  *term*  $\Rightarrow$  *bool* (⟨ $\llcorner$ ,  $\lrcorner$ ⟩  $\_ \rightarrow$   $\llcorner$ ,  $\lrcorner$ ⟩) **where**

*s\_App1*:  $\llcorner \llcorner \pi, e_1 \gg \gg m \rightarrow \llcorner \llcorner \pi', e_1' \gg \gg \llcorner \llcorner$   
 $\implies \llcorner \llcorner \pi, \text{App } e_1 e_2 \gg \gg m \rightarrow \llcorner \llcorner \pi', \text{App } e_1' e_2' \gg \gg \llcorner \llcorner$  |  
*s\_App2*:  $\llcorner \llcorner \text{value } v_1; \llcorner \llcorner \pi, e_2 \gg \gg m \rightarrow \llcorner \llcorner \pi', e_2' \gg \gg \llcorner \llcorner$   
 $\implies \llcorner \llcorner \pi, \text{App } v_1 e_2 \gg \gg m \rightarrow \llcorner \llcorner \pi', \text{App } v_1 e_2' \gg \gg \llcorner \llcorner$  |  
*s\_AppLam*:  $\llcorner \llcorner \text{value } v; \text{atom } x \# (v, \pi) \llcorner \llcorner$   
 $\implies \llcorner \llcorner \pi, \text{App } (\text{Lam } x e) v \gg \gg \_ \rightarrow \llcorner \llcorner \pi, e[v / x] \gg \gg \llcorner \llcorner$  |  
*s\_AppRec*:  $\llcorner \llcorner \text{value } v; \text{atom } x \# (v, \pi) \llcorner \llcorner$   
 $\implies \llcorner \llcorner \pi, \text{App } (\text{Rec } x e) v \gg \gg \_ \rightarrow \llcorner \llcorner \pi, \text{App } (e[(\text{Rec } x e) / x]) v \gg \gg \llcorner \llcorner$  |  
*s\_Let1*:  $\llcorner \llcorner \text{atom } x \# (e_1, e_1', \pi, \pi'); \llcorner \llcorner \pi, e_1 \gg \gg m \rightarrow \llcorner \llcorner \pi', e_1' \gg \gg \llcorner \llcorner$   
 $\implies \llcorner \llcorner \pi, \text{Let } e_1 x e_2 \gg \gg m \rightarrow \llcorner \llcorner \pi', \text{Let } e_1' x e_2 \gg \gg \llcorner \llcorner$  |  
*s\_Let2*:  $\llcorner \llcorner \text{value } v; \text{atom } x \# (v, \pi) \llcorner \llcorner$   
 $\implies \llcorner \llcorner \pi, \text{Let } v x e \gg \gg \_ \rightarrow \llcorner \llcorner \pi, e[v / x] \gg \gg \llcorner \llcorner$  |  
*s\_Inj1*:  $\llcorner \llcorner \llcorner \llcorner \pi, e \gg \gg m \rightarrow \llcorner \llcorner \llcorner \llcorner \pi', e' \gg \gg \llcorner \llcorner$   
 $\implies \llcorner \llcorner \llcorner \llcorner \pi, \text{Inj1 } e \gg \gg m \rightarrow \llcorner \llcorner \llcorner \llcorner \pi', \text{Inj1 } e' \gg \gg \llcorner \llcorner$  |  
*s\_Inj2*:  $\llcorner \llcorner \llcorner \llcorner \pi, e \gg \gg m \rightarrow \llcorner \llcorner \llcorner \llcorner \pi', e' \gg \gg \llcorner \llcorner$   
 $\implies \llcorner \llcorner \llcorner \llcorner \pi, \text{Inj2 } e \gg \gg m \rightarrow \llcorner \llcorner \llcorner \llcorner \pi', \text{Inj2 } e' \gg \gg \llcorner \llcorner$  |  
*s\_Case*:  $\llcorner \llcorner \llcorner \llcorner \llcorner \pi, e \gg \gg m \rightarrow \llcorner \llcorner \llcorner \llcorner \pi', e' \gg \gg \llcorner \llcorner$   
 $\implies \llcorner \llcorner \llcorner \llcorner \llcorner \pi, \text{Case } e e_1 e_2 \gg \gg m \rightarrow \llcorner \llcorner \llcorner \llcorner \pi', \text{Case } e' e_1 e_2 \gg \gg \llcorner \llcorner$  |  
 — Case rules are different from paper to account for recursive functions.  
*s\_CaseInj1*:  $\llcorner \llcorner \llcorner \llcorner \text{value } v \llcorner \llcorner$   
 $\implies \llcorner \llcorner \llcorner \llcorner \llcorner \pi, \text{Case } (\text{Inj1 } v) e_1 e_2 \gg \gg \_ \rightarrow \llcorner \llcorner \llcorner \llcorner \llcorner \pi, \text{App } e_1 v \gg \gg \llcorner \llcorner$  |  
*s\_CaseInj2*:  $\llcorner \llcorner \llcorner \llcorner \text{value } v \llcorner \llcorner$   
 $\implies \llcorner \llcorner \llcorner \llcorner \llcorner \pi, \text{Case } (\text{Inj2 } v) e_1 e_2 \gg \gg \_ \rightarrow \llcorner \llcorner \llcorner \llcorner \llcorner \pi, \text{App } e_2 v \gg \gg \llcorner \llcorner$  |  
*s\_Pair1*:  $\llcorner \llcorner \llcorner \llcorner \llcorner \llcorner \pi, e_1 \gg \gg m \rightarrow \llcorner \llcorner \llcorner \llcorner \llcorner \llcorner \pi', e_1' \gg \gg \llcorner \llcorner$   
 $\implies \llcorner \llcorner \llcorner \llcorner \llcorner \llcorner \pi, \text{Pair } e_1 e_2 \gg \gg m \rightarrow \llcorner \llcorner \llcorner \llcorner \llcorner \llcorner \pi', \text{Pair } e_1' e_2' \gg \gg \llcorner \llcorner$  |  
*s\_Pair2*:  $\llcorner \llcorner \llcorner \llcorner \llcorner \llcorner \text{value } v_1; \llcorner \llcorner \llcorner \llcorner \pi, e_2 \gg \gg m \rightarrow \llcorner \llcorner \llcorner \llcorner \llcorner \llcorner \pi', e_2' \gg \gg \llcorner \llcorner$   
 $\implies \llcorner \llcorner \llcorner \llcorner \llcorner \llcorner \llcorner \pi, \text{Pair } v_1 e_2 \gg \gg m \rightarrow \llcorner \llcorner \llcorner \llcorner \llcorner \llcorner \pi', \text{Pair } v_1 e_2' \gg \gg \llcorner \llcorner$  |  
*s\_Prj1*:  $\llcorner \llcorner \llcorner \llcorner \llcorner \llcorner \llcorner \llcorner \pi, e \gg \gg m \rightarrow \llcorner \llcorner \llcorner \llcorner \llcorner \llcorner \pi', e' \gg \gg \llcorner \llcorner$   
 $\implies \llcorner \llcorner \llcorner \llcorner \llcorner \llcorner \llcorner \llcorner \pi, \text{Prj1 } e \gg \gg m \rightarrow \llcorner \llcorner \llcorner \llcorner \llcorner \llcorner \pi', \text{Prj1 } e' \gg \gg \llcorner \llcorner$  |  
*s\_Prj2*:  $\llcorner \llcorner \llcorner \llcorner \llcorner \llcorner \llcorner \llcorner \llcorner \pi, e \gg \gg m \rightarrow \llcorner \llcorner \llcorner \llcorner \llcorner \llcorner \pi', e' \gg \gg \llcorner \llcorner$   
 $\implies \llcorner \llcorner \llcorner \llcorner \llcorner \llcorner \llcorner \llcorner \llcorner \pi, \text{Prj2 } e \gg \gg m \rightarrow \llcorner \llcorner \llcorner \llcorner \llcorner \llcorner \pi', \text{Prj2 } e' \gg \gg \llcorner \llcorner$  |  
*s\_PrjPair1*:  $\llcorner \llcorner \llcorner \llcorner \llcorner \llcorner \llcorner \llcorner \llcorner \text{value } v_1; \text{value } v_2 \llcorner \llcorner$   
 $\implies \llcorner \llcorner \llcorner \llcorner \llcorner \llcorner \llcorner \llcorner \llcorner \llcorner \llcorner \pi, \text{Prj1 } (\text{Pair } v_1 v_2) \gg \gg \_ \rightarrow \llcorner \llcorner \llcorner \llcorner \llcorner \llcorner \llcorner \llcorner \llcorner \pi, v_1 \gg \gg \llcorner \llcorner$  |  
*s\_PrjPair2*:  $\llcorner \llcorner \llcorner \llcorner \llcorner \llcorner \llcorner \llcorner \llcorner \text{value } v_1; \text{value } v_2 \llcorner \llcorner$   
 $\implies \llcorner \llcorner \llcorner \llcorner \llcorner \llcorner \llcorner \llcorner \llcorner \llcorner \llcorner \pi, \text{Prj2 } (\text{Pair } v_1 v_2) \gg \gg \_ \rightarrow \llcorner \llcorner \llcorner \llcorner \llcorner \llcorner \llcorner \llcorner \llcorner \pi, v_2 \gg \gg \llcorner \llcorner$  |  
*s\_Unroll*:  $\llcorner \llcorner \llcorner \llcorner \llcorner \llcorner \llcorner \llcorner \llcorner \llcorner \llcorner \pi, e \gg \gg m \rightarrow \llcorner \llcorner \llcorner \llcorner \llcorner \llcorner \llcorner \llcorner \llcorner \llcorner \pi', e' \gg \gg \llcorner \llcorner$   
 $\implies \llcorner \llcorner \llcorner \llcorner \llcorner \llcorner \llcorner \llcorner \llcorner \llcorner \llcorner \llcorner \pi, \text{Unroll } e \gg \gg m \rightarrow \llcorner \llcorner \llcorner \llcorner \llcorner \llcorner \llcorner \llcorner \llcorner \llcorner \pi', \text{Unroll } e' \gg \gg \llcorner \llcorner$  |

