

# MiniSail

Mark P. Wassell

April 11, 2026

## Abstract

MiniSail is a kernel language for Sail [1], an instruction set architecture (ISA) specification language. Sail is an imperative language with a light-weight dependent type system similar to refinement type systems such as [2]. From an ISA specification, the Sail compiler can generate theorem prover code and C (or OCaml) to give an executable emulator for an architecture. The idea behind MiniSail is to capture the key and novel features of Sail in terms of their syntax, typing rules and operational semantics, and to confirm that they work together by proving progress and preservation lemmas. We use the Nominal2 library to handle binding.

# Contents

<b>1</b>	<b>Prelude</b>	<b>5</b>
1.1	Lemmas helping with equivariance proofs . . . . .	5
1.2	Freshness via equivariance . . . . .	6
1.3	Additional simplification rules . . . . .	6
1.4	Additional equivariance lemmas . . . . .	6
1.5	Freshness lemmas . . . . .	8
1.6	Freshness and support for subsets of variables . . . . .	9
1.7	The set of free variables of an expression . . . . .	9
1.8	Other useful lemmas . . . . .	10
<b>2</b>	<b>Syntax</b>	<b>15</b>
2.1	Program Syntax . . . . .	15
2.1.1	AST Datatypes . . . . .	15
2.1.2	Lemmas . . . . .	17
2.2	Context Syntax . . . . .	23
2.2.1	Datatypes . . . . .	23
2.2.2	Functions and Lemmas . . . . .	23
2.2.3	Immutable Variable Context Lemmas . . . . .	25
2.2.4	Mutable Variable Context Lemmas . . . . .	28
2.2.5	Lookup Functions . . . . .	29
2.3	Functions for bit list/vectors . . . . .	30
<b>3</b>	<b>Immutable Variable Substitution</b>	<b>31</b>
3.1	Class . . . . .	31
3.2	Values . . . . .	32
3.3	Expressions . . . . .	33
3.4	Expressions in Constraints . . . . .	34
3.5	Constraints . . . . .	35
3.6	Variable Context . . . . .	37
3.7	Types . . . . .	38
3.8	Mutable Variable Context . . . . .	41
3.9	Statements . . . . .	42
3.10	Type Definition . . . . .	44
3.11	Variable Context . . . . .	45
3.12	Lookup . . . . .	45

<b>4</b>	<b>Basic Type Variable Substitution</b>	<b>46</b>
4.1	Class . . . . .	46
4.2	Base Type . . . . .	47
4.3	Value . . . . .	48
4.4	Constraints Expressions . . . . .	48
4.5	Constraints . . . . .	49
4.6	Types . . . . .	49
4.7	Expressions . . . . .	50
4.8	Statements . . . . .	50
4.9	Function Type . . . . .	52
4.10	Contexts . . . . .	53
	4.10.1 Immutable Variables . . . . .	53
	4.10.2 Mutable Variables . . . . .	53
<b>5</b>	<b>Wellformed Terms</b>	<b>56</b>
5.1	Definitions . . . . .	56
<b>6</b>	<b>Refinement Constraint Logic</b>	<b>67</b>
6.1	Evaluation and Satisfiability . . . . .	67
	6.1.1 Valuation . . . . .	67
	6.1.2 Evaluation base-types . . . . .	69
	6.1.3 Wellformed vvaluations . . . . .	69
	6.1.4 Evaluating Terms . . . . .	69
	6.1.5 Satisfiability . . . . .	70
6.2	Validity . . . . .	70
6.3	Lemmas . . . . .	71
<b>7</b>	<b>Syntax Lemmas</b>	<b>72</b>
7.1	Support, lookup and contexts . . . . .	72
7.2	Type Definitions . . . . .	75
7.3	Function Definitions . . . . .	75
<b>8</b>	<b>Wellformedness Lemmas</b>	<b>77</b>
8.1	Prelude . . . . .	77
8.2	Strong Elimination . . . . .	77
8.3	Context Extension . . . . .	77
8.4	Context . . . . .	78
8.5	Converting between wb forms . . . . .	79
8.6	Support . . . . .	81
8.7	Freshness . . . . .	84
8.8	Misc . . . . .	86
8.9	Context Strengthening . . . . .	86
8.10	Type Definitions . . . . .	87
	8.10.1 Simple . . . . .	88
	8.10.2 Polymorphic . . . . .	89
8.11	Equivariance Lemmas . . . . .	91
8.12	Lookup . . . . .	91