$s\_Roll: \quad \ll \pi, e \gg m \rightarrow \ll \pi', e' \gg$   
 $\quad \Rightarrow \ll \pi, Roll\ e \gg m \rightarrow \ll \pi', Roll\ e' \gg \mid$   
 $s\_UnrollRoll: \llbracket value\ v \rrbracket$   
 $\quad \Rightarrow \ll \pi, Unroll\ (Roll\ v) \gg \_ \rightarrow \ll \pi, v \gg \mid$   
— Mode-specific rules  
 $s\_Auth: \quad \ll \pi, e \gg m \rightarrow \ll \pi', e' \gg$   
 $\quad \Rightarrow \ll \pi, Auth\ e \gg m \rightarrow \ll \pi', Auth\ e' \gg \mid$   
 $s\_Unauth: \quad \ll \pi, e \gg m \rightarrow \ll \pi', e' \gg$   
 $\quad \Rightarrow \ll \pi, Unauth\ e \gg m \rightarrow \ll \pi', Unauth\ e' \gg \mid$   
 $s\_AuthI: \quad \llbracket value\ v \rrbracket$   
 $\quad \Rightarrow \ll \pi, Auth\ v \gg I \rightarrow \ll \pi, v \gg \mid$   
 $s\_UnauthI: \quad \llbracket value\ v \rrbracket$   
 $\quad \Rightarrow \ll \pi, Unauth\ v \gg I \rightarrow \ll \pi, v \gg \mid$   
 $s\_AuthP: \quad \llbracket closed\ (v); value\ v \rrbracket$   
 $\quad \Rightarrow \ll \pi, Auth\ v \gg P \rightarrow \ll \pi, Hashed\ (hash\ (v))\ v \gg \mid$   
 $s\_UnauthP: \quad \llbracket value\ v \rrbracket$   
 $\quad \Rightarrow \ll \pi, Unauth\ (Hashed\ h\ v) \gg P \rightarrow \ll \pi @ [(v)], v \gg \mid$   
 $s\_AuthV: \quad \llbracket closed\ v; value\ v \rrbracket$   
 $\quad \Rightarrow \ll \pi, Auth\ v \gg V \rightarrow \ll \pi, Hash\ (hash\ v) \gg \mid$   
 $s\_UnauthV: \quad \llbracket closed\ s_0; hash\ s_0 = h \rrbracket$   
 $\quad \Rightarrow \ll s_0 \# \pi, Unauth\ (Hash\ h) \gg V \rightarrow \ll \pi, s_0 \gg$

**declare** *smallstep.intros*[simp]  
**declare** *smallstep.intros*[intro]

**equivariance** *smallstep*  
**nominal\_inductive** *smallstep*  
**avoids** *s\_AppLam*: *x*  
 $\mid$  *s\_AppRec*: *x*  
 $\mid$  *s\_Let1*: *x*  
 $\mid$  *s\_Let2*: *x*  
*<proof>*

**inductive** *smallsteps* :: *proofstream*  $\Rightarrow$  *term*  $\Rightarrow$  *mode*  $\Rightarrow$  *nat*  $\Rightarrow$  *proofstream*  $\Rightarrow$  *term*  $\Rightarrow$  *bool* ( $\llbracket \_ \rrbracket$   
 $\_ \rightarrow \_ \llbracket \_ \rrbracket$ ) **where**  
 $s\_Id: \ll \pi, e \gg \_ \rightarrow 0 \ll \pi, e \gg \mid$   
 $s\_Tr: \llbracket \ll \pi_1, e_1 \gg m \rightarrow i \ll \pi_2, e_2 \gg; \ll \pi_2, e_2 \gg m \rightarrow \ll \pi_3, e_3 \gg \rrbracket$   
 $\quad \Rightarrow \ll \pi_1, e_1 \gg m \rightarrow (i+1) \ll \pi_3, e_3 \gg$

**declare** *smallsteps.intros*[simp]  
**declare** *smallsteps.intros*[intro]

**equivariance** *smallsteps*  
**nominal\_inductive** *smallsteps* *<proof>*

**lemma** *steps\_1\_step*[simp]:  $\ll \pi, e \gg m \rightarrow 1 \ll \pi', e' \gg = \ll \pi, e \gg m \rightarrow \ll \pi', e' \gg$  (**is**  $?L \leftrightarrow ?R$ )  
*<proof>*

Inversion rules for smallstep(s) predicates.