8.13	Function Definitions	93
8.14	Weakening	97
8.15	Useful well-formedness instances	100
8.16	Replacing the constraint on a variable in a context	101
8.17	Preservation of well-formedness under substitution	103
<b>9</b>	<b>Type System</b>	<b>108</b>
9.1	Subtyping	108
9.2	Literals	108
9.3	Values	109
9.4	Expressions	110
9.5	Statements	113
9.6	Programs	115
<b>10</b>	<b>Operational Semantics</b>	<b>118</b>
10.1	Reduction Rules	118
10.2	Reduction Typing	120
<b>11</b>	<b>Refinement Constraint Logic Lemmas</b>	<b>123</b>
11.1	Lemmas	123
11.2	Existence of evaluation	123
11.3	Satisfiability	125
11.4	Substitution for Evaluation	126
11.5	Validity	127
11.5.1	Weakening and Strengthening	128
11.5.2	Updating valuation	129
11.6	Base Type Substitution	130
11.7	Expression Operator Lemmas	134
<b>12</b>	<b>Typing Lemmas</b>	<b>138</b>
12.1	Prelude	138
12.2	Subtyping	139
12.3	Literals	144
12.4	Values	145
12.5	Expressions	149
12.6	Statements	150
12.7	Additional Elimination and Intros	151
12.7.1	Values	151
12.7.2	Expressions	152
12.8	Weakening	152
12.8.1	Weakening Immutable Variable Context	153
<b>13</b>	<b>Context Subtyping Lemmas</b>	<b>155</b>
13.1	Replace or exchange type of variable in a context	155
13.2	Validity and Subtyping	158
13.3	Literals	159
13.4	Values	159

13.5 Expressions . . . . .	160
13.6 Statements . . . . .	160
<b>14 Immutable Variable Substitution Lemmas</b>	<b>162</b>
14.1 Proof Methods . . . . .	162
14.2 Prelude . . . . .	162
14.3 Context . . . . .	162
14.4 Validity . . . . .	163
14.5 Subtyping . . . . .	163
14.6 Values . . . . .	164
14.7 Expressions . . . . .	166
14.8 Statements . . . . .	166
<b>15 Basic Type Variable Substitution Lemmas</b>	<b>168</b>
<b>16 Safety</b>	<b>171</b>
16.1 Store Lemmas . . . . .	171
16.2 Preservation . . . . .	172
16.2.1 Function Application . . . . .	172
16.2.2 Operators . . . . .	174
16.2.3 Let Statements . . . . .	174
16.2.4 Other Statements . . . . .	175
16.2.5 Main Lemma . . . . .	177
16.3 Progress . . . . .	177
16.4 Safety . . . . .	178

# Chapter 1

## Prelude

Some useful Nominal lemmas. Many of these are from Launchbury.Nominal-Utils.

### 1.1 Lemmas helping with equivariance proofs

**lemma** *perm-rel-lemma*:

**assumes**  $\bigwedge \pi x y. r (\pi \cdot x) (\pi \cdot y) \implies r x y$   
**shows**  $r (\pi \cdot x) (\pi \cdot y) \longleftrightarrow r x y$  (**is** ?l  $\longleftrightarrow$  ?r)

*<proof>*

**lemma** *perm-rel-lemma2*:

**assumes**  $\bigwedge \pi x y. r x y \implies r (\pi \cdot x) (\pi \cdot y)$   
**shows**  $r x y \longleftrightarrow r (\pi \cdot x) (\pi \cdot y)$  (**is** ?l  $\longleftrightarrow$  ?r)

*<proof>*

**lemma** *fun-eqvtI*:

**assumes** *f-eqvt[eqvt]*:  $(\bigwedge p x. p \cdot (f x) = f (p \cdot x))$   
**shows**  $p \cdot f = f$  *<proof>*

**lemma** *eqvt-at-apply*:

**assumes** *eqvt-at f x*  
**shows**  $(p \cdot f) x = f x$

*<proof>*

**lemma** *eqvt-at-apply'*:

**assumes** *eqvt-at f x*  
**shows**  $p \cdot f x = f (p \cdot x)$

*<proof>*

**lemma** *eqvt-at-apply''*:

**assumes** *eqvt-at f x*  
**shows**  $(p \cdot f) (p \cdot x) = f (p \cdot x)$

*<proof>*

**lemma** *size-list-eqvt[eqvt]*:  $p \cdot \text{size-list } f x = \text{size-list } (p \cdot f) (p \cdot x)$

*<proof>*

## 1.2 Freshness via equivariance

**lemma** *eqvt-fresh-cong1*:  $(\bigwedge p x. p \cdot (f x) = f (p \cdot x)) \implies a \# x \implies a \# f x$   
*<proof>*

**lemma** *eqvt-fresh-cong2*:  
 **assumes** *eqvt*:  $(\bigwedge p x y. p \cdot (f x y) = f (p \cdot x) (p \cdot y))$   
 **and** *fresh1*:  $a \# x$  **and** *fresh2*:  $a \# y$   
 **shows**  $a \# f x y$   
*<proof>*

**lemma** *eqvt-fresh-star-cong1*:  
 **assumes** *eqvt*:  $(\bigwedge p x. p \cdot (f x) = f (p \cdot x))$   
 **and** *fresh1*:  $a \#^* x$   
 **shows**  $a \#^* f x$   
*<proof>*

**lemma** *eqvt-fresh-star-cong2*:  
 **assumes** *eqvt*:  $(\bigwedge p x y. p \cdot (f x y) = f (p \cdot x) (p \cdot y))$   
 **and** *fresh1*:  $a \#^* x$  **and** *fresh2*:  $a \#^* y$   
 **shows**  $a \#^* f x y$   
*<proof>*

**lemma** *eqvt-fresh-cong3*:  
 **assumes** *eqvt*:  $(\bigwedge p x y z. p \cdot (f x y z) = f (p \cdot x) (p \cdot y) (p \cdot z))$   
 **and** *fresh1*:  $a \# x$  **and** *fresh2*:  $a \# y$  **and** *fresh3*:  $a \# z$   
 **shows**  $a \# f x y z$   
*<proof>*

**lemma** *eqvt-fresh-star-cong3*:  
 **assumes** *eqvt*:  $(\bigwedge p x y z. p \cdot (f x y z) = f (p \cdot x) (p \cdot y) (p \cdot z))$   
 **and** *fresh1*:  $a \#^* x$  **and** *fresh2*:  $a \#^* y$  **and** *fresh3*:  $a \#^* z$   
 **shows**  $a \#^* f x y z$   
*<proof>*

## 1.3 Additional simplification rules

**lemma** *not-self-fresh[simp]*:  $atom x \# x \longleftrightarrow False$   
*<proof>*

**lemma** *fresh-star-singleton*:  $\{ x \} \#^* e \longleftrightarrow x \# e$   
*<proof>*

## 1.4 Additional equivariance lemmas

**lemma** *eqvt-cases*:  
 **fixes**  $f x \pi$   
 **assumes** *eqvt*:  $\bigwedge x. \pi \cdot f x = f (\pi \cdot x)$   
 **obtains**  $f x f (\pi \cdot x) \mid \neg f x \neg f (\pi \cdot x)$   
*<proof>*