**lemma** *value\_no\_step*[intro]:  
**assumes**  $\ll \pi_1, v \gg m \rightarrow \ll \pi_2, t \gg value\ v$   
**shows** *False*  
*<proof>*

**lemma** *subst\_term\_perm*:  
**assumes** *atom*  $x' \# (x, e)$   
**shows**  $e[v / x] = ((x \leftrightarrow x') \cdot e)[v / x]$   
*<proof>*

**inductive\_cases**  $s\_Unit\_inv[elim]$ :  $\ll \pi_1, Unit \gg m \rightarrow \ll \pi_2, v \gg$

**inductive\_cases**  $s\_App\_inv[consumes\ 1, case\_names\ App1\ App2\ AppLam\ AppRec, elim]$ :  $\ll \pi, App\ v_1\ v_2 \gg m \rightarrow \ll \pi', e \gg$

**lemma**  $s\_Let\_inv'$ :

**assumes**  $\ll \pi, Let\ e_1\ x\ e_2 \gg m \rightarrow \ll \pi', e' \gg$

**and**  $atom\ x \# (e_1, \pi)$

**obtains**  $e_1'$  **where**  $(e' = e_2[e_1 / x] \wedge value\ e_1 \wedge \pi = \pi') \vee (\ll \pi, e_1 \gg m \rightarrow \ll \pi', e_1' \gg \wedge e' = Let\ e_1'\ x\ e_2 \wedge \neg value\ e_1)$

*<proof>*

**lemma**  $s\_Let\_inv[consumes\ 2, case\_names\ Let1\ Let2, elim]$ :

**assumes**  $\ll \pi, Let\ e_1\ x\ e_2 \gg m \rightarrow \ll \pi', e' \gg$

**and**  $atom\ x \# (e_1, \pi)$

**and**  $e' = e_2[e_1 / x] \wedge value\ e_1 \wedge \pi = \pi' \implies Q$

**and**  $\bigwedge e_1'. \ll \pi, e_1 \gg m \rightarrow \ll \pi', e_1' \gg \wedge e' = Let\ e_1'\ x\ e_2 \wedge \neg value\ e_1 \implies Q$

**shows**  $Q$

*<proof>*

**inductive\_cases**  $s\_Case\_inv[consumes\ 1, case\_names\ Case\ Inj1\ Inj2, elim]$ :

$\ll \pi, Case\ e\ e_1\ e_2 \gg m \rightarrow \ll \pi', e' \gg$

**inductive\_cases**  $s\_Prj1\_inv[consumes\ 1, case\_names\ Prj1\ PrjPair1, elim]$ :

$\ll \pi, Prj1\ e \gg m \rightarrow \ll \pi', v \gg$

**inductive\_cases**  $s\_Prj2\_inv[consumes\ 1, case\_names\ Prj2\ PrjPair2, elim]$ :

$\ll \pi, Prj2\ e \gg m \rightarrow \ll \pi', v \gg$

**inductive\_cases**  $s\_Pair\_inv[consumes\ 1, case\_names\ Pair1\ Pair2, elim]$ :

$\ll \pi, Pair\ e_1\ e_2 \gg m \rightarrow \ll \pi', e' \gg$

**inductive\_cases**  $s\_Inj1\_inv[consumes\ 1, case\_names\ Inj1, elim]$ :

$\ll \pi, Inj1\ e \gg m \rightarrow \ll \pi', e' \gg$

**inductive\_cases**  $s\_Inj2\_inv[consumes\ 1, case\_names\ Inj2, elim]$ :

$\ll \pi, Inj2\ e \gg m \rightarrow \ll \pi', e' \gg$

**inductive\_cases**  $s\_Roll\_inv[consumes\ 1, case\_names\ Roll, elim]$ :

$\ll \pi, Roll\ e \gg m \rightarrow \ll \pi', e' \gg$

**inductive\_cases**  $s\_Unroll\_inv[consumes\ 1, case\_names\ Unroll\ UnrollRoll, elim]$ :

$\ll \pi, Unroll\ e \gg m \rightarrow \ll \pi', e' \gg$

**inductive\_cases**  $s\_AuthI\_inv[consumes\ 1, case\_names\ Auth\ AuthI, elim]$ :

$\ll \pi, Auth\ e \gg I \rightarrow \ll \pi', e' \gg$

**inductive\_cases**  $s\_UnauthI\_inv[consumes\ 1, case\_names\ Unauth\ UnauthI, elim]$ :

$\ll \pi, Unauth\ e \gg I \rightarrow \ll \pi', e' \gg$

**inductive\_cases**  $s\_AuthP\_inv[consumes\ 1, case\_names\ Auth\ AuthP, elim]$ :

$\ll \pi, Auth\ e \gg P \rightarrow \ll \pi', e' \gg$

**inductive\_cases**  $s\_UnauthP\_inv[consumes\ 1, case\_names\ Unauth\ UnauthP, elim]$ :

$\ll \pi, Unauth\ e \gg P \rightarrow \ll \pi', e' \gg$

**inductive\_cases**  $s\_AuthV\_inv[consumes\ 1, case\_names\ Auth\ AuthV, elim]$ :

$\ll \pi, Auth\ e \gg V \rightarrow \ll \pi', e' \gg$

**inductive\_cases**  $s\_UnauthV\_inv[consumes\ 1, case\_names\ Unauth\ UnauthV, elim]$ :

$\ll \pi, Unauth\ e \gg V \rightarrow \ll \pi', e' \gg$

**inductive\_cases**  $s\_Id\_inv[elim]$ :  $\ll \pi_1, e_1 \gg m \rightarrow 0 \ll \pi_2, e_2 \gg$

**inductive\_cases**  $s\_Tr\_inv[elim]$ :  $\ll \pi_1, e_1 \gg m \rightarrow i \ll \pi_3, e_3 \gg$

Freshness with smallstep.

**lemma**  $fresh\_smallstep\_I$ :

**fixes**  $x :: var$

**assumes**  $\ll \pi, e \gg I \rightarrow \ll \pi', e' \gg atom\ x \# e$

**shows**  $atom\ x \# e'$

*<proof>*

**lemma** *fresh\_smallstep\_P*:

**fixes**  $x :: \text{var}$

**assumes**  $\ll \pi, e \gg P \rightarrow \ll \pi', e' \gg \text{atom } x \# e$

**shows**  $\text{atom } x \# e'$

*<proof>*

**lemma** *fresh\_smallsteps\_I*:

**fixes**  $x :: \text{var}$

**assumes**  $\ll \pi, e \gg I \rightarrow i \ll \pi', e' \gg \text{atom } x \# e$

**shows**  $\text{atom } x \# e'$

*<proof>*

**lemma** *fresh\_ps\_smallstep\_P*:

**fixes**  $x :: \text{var}$

**assumes**  $\ll \pi, e \gg P \rightarrow \ll \pi', e' \gg \text{atom } x \# e \text{ atom } x \# \pi$

**shows**  $\text{atom } x \# \pi'$

*<proof>*

Proofstream lemmas.

**lemma** *smallstepI\_ps\_eq*:

**assumes**  $\ll \pi, e \gg I \rightarrow \ll \pi', e' \gg$

**shows**  $\pi = \pi'$

*<proof>*

**lemma** *smallstepI\_ps\_emptyD*:

$\ll \pi, e \gg I \rightarrow \ll [], e' \gg \implies \ll [], e \gg I \rightarrow \ll [], e' \gg$

$\ll [], e \gg I \rightarrow \ll \pi, e' \gg \implies \ll [], e \gg I \rightarrow \ll [], e' \gg$

*<proof>*

**lemma** *smallstepsI\_ps\_eq*:

**assumes**  $\ll \pi, e \gg I \rightarrow i \ll \pi', e' \gg$

**shows**  $\pi = \pi'$

*<proof>*

**lemma** *smallstepsI\_ps\_emptyD*:

$\ll \pi, e \gg I \rightarrow i \ll [], e' \gg \implies \ll [], e \gg I \rightarrow i \ll [], e' \gg$

$\ll [], e \gg I \rightarrow i \ll \pi, e' \gg \implies \ll [], e \gg I \rightarrow i \ll [], e' \gg$

*<proof>*

**lemma** *smallstepV\_consumes\_proofstream*:

**assumes**  $\ll \pi_1, eV \gg V \rightarrow \ll \pi_2, eV' \gg$

**obtains**  $\pi$  **where**  $\pi_1 = \pi @ \pi_2$

*<proof>*

**lemma** *smallstepsV\_consumes\_proofstream*:

**assumes**  $\ll \pi_1, eV \gg V \rightarrow i \ll \pi_2, eV' \gg$

**obtains**  $\pi$  **where**  $\pi_1 = \pi @ \pi_2$

*<proof>*

**lemma** *smallstepP\_generates\_proofstream*:

**assumes**  $\ll \pi_1, eP \gg P \rightarrow \ll \pi_2, eP' \gg$

**obtains**  $\pi$  **where**  $\pi_2 = \pi_1 @ \pi$

*<proof>*

**lemma** *smallstepsP\_generates\_proofstream*:

**assumes**  $\ll \pi_1, eP \gg P \rightarrow i \ll \pi_2, eP' \gg$

**obtains**  $\pi$  **where**  $\pi_2 = \pi_1 @ \pi$   
 ⟨proof⟩

**lemma** *smallstepV\_ps\_append*:  
 $\ll \pi, eV \gg V \rightarrow \ll \pi', eV' \gg \longleftrightarrow \ll \pi @ X, eV \gg V \rightarrow \ll \pi' @ X, eV' \gg$  (is ?L  $\longleftrightarrow$  ?R)  
 ⟨proof⟩

**lemma** *smallstepV\_ps\_to\_suffix*:  
**assumes**  $\ll \pi, e \gg V \rightarrow \ll \pi' @ X, e' \gg$   
**obtains**  $\pi''$  **where**  $\pi = \pi'' @ X$   
 ⟨proof⟩

**lemma** *smallstepsV\_ps\_append*:  
 $\ll \pi, eV \gg V \rightarrow i \ll \pi', eV' \gg \longleftrightarrow \ll \pi @ X, eV \gg V \rightarrow i \ll \pi' @ X, eV' \gg$  (is ?L  $\longleftrightarrow$  ?R)  
 ⟨proof⟩

**lemma** *smallstepP\_ps\_prepend*:  
 $\ll \pi, eP \gg P \rightarrow \ll \pi', eP' \gg \longleftrightarrow \ll X @ \pi, eP \gg P \rightarrow \ll X @ \pi', eP' \gg$  (is ?L  $\longleftrightarrow$  ?R)  
 ⟨proof⟩

**lemma** *smallstepsP\_ps\_prepend*:  
 $\ll \pi, eP \gg P \rightarrow i \ll \pi', eP' \gg \longleftrightarrow \ll X @ \pi, eP \gg P \rightarrow i \ll X @ \pi', eP' \gg$  (is ?L  $\longleftrightarrow$  ?R)  
 ⟨proof⟩

### 3.8 Type Progress

**lemma** *type\_progress*:  
**assumes**  $\{\$\$ \} \vdash_W e : \tau$   
**shows**  $\text{value } e \vee (\exists e'. \ll [], e \gg I \rightarrow \ll [], e' \gg)$   
 ⟨proof⟩

### 3.9 Weak Type Preservation

**lemma** *fresh\_tyenv\_None*:  
**fixes**  $\Gamma :: \text{tyenv}$   
**shows**  $\text{atom } x \# \Gamma \longleftrightarrow \Gamma \ \$\$ x = \text{None}$  (is ?L  $\longleftrightarrow$  ?R)  
 ⟨proof⟩

**lemma** *judge\_weak\_fresh\_env\_fresh\_term[dest]*:  
**fixes**  $a :: \text{var}$   
**assumes**  $\Gamma \vdash_W e : \tau$   $\text{atom } a \# \Gamma$   
**shows**  $\text{atom } a \# e$   
 ⟨proof⟩

**lemma** *judge\_weak\_weakening\_1*:  
**assumes**  $\Gamma \vdash_W e : \tau$   $\text{atom } y \# e$   
**shows**  $\Gamma(y \ \$\$ := \tau') \vdash_W e : \tau$   
 ⟨proof⟩

**lemma** *judge\_weak\_weakening\_2*:  
**assumes**  $\Gamma \vdash_W e : \tau$   $\text{atom } y \# \Gamma$   
**shows**  $\Gamma(y \ \$\$ := \tau') \vdash_W e : \tau$   
 ⟨proof⟩

**lemma** *judge\_weak\_weakening\_env*:  
**assumes**  $\{\$\$ \} \vdash_W e : \tau$   
**shows**  $\Gamma \vdash_W e : \tau$   
 ⟨proof⟩

**lemma** *value\_subst\_value*:

**assumes** *value e value e'*  
**shows** *value (e[e' / x])*  
*<proof>*

**lemma** *judge\_weak\_subst[intro]*:

**assumes**  $\Gamma(a \text{ \textit{\$} \$} = \tau') \vdash_W e : \tau \text{ \textit{\$} \$} \vdash_W e' : \tau'$   
**shows**  $\Gamma \vdash_W e[e' / a] : \tau$   
*<proof>*

**lemma** *type\_preservation*:

**assumes**  $\ll [], e \gg I \rightarrow \ll [], e' \gg \text{ \textit{\$} \$} \vdash_W e : \tau$   
**shows**  $\text{ \textit{\$} \$} \vdash_W e' : \tau$   
*<proof>*

### 3.10 Corrected Lemma 1 from Miller et al. [2]: Weak Type Soundness

**lemma** *type\_soundness*:

**assumes**  $\text{ \textit{\$} \$} \vdash_W e : \tau$   
**shows** *value e*  $\vee (\exists e'. \ll [], e \gg I \rightarrow \ll [], e' \gg \wedge \text{ \textit{\$} \$} \vdash_W e' : \tau)$   
*<proof>*

## 4 Agreement Relation

**inductive** *agree* :: *tyenv*  $\Rightarrow$  *term*  $\Rightarrow$  *term*  $\Rightarrow$  *term*  $\Rightarrow$  *ty*  $\Rightarrow$  *bool* (*<*  $\vdash$  *\_*, *\_*, *\_* : *\_*) [150,0,0,0,150] 149)