**lemma** *range-eqvt*:  $\pi \cdot range Y = range (\pi \cdot Y)$

$\langle \text{proof} \rangle$

**lemma** *case-option-eqv*[*eqvt*]:

$$\pi \cdot \text{case-option } d \ f \ x = \text{case-option } (\pi \cdot d) \ (\pi \cdot f) \ (\pi \cdot x)$$

$\langle \text{proof} \rangle$

**lemma** *supp-option-eqv*:

$$\text{supp } (\text{case-option } d \ f \ x) \subseteq \text{supp } d \cup \text{supp } f \cup \text{supp } x$$

$\langle \text{proof} \rangle$

**lemma** *funpow-eqv*[*simp*,*eqvt*]:

$$\pi \cdot ((f :: 'a \Rightarrow 'a::pt) \ \overset{\sim}{\sim} \ n) = (\pi \cdot f) \ \overset{\sim}{\sim} \ (\pi \cdot n)$$

$\langle \text{proof} \rangle$

**lemma** *delete-eqv*[*eqvt*]:

$$\pi \cdot \text{AList.delete } x \ \Gamma = \text{AList.delete } (\pi \cdot x) \ (\pi \cdot \Gamma)$$

$\langle \text{proof} \rangle$

**lemma** *restrict-eqv*[*eqvt*]:

$$\pi \cdot \text{AList.restrict } S \ \Gamma = \text{AList.restrict } (\pi \cdot S) \ (\pi \cdot \Gamma)$$

$\langle \text{proof} \rangle$

**lemma** *supp-restrict*:

$$\text{supp } (\text{AList.restrict } S \ \Gamma) \subseteq \text{supp } \Gamma$$

$\langle \text{proof} \rangle$

**lemma** *clearjunk-eqv*[*eqvt*]:

$$\pi \cdot \text{AList.clearjunk } \Gamma = \text{AList.clearjunk } (\pi \cdot \Gamma)$$

$\langle \text{proof} \rangle$

**lemma** *map-ran-eqv*[*eqvt*]:

$$\pi \cdot \text{map-ran } f \ \Gamma = \text{map-ran } (\pi \cdot f) \ (\pi \cdot \Gamma)$$

$\langle \text{proof} \rangle$

**lemma** *dom-perm*:

$$\text{dom } (\pi \cdot f) = \pi \cdot (\text{dom } f)$$

$\langle \text{proof} \rangle$

**lemmas** *dom-perm-rev*[*simp*,*eqvt*] = *dom-perm*[*symmetric*]

**lemma** *ran-perm*[*simp*]:

$$\pi \cdot (\text{ran } f) = \text{ran } (\pi \cdot f)$$

$\langle \text{proof} \rangle$

**lemma** *map-add-eqv*[*eqvt*]:

$$\pi \cdot (m1 \ ++ \ m2) = (\pi \cdot m1) \ ++ \ (\pi \cdot m2)$$

$\langle \text{proof} \rangle$

**lemma** *map-of-eqv*[*eqvt*]:

$$\pi \cdot \text{map-of } l = \text{map-of } (\pi \cdot l)$$

$\langle \text{proof} \rangle$

**lemma** *concat-eqvt*[*eqvt*]:  $\pi \cdot \text{concat } l = \text{concat } (\pi \cdot l)$   
 ⟨*proof*⟩

**lemma** *tranclp-eqvt*[*eqvt*]:  $\pi \cdot \text{tranclp } P v_1 v_2 = \text{tranclp } (\pi \cdot P) (\pi \cdot v_1) (\pi \cdot v_2)$   
 ⟨*proof*⟩

**lemma** *rtranclp-eqvt*[*eqvt*]:  $\pi \cdot \text{rtranclp } P v_1 v_2 = \text{rtranclp } (\pi \cdot P) (\pi \cdot v_1) (\pi \cdot v_2)$   
 ⟨*proof*⟩

**lemma** *Set-filter-eqvt*[*eqvt*]:  $\pi \cdot \text{Set.filter } P S = \text{Set.filter } (\pi \cdot P) (\pi \cdot S)$   
 ⟨*proof*⟩