**where**

*a\_Unit*:  $\Gamma \vdash \textit{Unit}, \textit{Unit}, \textit{Unit} : \textit{One} \mid$   
*a\_Var*:  $\Gamma \text{ \textit{\$} \$} x = \textit{Some } \tau$   
 $\Rightarrow \Gamma \vdash \textit{Var } x, \textit{Var } x, \textit{Var } x : \tau \mid$   
*a\_Lam*:  $\ll \textit{atom } x \# \Gamma; \Gamma(x \text{ \textit{\$} \$} = \tau_1) \vdash e, eP, eV : \tau_2 \gg$   
 $\Rightarrow \Gamma \vdash \textit{Lam } x e, \textit{Lam } x eP, \textit{Lam } x eV : \textit{Fun } \tau_1 \tau_2 \mid$   
*a\_App*:  $\ll \Gamma \vdash e_1, eP_1, eV_1 : \textit{Fun } \tau_1 \tau_2; \Gamma \vdash e_2, eP_2, eV_2 : \tau_1 \gg$   
 $\Rightarrow \Gamma \vdash \textit{App } e_1 e_2, \textit{App } eP_1 eP_2, \textit{App } eV_1 eV_2 : \tau_2 \mid$   
*a\_Let*:  $\ll \textit{atom } x \# (\Gamma, e_1, eP_1, eV_1); \Gamma \vdash e_1, eP_1, eV_1 : \tau_1; \Gamma(x \text{ \textit{\$} \$} = \tau_1) \vdash e_2, eP_2, eV_2 : \tau_2 \gg$   
 $\Rightarrow \Gamma \vdash \textit{Let } e_1 x e_2, \textit{Let } eP_1 x eP_2, \textit{Let } eV_1 x eV_2 : \tau_2 \mid$   
*a\_Rec*:  $\ll \textit{atom } x \# \Gamma; \textit{atom } y \# (\Gamma, x); \Gamma(x \text{ \textit{\$} \$} = \textit{Fun } \tau_1 \tau_2) \vdash \textit{Lam } y e, \textit{Lam } y eP, \textit{Lam } y eV : \textit{Fun } \tau_1$   
 $\tau_2 \gg$   
 $\Rightarrow \Gamma \vdash \textit{Rec } x (\textit{Lam } y e), \textit{Rec } x (\textit{Lam } y eP), \textit{Rec } x (\textit{Lam } y eV) : \textit{Fun } \tau_1 \tau_2 \mid$   
*a\_Inj1*:  $\ll \Gamma \vdash e, eP, eV : \tau_1 \gg$   
 $\Rightarrow \Gamma \vdash \textit{Inj1 } e, \textit{Inj1 } eP, \textit{Inj1 } eV : \textit{Sum } \tau_1 \tau_2 \mid$   
*a\_Inj2*:  $\ll \Gamma \vdash e, eP, eV : \tau_2 \gg$   
 $\Rightarrow \Gamma \vdash \textit{Inj2 } e, \textit{Inj2 } eP, \textit{Inj2 } eV : \textit{Sum } \tau_1 \tau_2 \mid$   
*a\_Case*:  $\ll \Gamma \vdash e, eP, eV : \textit{Sum } \tau_1 \tau_2; \Gamma \vdash e_1, eP_1, eV_1 : \textit{Fun } \tau_1 \tau; \Gamma \vdash e_2, eP_2, eV_2 : \textit{Fun } \tau_2 \tau \gg$   
 $\Rightarrow \Gamma \vdash \textit{Case } e e_1 e_2, \textit{Case } eP eP_1 eP_2, \textit{Case } eV eV_1 eV_2 : \tau \mid$   
*a\_Pair*:  $\ll \Gamma \vdash e_1, eP_1, eV_1 : \tau_1; \Gamma \vdash e_2, eP_2, eV_2 : \tau_2 \gg$   
 $\Rightarrow \Gamma \vdash \textit{Pair } e_1 e_2, \textit{Pair } eP_1 eP_2, \textit{Pair } eV_1 eV_2 : \textit{Prod } \tau_1 \tau_2 \mid$   
*a\_Prj1*:  $\ll \Gamma \vdash e, eP, eV : \textit{Prod } \tau_1 \tau_2 \gg$   
 $\Rightarrow \Gamma \vdash \textit{Prj1 } e, \textit{Prj1 } eP, \textit{Prj1 } eV : \tau_1 \mid$   
*a\_Prj2*:  $\ll \Gamma \vdash e, eP, eV : \textit{Prod } \tau_1 \tau_2 \gg$   
 $\Rightarrow \Gamma \vdash \textit{Prj2 } e, \textit{Prj2 } eP, \textit{Prj2 } eV : \tau_2 \mid$   
*a\_Roll*:  $\ll \textit{atom } \alpha \# \Gamma; \Gamma \vdash e, eP, eV : \textit{subst\_type } \tau (\textit{Mu } \alpha \tau) \alpha \gg$   
 $\Rightarrow \Gamma \vdash \textit{Roll } e, \textit{Roll } eP, \textit{Roll } eV : \textit{Mu } \alpha \tau \mid$   
*a\_Unroll*:  $\ll \textit{atom } \alpha \# \Gamma; \Gamma \vdash e, eP, eV : \textit{Mu } \alpha \tau \gg$   
 $\Rightarrow \Gamma \vdash \textit{Unroll } e, \textit{Unroll } eP, \textit{Unroll } eV : \textit{subst\_type } \tau (\textit{Mu } \alpha \tau) \alpha \mid$   
*a\_Auth*:  $\ll \Gamma \vdash e, eP, eV : \tau \gg$   
 $\Rightarrow \Gamma \vdash \textit{Auth } e, \textit{Auth } eP, \textit{Auth } eV : \textit{AuthT } \tau \mid$   
*a\_Unauth*:  $\ll \Gamma \vdash e, eP, eV : \textit{AuthT } \tau \gg$

$\implies \Gamma \vdash \text{Unauth } e, \text{Unauth } eP, \text{Unauth } eV : \tau \mid$   
 $a\_HashI: \llbracket \{\$\$ \} \vdash v, vP, (vP) : \tau; \text{hash } (vP) = h; \text{value } v; \text{value } vP \rrbracket$   
 $\implies \Gamma \vdash v, \text{Hashed } h \ vP, \text{Hash } h : \text{AuthT } \tau$

**declare** *agree.intros*[*intro*]

**equivariance** *agree*

**nominal\_inductive** *agree*

**avoids** *a\_Lam*: *x*  
 $\mid$  *a\_Rec*: *x* **and** *y*  
 $\mid$  *a\_Let*: *x*  
 $\mid$  *a\_Roll*:  $\alpha$   
 $\mid$  *a\_Unroll*:  $\alpha$   
*<proof>*

**lemma** *Abs\_lst\_eq\_3tuple*:

**fixes** *x x' :: var*  
**fixes** *e eP eV e' eP' eV' :: term*  
**assumes**  $\llbracket \text{atom } x \rrbracket \text{lst. } e = \llbracket \text{atom } x' \rrbracket \text{lst. } e'$   
**and**  $\llbracket \text{atom } x \rrbracket \text{lst. } eP = \llbracket \text{atom } x' \rrbracket \text{lst. } eP'$   
**and**  $\llbracket \text{atom } x \rrbracket \text{lst. } eV = \llbracket \text{atom } x' \rrbracket \text{lst. } eV'$   
**shows**  $\llbracket \text{atom } x \rrbracket \text{lst. } (e, eP, eV) = \llbracket \text{atom } x' \rrbracket \text{lst. } (e', eP', eV')$   
*<proof>*

**lemma** *agree\_fresh\_env\_fresh\_term*:

**fixes** *a :: var*  
**assumes**  $\Gamma \vdash e, eP, eV : \tau$  *atom a*  $\# \Gamma$   
**shows** *atom a*  $\# (e, eP, eV)$   
*<proof>*

**lemma** *agree\_empty\_fresh*[*dest*]:

**fixes** *a :: var*  
**assumes**  $\{\$\$ \} \vdash e, eP, eV : \tau$   
**shows**  $\{\text{atom } a\} \#* \{e, eP, eV\}$   
*<proof>*

Inversion rules for agreement.