**lemma** *Sigma-eqvt'*[*eqvt*]:  $\pi \cdot \text{Sigma} = \text{Sigma}$   
 ⟨*proof*⟩

**lemma** *override-on-eqvt*[*eqvt*]:  
 $\pi \cdot (\text{override-on } m1 m2 S) = \text{override-on } (\pi \cdot m1) (\pi \cdot m2) (\pi \cdot S)$   
 ⟨*proof*⟩

**lemma** *card-eqvt*[*eqvt*]:  
 $\pi \cdot (\text{card } S) = \text{card } (\pi \cdot S)$   
 ⟨*proof*⟩

**lemma** *Projl-permute*:  
**assumes**  $a: \exists y. f = \text{Inl } y$   
**shows**  $(p \cdot (\text{Sum-Type.proj1 } f)) = \text{Sum-Type.proj1 } (p \cdot f)$   
 ⟨*proof*⟩

**lemma** *Projr-permute*:  
**assumes**  $a: \exists y. f = \text{Inr } y$   
**shows**  $(p \cdot (\text{Sum-Type.proj2 } f)) = \text{Sum-Type.proj2 } (p \cdot f)$   
 ⟨*proof*⟩

## 1.5 Freshness lemmas

**lemma** *fresh-list-elem*:  
**assumes**  $a \# \Gamma$   
**and**  $e \in \text{set } \Gamma$   
**shows**  $a \# e$   
 ⟨*proof*⟩

**lemma** *set-not-fresh*:  
 $x \in \text{set } L \implies \neg(\text{atom } x \# L)$   
 ⟨*proof*⟩

**lemma** *pure-fresh-star*[*simp*]:  $a \#* (x :: 'a :: \text{pure})$   
 ⟨*proof*⟩

**lemma** *supp-set-mem*:  $x \in \text{set } L \implies \text{supp } x \subseteq \text{supp } L$   
 ⟨*proof*⟩

**lemma** *set-supp-mono*:  $set\ L \subseteq set\ L2 \implies supp\ L \subseteq supp\ L2$   
 ⟨proof⟩

**lemma** *fresh-star-at-base*:  
**fixes**  $x :: 'a :: at-base$   
**shows**  $S \#* x \longleftrightarrow atom\ x \notin S$   
 ⟨proof⟩

## 1.6 Freshness and support for subsets of variables

**lemma** *supp-mono*:  $finite\ (B :: 'a :: fs\ set) \implies A \subseteq B \implies supp\ A \subseteq supp\ B$   
 ⟨proof⟩

**lemma** *fresh-subset*:  
 $finite\ B \implies x \# (B :: 'a :: at-base\ set) \implies A \subseteq B \implies x \# A$   
 ⟨proof⟩

**lemma** *fresh-star-subset*:  
 $finite\ B \implies x \#* (B :: 'a :: at-base\ set) \implies A \subseteq B \implies x \#* A$   
 ⟨proof⟩

**lemma** *fresh-star-set-subset*:  
 $x \#* (B :: 'a :: at-base\ list) \implies set\ A \subseteq set\ B \implies x \#* A$   
 ⟨proof⟩

## 1.7 The set of free variables of an expression

**definition**  $fv :: 'a :: pt \Rightarrow 'b :: at-base\ set$   
**where**  $fv\ e = \{v. atom\ v \in supp\ e\}$

**lemma** *fv-eqvt[simp,eqvt]*:  $\pi \cdot (fv\ e) = fv\ (\pi \cdot e)$   
 ⟨proof⟩

**lemma** *fv-Nil[simp]*:  $fv\ [] = \{\}$   
 ⟨proof⟩

**lemma** *fv-Cons[simp]*:  $fv\ (x \# xs) = fv\ x \cup fv\ xs$   
 ⟨proof⟩

**lemma** *fv-Pair[simp]*:  $fv\ (x, y) = fv\ x \cup fv\ y$   
 ⟨proof⟩

**lemma** *fv-append[simp]*:  $fv\ (x @ y) = fv\ x \cup fv\ y$   
 ⟨proof⟩

**lemma** *fv-at-base[simp]*:  $fv\ a = \{a :: 'a :: at-base\}$   
 ⟨proof⟩

**lemma** *fv-pure[simp]*:  $fv\ (a :: 'a :: pure) = \{\}$   
 ⟨proof⟩

**lemma** *fv-set-at-base[simp]*:  $fv\ (l :: ('a :: at-base)\ list) = set\ l$   
 ⟨proof⟩

**lemma** *flip-not-fv*:  $a \notin fv\ x \implies b \notin fv\ x \implies (a \leftrightarrow b) \cdot x = x$

*<proof>*

**lemma** *fv-not-fresh*:  $atom\ x \# e \longleftrightarrow x \notin fv\ e$   
*<proof>*

**lemma** *fresh-fv*:  $finite\ (fv\ e :: 'a\ set) \implies atom\ (x :: ('a::at-base)) \# (fv\ e :: 'a\ set) \longleftrightarrow atom\ x \# e$   
*<proof>*

**lemma** *finite-fv[simp]*:  $finite\ (fv\ (e::'a::fs) :: ('b::at-base)\ set)$   
*<proof>*

**definition** *fv-list* ::  $'a::fs \Rightarrow 'b::at-base\ list$   
**where** *fv-list*  $e = (SOME\ l.\ set\ l = fv\ e)$

**lemma** *set-fv-list[simp]*:  $set\ (fv-list\ e) = (fv\ e :: ('b::at-base)\ set)$   
*<proof>*

**lemma** *fresh-fv-list[simp]*:  
 $a \# (fv-list\ e :: 'b::at-base\ list) \longleftrightarrow a \# (fv\ e :: 'b::at-base\ set)$   
*<proof>*

## 1.8 Other useful lemmas

**lemma** *pure-permute-id*:  $permute\ p = (\lambda\ x.\ (x::'a::pure))$   
*<proof>*

**lemma** *supp-set-elem-finite*:  
**assumes** *finite*  $S$   
**and**  $(m::'a::fs) \in S$   
**and**  $y \in supp\ m$   
**shows**  $y \in supp\ S$   
*<proof>*

**lemmas** *fresh-star-Cons* = *fresh-star-list*(2)

**lemma** *mem-permute-set*:  
**shows**  $x \in p \cdot S \longleftrightarrow (-\ p \cdot x) \in S$   
*<proof>*

**lemma** *flip-set-both-not-in*:  
**assumes**  $x \notin S$  **and**  $x' \notin S$   
**shows**  $((x' \leftrightarrow x) \cdot S) = S$   
*<proof>*

**lemma** *inj-atom*: *inj atom* *<proof>*

**lemmas** *image-Int[OF inj-atom, simp]*

**lemma** *eqvt-uncurry*:  $eqvt\ f \implies eqvt\ (case-prod\ f)$   
*<proof>*

**lemma** *supp-fun-app-eqvt2*:

**assumes**  $a: \text{eqvt } f$   
**shows**  $\text{supp } (f \ x \ y) \subseteq \text{supp } x \cup \text{supp } y$   
 $\langle \text{proof} \rangle$

**lemma** *supp-fun-app-eqvt3*:  
**assumes**  $a: \text{eqvt } f$   
**shows**  $\text{supp } (f \ x \ y \ z) \subseteq \text{supp } x \cup \text{supp } y \cup \text{supp } z$   
 $\langle \text{proof} \rangle$

**lemma** *permute-0[simp]*:  $\text{permute } 0 = (\lambda \ x. \ x)$   
 $\langle \text{proof} \rangle$

**lemma** *permute-comp[simp]*:  $\text{permute } x \circ \text{permute } y = \text{permute } (x + y)$   $\langle \text{proof} \rangle$