**declare**  $\llbracket \text{simproc del: } \alpha \_ \text{lst} \rrbracket$

**lemma** *a\_Lam\_inv\_I*[*elim*]:

**assumes**  $\Gamma \vdash (\text{Lam } x \ e'), eP, eV : (\text{Fun } \tau_1 \ \tau_2)$   
**and** *atom x*  $\# \Gamma$   
**obtains** *eP' eV'* **where**  $eP = \text{Lam } x \ eP' \ eV = \text{Lam } x \ eV' \ \Gamma(x \ \$\$ := \tau_1) \vdash e', eP', eV' : \tau_2$   
*<proof>*

**lemma** *a\_Lam\_inv\_P*[*elim*]:

**assumes**  $\{\$\$ \} \vdash v, (\text{Lam } x \ vP'), vV : (\text{Fun } \tau_1 \ \tau_2)$   
**obtains** *v' vV'* **where**  $v = \text{Lam } x \ v' \ vV = \text{Lam } x \ vV' \ \{\$\$ \}(x \ \$\$ := \tau_1) \vdash v', vP', vV' : \tau_2$   
*<proof>*

**lemma** *a\_Lam\_inv\_V*[*elim*]:

**assumes**  $\{\$\$ \} \vdash v, vP, (\text{Lam } x \ vV') : (\text{Fun } \tau_1 \ \tau_2)$   
**obtains** *v' vP'* **where**  $v = \text{Lam } x \ v' \ vP = \text{Lam } x \ vP' \ \{\$\$ \}(x \ \$\$ := \tau_1) \vdash v', vP', vV' : \tau_2$   
*<proof>*

**lemma** *a\_Rec\_inv\_I*[*elim*]:

**assumes**  $\Gamma \vdash \text{Rec } x \ e, eP, eV : \text{Fun } \tau_1 \ \tau_2$   
**and** *atom x*  $\# \Gamma$

**obtains**  $y e' eP' eV'$   
**where**  $e = Lam\ y\ e' eP = Rec\ x\ (Lam\ y\ eP')\ eV = Rec\ x\ (Lam\ y\ eV')\ atom\ y\ \# (\Gamma, x)$   
 $\Gamma(x\ \$\$ := Fun\ \tau_1\ \tau_2) \vdash Lam\ y\ e', Lam\ y\ eP', Lam\ y\ eV' : Fun\ \tau_1\ \tau_2$   
 $\langle proof \rangle$

**lemma**  $a\_Rec\_inv\_P[elim]$ :  
**assumes**  $\Gamma \vdash e, Rec\ x\ eP, eV : Fun\ \tau_1\ \tau_2$   
**and**  $atom\ x\ \# \Gamma$   
**obtains**  $y e' eP' eV'$   
**where**  $e = Rec\ x\ (Lam\ y\ e')\ eP = Lam\ y\ eP'\ eV = Rec\ x\ (Lam\ y\ eV')\ atom\ y\ \# (\Gamma, x)$   
 $\Gamma(x\ \$\$ := Fun\ \tau_1\ \tau_2) \vdash Lam\ y\ e', Lam\ y\ eP', Lam\ y\ eV' : Fun\ \tau_1\ \tau_2$   
 $\langle proof \rangle$

**lemma**  $a\_Rec\_inv\_V[elim]$ :  
**assumes**  $\Gamma \vdash e, eP, Rec\ x\ eV : Fun\ \tau_1\ \tau_2$   
**and**  $atom\ x\ \# \Gamma$   
**obtains**  $y e' eP' eV'$   
**where**  $e = Rec\ x\ (Lam\ y\ e')\ eP = Rec\ x\ (Lam\ y\ eP')\ eV = Lam\ y\ eV'\ atom\ y\ \# (\Gamma, x)$   
 $\Gamma(x\ \$\$ := Fun\ \tau_1\ \tau_2) \vdash Lam\ y\ e', Lam\ y\ eP', Lam\ y\ eV' : Fun\ \tau_1\ \tau_2$   
 $\langle proof \rangle$

**inductive\_cases**  $a\_Inj1\_inv\_I[elim]$ :  $\Gamma \vdash Inj1\ e, eP, eV : Sum\ \tau_1\ \tau_2$   
**inductive\_cases**  $a\_Inj1\_inv\_P[elim]$ :  $\Gamma \vdash e, Inj1\ eP, eV : Sum\ \tau_1\ \tau_2$   
**inductive\_cases**  $a\_Inj1\_inv\_V[elim]$ :  $\Gamma \vdash e, eP, Inj1\ eV : Sum\ \tau_1\ \tau_2$

**inductive\_cases**  $a\_Inj2\_inv\_I[elim]$ :  $\Gamma \vdash Inj2\ e, eP, eV : Sum\ \tau_1\ \tau_2$   
**inductive\_cases**  $a\_Inj2\_inv\_P[elim]$ :  $\Gamma \vdash e, Inj2\ eP, eV : Sum\ \tau_1\ \tau_2$   
**inductive\_cases**  $a\_Inj2\_inv\_V[elim]$ :  $\Gamma \vdash e, eP, Inj2\ eV : Sum\ \tau_1\ \tau_2$

**inductive\_cases**  $a\_Pair\_inv\_I[elim]$ :  $\Gamma \vdash Pair\ e_1\ e_2, eP, eV : Prod\ \tau_1\ \tau_2$   
**inductive\_cases**  $a\_Pair\_inv\_P[elim]$ :  $\Gamma \vdash e, Pair\ eP_1\ eP_2, eV : Prod\ \tau_1\ \tau_2$

**lemma**  $a\_Roll\_inv\_I[elim]$ :  
**assumes**  $\Gamma \vdash Roll\ e', eP, eV : Mu\ \alpha\ \tau$   
**obtains**  $eP' eV'$   
**where**  $eP = Roll\ eP'\ eV = Roll\ eV'\ \Gamma \vdash e', eP', eV' : subst\_type\ \tau\ (Mu\ \alpha\ \tau)\ \alpha$   
 $\langle proof \rangle$

**lemma**  $a\_Roll\_inv\_P[elim]$ :  
**assumes**  $\Gamma \vdash e, Roll\ eP', eV : Mu\ \alpha\ \tau$   
**obtains**  $e' eV'$   
**where**  $e = Roll\ e'\ eV = Roll\ eV'\ \Gamma \vdash e', eP', eV' : subst\_type\ \tau\ (Mu\ \alpha\ \tau)\ \alpha$   
 $\langle proof \rangle$

**lemma**  $a\_Roll\_inv\_V[elim]$ :  
**assumes**  $\Gamma \vdash e, eP, Roll\ eV' : Mu\ \alpha\ \tau$   
**obtains**  $e' eP'$   
**where**  $e = Roll\ e'\ eP = Roll\ eP'\ \Gamma \vdash e', eP', eV' : subst\_type\ \tau\ (Mu\ \alpha\ \tau)\ \alpha$   
 $\langle proof \rangle$

**inductive\_cases**  $a\_HashI\_inv[elim]$ :  $\Gamma \vdash v, Hashed\ (hash\ (vP))\ vP, Hash\ (hash\ (vP)) : AuthT\ \tau$

Inversion on types for agreement.

**lemma**  $a\_AuthT\_value\_inv$ :  
**assumes**  $\{\$\$ \} \vdash v, vP, vV : AuthT\ \tau$   
**and**  $value\ v\ value\ vP\ value\ vV$   
**obtains**  $vP'$  **where**  $vP = Hashed\ (hash\ (vP'))\ vP'\ vV = Hash\ (hash\ (vP'))\ value\ vP'$   
 $\langle proof \rangle$

**inductive\_cases** *a\_Mu\_inv*[*elim*]:  $\Gamma \vdash e, eP, eV : Mu \ \alpha \ \tau$   
**inductive\_cases** *a\_Sum\_inv*[*elim*]:  $\Gamma \vdash e, eP, eV : Sum \ \tau_1 \ \tau_2$   
**inductive\_cases** *a\_Prod\_inv*[*elim*]:  $\Gamma \vdash e, eP, eV : Prod \ \tau_1 \ \tau_2$   
**inductive\_cases** *a\_Fun\_inv*[*elim*]:  $\Gamma \vdash e, eP, eV : Fun \ \tau_1 \ \tau_2$

**declare** [[*simproc add: alpha\_lst*]]

**lemma** *agree\_weakening\_1*:

**assumes**  $\Gamma \vdash e, eP, eV : \tau \ \text{atom } y \ \# \ e \ \text{atom } y \ \# \ eP \ \text{atom } y \ \# \ eV$   
**shows**  $\Gamma(y \ \$\$ := \tau') \vdash e, eP, eV : \tau$

*<proof>*

**lemma** *agree\_weakening\_2*:

**assumes**  $\Gamma \vdash e, eP, eV : \tau \ \text{atom } y \ \# \ \Gamma$   
**shows**  $\Gamma(y \ \$\$ := \tau') \vdash e, eP, eV : \tau$

*<proof>*

**lemma** *agree\_weakening\_env*:

**assumes**  $\{\$\$ \} \vdash e, eP, eV : \tau$   
**shows**  $\Gamma \vdash e, eP, eV : \tau$

*<proof>*

## 5 Formalization of Miller et al.'s [2] Main Results

**lemma** *judge\_imp\_agree*:

**assumes**  $\Gamma \vdash e : \tau$   
**shows**  $\Gamma \vdash e, e, e : \tau$

*<proof>*

### 5.1 Lemma 2.1

**lemma** *lemma2\_1*:

**assumes**  $\Gamma \vdash e, eP, eV : \tau$   
**shows**  $(eP) = eV$

*<proof>*

### 5.2 Counterexample to Lemma 2.2

**lemma** *lemma2\_2\_false*:

**fixes**  $x :: \text{var}$

**assumes**  $\bigwedge \Gamma \ e \ eP \ eV \ \tau \ eP' \ eV'. \llbracket \Gamma \vdash e, eP, eV : \tau; \Gamma \vdash e, eP', eV' : \tau \rrbracket \implies eP = eP' \wedge eV = eV'$   
**shows** *False*

*<proof>*

**lemma** *smallstep\_ideal\_deterministic*:

$\llbracket [], t \rrbracket I \rightarrow \llbracket [], u \rrbracket \implies \llbracket [], t \rrbracket I \rightarrow \llbracket [], u' \rrbracket \implies u = u'$

*<proof>*

**lemma** *smallsteps\_ideal\_deterministic*:

$\llbracket [], t \rrbracket I \rightarrow i \llbracket [], u \rrbracket \implies \llbracket [], t \rrbracket I \rightarrow i \llbracket [], u' \rrbracket \implies u = u'$

*<proof>*

### 5.3 Lemma 2.3

**lemma** *lemma2\_3*:

**assumes**  $\Gamma \vdash e, eP, eV : \tau$

**shows**  $erase\_env \Gamma \vdash_W e : erase \tau$   
 $\langle proof \rangle$

## 5.4 Lemma 2.4

**lemma**  $lemma2\_4[dest]$ :  
**assumes**  $\Gamma \vdash e, eP, eV : \tau$   
**shows**  $value e \wedge value eP \wedge value eV \vee \neg value e \wedge \neg value eP \wedge \neg value eV$   
 $\langle proof \rangle$

## 5.5 Lemma 3

**lemma**  $lemma3\_general$ :  
**fixes**  $\Gamma :: tyenv$  **and**  $vs vPs vVs :: tenu$   
**assumes**  $\Gamma \vdash e : \tau$   $A \sqsubseteq fmdom \Gamma$   
**and**  $fmdom vs = A$   $fmdom vPs = A$   $fmdom vVs = A$   
**and**  $\forall x. x \in A \longrightarrow (\exists \tau' v vP h.$   
 $\Gamma \ \$\$ x = Some (AuthT \tau') \wedge$   
 $vs \ \$\$ x = Some v \wedge$   
 $vPs \ \$\$ x = Some (Hashed h vP) \wedge$   
 $vVs \ \$\$ x = Some (Hash h) \wedge$   
 $\{\$\$\} \vdash v, Hashed h vP, Hash h : (AuthT \tau'))$   
**shows**  $fmdrop\_fset A \Gamma \vdash psubst\_term e vs, psubst\_term e vPs, psubst\_term e vVs : \tau$   
 $\langle proof \rangle$

**lemmas**  $lemma3 = lemma3\_general[\text{where } A = fmdom \Gamma \text{ and } \Gamma = \Gamma, \text{simplified}] \text{ for } \Gamma$

## 5.6 Lemma 4

**lemma**  $lemma4$ :  
**assumes**  $\Gamma(x \ \$\$ := \tau') \vdash e, eP, eV : \tau$   
**and**  $\{\$\$\} \vdash v, vP, vV : \tau'$   
**and**  $value v \ value vP \ value vV$   
**shows**  $\Gamma \vdash e[v / x], eP[vP / x], eV[vV / x] : \tau$   
 $\langle proof \rangle$