**lemma** *map-permute*:  $\text{map } (\text{permute } p) = \text{permute } p$   
 $\langle \text{proof} \rangle$

**lemma** *fresh-star-restrictA[intro]*:  $a \#* \Gamma \Longrightarrow a \#* \text{AList.restrict } V \ \Gamma$   
 $\langle \text{proof} \rangle$

**lemma** *Abs-lst-Nil-eq[simp]*:  $[\ ] \text{lst. } (x::'a::fs) = [xs] \text{lst. } x' \longleftrightarrow (([\ ], x) = (xs, x^\wedge))$   
 $\langle \text{proof} \rangle$

**lemma** *Abs-lst-Nil-eq2[simp]*:  $[xs] \text{lst. } (x::'a::fs) = [\ ] \text{lst. } x' \longleftrightarrow ((xs, x) = ([\ ], x^\wedge))$   
 $\langle \text{proof} \rangle$

**lemma** *prod-cases8 [cases type]*:  
**obtains** (*fields*)  $a \ b \ c \ d \ e \ f \ g \ h$  **where**  $y = (a, b, c, d, e, f, g, h)$   
 $\langle \text{proof} \rangle$

**lemma** *prod-induct8 [case-names fields, induct type]*:  
 $(\bigwedge a \ b \ c \ d \ e \ f \ g \ h. P \ (a, b, c, d, e, f, g, h)) \Longrightarrow P \ x$   
 $\langle \text{proof} \rangle$

**lemma** *prod-cases9 [cases type]*:  
**obtains** (*fields*)  $a \ b \ c \ d \ e \ f \ g \ h \ i$  **where**  $y = (a, b, c, d, e, f, g, h, i)$   
 $\langle \text{proof} \rangle$

**lemma** *prod-induct9 [case-names fields, induct type]*:  
 $(\bigwedge a \ b \ c \ d \ e \ f \ g \ h \ i. P \ (a, b, c, d, e, f, g, h, i)) \Longrightarrow P \ x$   
 $\langle \text{proof} \rangle$

**named-theorems** *nominal-prod-simps*

**named-theorems** *ms-fresh Facts for helping with freshness proofs*

**lemma** *fresh-prod2[nominal-prod-simps,ms-fresh]*:  $x \# (a, b) = (x \# a \wedge x \# b)$   
 $\langle \text{proof} \rangle$

**lemma** *fresh-prod3[nominal-prod-simps,ms-fresh]*:  $x \# (a, b, c) = (x \# a \wedge x \# b \wedge x \# c)$   
 $\langle \text{proof} \rangle$

**lemma** *fresh-prod4*[*nominal-prod-simps,ms-fresh*]:  $x \# (a,b,c,d) = (x \# a \wedge x \# b \wedge x \# c \wedge x \# d)$   
*<proof>*

**lemma** *fresh-prod5*[*nominal-prod-simps,ms-fresh*]:  $x \# (a,b,c,d,e) = (x \# a \wedge x \# b \wedge x \# c \wedge x \# d \wedge x \# e)$   
*<proof>*

**lemma** *fresh-prod6*[*nominal-prod-simps,ms-fresh*]:  $x \# (a,b,c,d,e,f) = (x \# a \wedge x \# b \wedge x \# c \wedge x \# d \wedge x \# e \wedge x \# f)$   
*<proof>*

**lemma** *fresh-prod7*[*nominal-prod-simps,ms-fresh*]:  $x \# (a,b,c,d,e,f,g) = (x \# a \wedge x \# b \wedge x \# c \wedge x \# d \wedge x \# e \wedge x \# f \wedge x \# g)$   
*<proof>*

**lemma** *fresh-prod8*[*nominal-prod-simps,ms-fresh*]:  $x \# (a,b,c,d,e,f,g,h) = (x \# a \wedge x \# b \wedge x \# c \wedge x \# d \wedge x \# e \wedge x \# f \wedge x \# g \wedge x \# h)$   
*<proof>*