## 5.7 Lemma 5: Single-Step Correctness

**lemma**  $lemma5$ :  
**assumes**  $\{\$\$\} \vdash e, eP, eV : \tau$   
**and**  $\ll [], e \gg I \rightarrow \ll [], e' \gg$   
**obtains**  $eP' eV' \pi$   
**where**  $\{\$\$\} \vdash e', eP', eV' : \tau \ \forall \pi_P. \ll \pi_P, eP \gg P \rightarrow \ll \pi_P @ \pi, eP' \gg \forall \pi'. \ll \pi @ \pi', eV \gg V \rightarrow$   
 $\ll \pi', eV' \gg$   
 $\langle proof \rangle$

## 5.8 Lemma 6: Single-Step Security

**lemma**  $lemma6$ :  
**assumes**  $\{\$\$\} \vdash e, eP, eV : \tau$   
**and**  $\ll \pi_A, eV \gg V \rightarrow \ll \pi', eV' \gg$   
**obtains**  $e' eP' \pi$   
**where**  $\ll [], e \gg I \rightarrow \ll [], e' \gg \forall \pi_P. \ll \pi_P, eP \gg P \rightarrow \ll \pi_P @ \pi, eP' \gg$   
**and**  $\{\$\$\} \vdash e', eP', eV' : \tau \wedge \pi_A = \pi @ \pi' \vee$   
 $(\exists s s'. closed s \wedge closed s' \wedge \pi = [s] \wedge \pi_A = [s'] @ \pi' \wedge s \neq s' \wedge hash s = hash s')$   
 $\langle proof \rangle$

## 5.9 Theorem 1: Correctness

**lemma** *theorem1\_correctness*:  
**assumes**  $\{\$\$ \} \vdash e, eP, eV : \tau$   
**and**  $\ll \[], e \gg I \rightarrow i \ll \[], e' \gg$   
**obtains**  $eP' eV' \pi$   
**where**  $\ll \[], eP \gg P \rightarrow i \ll \pi, eP' \gg$   
 $\ll \pi, eV \gg V \rightarrow i \ll \[], eV' \gg$   
 $\{\$\$ \} \vdash e', eP', eV' : \tau$   
*<proof>*

## 5.10 Counterexamples to Theorem 1: Security

Counterexample using administrative normal form.

**lemma** *security\_false*:  
**assumes** *agree*:  $\bigwedge e eP eV \tau \pi A i \pi' eV'. [\{\$\$ \} \vdash e, eP, eV : \tau; \ll \pi A, eV \gg V \rightarrow i \ll \pi', eV' \gg]$   
 $\implies$   
 $\exists e' eP' \pi j \pi_0 s s'. (\ll \[], e \gg I \rightarrow i \ll \[], e' \gg \wedge \ll \[], eP \gg P \rightarrow i \ll \pi, eP' \gg \wedge (\pi A = \pi @ \pi')$   
 $\wedge \{\$\$ \} \vdash e', eP', eV' : \tau) \vee$   
 $(j \leq i \wedge \ll \[], eP \gg P \rightarrow j \ll \pi_0 @ [s], eP' \gg \wedge (\pi A = \pi_0 @ [s'] @ \pi') \wedge s \neq s' \wedge \text{hash } s = \text{hash } s')$   
**and** *collision*:  $\text{hash } (\text{Inj1 } \text{Unit}) = \text{hash } (\text{Inj2 } \text{Unit})$   
**and** *no\_collision\_with\_Unit*:  $\bigwedge t. \text{hash } \text{Unit} = \text{hash } t \implies t = \text{Unit}$   
**shows** *False*  
*<proof>*

Alternative, shorter counterexample not in administrative normal form.

**lemma** *security\_false\_alt*:  
**assumes** *agree*:  $\bigwedge e eP eV \tau \pi A i \pi' eV'. [\{\$\$ \} \vdash e, eP, eV : \tau; \ll \pi A, eV \gg V \rightarrow i \ll \pi', eV' \gg]$   
 $\implies$   
 $\exists e' eP' \pi j \pi_0 s s'. (\ll \[], e \gg I \rightarrow i \ll \[], e' \gg \wedge \ll \[], eP \gg P \rightarrow i \ll \pi, eP' \gg \wedge (\pi A = \pi @ \pi')$   
 $\wedge \{\$\$ \} \vdash e', eP', eV' : \tau) \vee$   
 $(j \leq i \wedge \ll \[], eP \gg P \rightarrow j \ll \pi_0 @ [s], eP' \gg \wedge (\pi A = \pi_0 @ [s'] @ \pi') \wedge s \neq s' \wedge \text{hash } s = \text{hash } s')$   
**and** *collision*:  $\text{hash } (\text{Inj1 } \text{Unit}) = \text{hash } (\text{Inj2 } \text{Unit})$   
**and** *no\_collision\_with\_Unit*:  $\bigwedge t. \text{hash } \text{Unit} = \text{hash } t \implies t = \text{Unit}$   
**shows** *False*  
*<proof>*

## 5.11 Corrected Theorem 1: Security

**lemma** *theorem1\_security*:  
**assumes**  $\{\$\$ \} \vdash e, eP, eV : \tau$   
**and**  $\ll \pi A, eV \gg V \rightarrow i \ll \pi', eV' \gg$   
**shows**  $(\exists e' eP' \pi. \ll \[], e \gg I \rightarrow i \ll \[], e' \gg \wedge \ll \[], eP \gg P \rightarrow i \ll \pi, eP' \gg \wedge \pi A = \pi @ \pi' \wedge \{\$\$ \} \vdash e', eP', eV' : \tau) \vee$   
 $(\exists eP' j \pi_0 \pi_0' s s'. j \leq i \wedge \ll \[], eP \gg P \rightarrow j \ll \pi_0 @ [s], eP' \gg \wedge \pi A = \pi_0 @ [s'] @ \pi_0' @ \pi' \wedge s \neq s' \wedge \text{hash } s = \text{hash } s' \wedge \text{closed } s \wedge \text{closed } s')$   
*<proof>*

## 5.12 Remark 1

**lemma** *remark1\_single*:  
**assumes**  $\{\$\$ \} \vdash e, eP, eV : \tau$   
**and**  $\ll \pi P, eP \gg P \rightarrow \ll \pi P @ \pi, eP' \gg$   
**obtains**  $e' eV'$  **where**  $\{\$\$ \} \vdash e', eP', eV' : \tau \wedge \ll \[], e \gg I \rightarrow \ll \[], e' \gg \wedge \ll \pi, eV \gg V \rightarrow \ll \[], eV' \gg$   
*<proof>*

**lemma** *remark1*:

**assumes**  $\{\$\$ \} \vdash e, eP, eV : \tau$

**and**  $\ll \pi_P, eP \gg P \rightarrow i \ll \pi_P @ \pi, eP' \gg$

**obtains**  $e' eV'$

**where**  $\{\$\$ \} \vdash e', eP', eV' : \tau \ll [], e \gg I \rightarrow i \ll [], e' \gg \ll \pi, eV \gg V \rightarrow i \ll [], eV' \gg$

*<proof>*

## References

- [1] M. Brun and D. Traytel. Generic authenticated data structures, formally. In J. Harrison, J. O’Leary, and A. Tolmach, editors, *ITP 2019*, volume 141 of *LIPICs*, pages 10:1–10:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.
- [2] A. Miller, M. Hicks, J. Katz, and E. Shi. Authenticated data structures, generically. In S. Jagannathan and P. Sewell, editors, *POPL 2014*, pages 411–424. ACM, 2014.