**lemma** *fresh-prod9*[*nominal-prod-simps,ms-fresh*]:  $x \# (a,b,c,d,e,f,g,h,i) = (x \# a \wedge x \# b \wedge x \# c \wedge x \# d \wedge x \# e \wedge x \# f \wedge x \# g \wedge x \# h \wedge x \# i)$   
*<proof>*

**lemma** *fresh-prod10*[*nominal-prod-simps,ms-fresh*]:  $x \# (a,b,c,d,e,f,g,h,i,j) = (x \# a \wedge x \# b \wedge x \# c \wedge x \# d \wedge x \# e \wedge x \# f \wedge x \# g \wedge x \# h \wedge x \# i \wedge x \# j)$   
*<proof>*

**lemma** *fresh-prod12*[*nominal-prod-simps,ms-fresh*]:  $x \# (a,b,c,d,e,f,g,h,i,j,k,l) = (x \# a \wedge x \# b \wedge x \# c \wedge x \# d \wedge x \# e \wedge x \# f \wedge x \# g \wedge x \# h \wedge x \# i \wedge x \# j \wedge x \# k \wedge x \# l)$   
*<proof>*

**lemmas** *fresh-prodN = fresh-Pair fresh-prod3 fresh-prod4 fresh-prod5 fresh-prod6 fresh-prod7 fresh-prod8 fresh-prod9 fresh-prod10 fresh-prod12*

**lemma** *fresh-prod2I*:  
fixes  $x$  and  $x1$  and  $x2$   
assumes  $x \# x1$  and  $x \# x2$   
shows  $x \# (x1,x2)$  *<proof>*

**lemma** *fresh-prod3I*:  
fixes  $x$  and  $x1$  and  $x2$  and  $x3$   
assumes  $x \# x1$  and  $x \# x2$  and  $x \# x3$   
shows  $x \# (x1,x2,x3)$  *<proof>*

**lemma** *fresh-prod4I*:  
fixes  $x$  and  $x1$  and  $x2$  and  $x3$  and  $x4$   
assumes  $x \# x1$  and  $x \# x2$  and  $x \# x3$  and  $x \# x4$   
shows  $x \# (x1,x2,x3,x4)$  *<proof>*

**lemma** *fresh-prod5I*:  
fixes  $x$  and  $x1$  and  $x2$  and  $x3$  and  $x4$  and  $x5$   
assumes  $x \# x1$  and  $x \# x2$  and  $x \# x3$  and  $x \# x4$  and  $x \# x5$

**shows**  $x \# (x1, x2, x3, x4, x5)$   $\langle proof \rangle$

**lemma** *flip-collapse[simp]*:

**fixes**  $b1::'a::pt$  **and**  $bv1::'b::at$  **and**  $bv2::'b::at$   
**assumes**  $atom\ bv2 \# b1$  **and**  $atom\ c \# (bv1, bv2, b1)$  **and**  $bv1 \neq bv2$   
**shows**  $(bv2 \leftrightarrow c) \cdot (bv1 \leftrightarrow bv2) \cdot b1 = (bv1 \leftrightarrow c) \cdot b1$   
 $\langle proof \rangle$

**lemma** *triple-eqvt[simp]*:

$p \cdot (x, b, c) = (p \cdot x, p \cdot b, p \cdot c)$   
 $\langle proof \rangle$

**lemma** *lst-fst*:

**fixes**  $x::'a::at$  **and**  $t1::'b::fs$  **and**  $x'::'a::at$  **and**  $t2::'c::fs$   
**assumes**  $([[atom\ x]]lst. (t1, t2) = [[atom\ x']]lst. (t1', t2'))$   
**shows**  $([[atom\ x]]lst. t1 = [[atom\ x']]lst. t1')$   
 $\langle proof \rangle$

**lemma** *lst-snd*:

**fixes**  $x::'a::at$  **and**  $t1::'b::fs$  **and**  $x'::'a::at$  **and**  $t2::'c::fs$   
**assumes**  $([[atom\ x]]lst. (t1, t2) = [[atom\ x']]lst. (t1', t2'))$   
**shows**  $([[atom\ x]]lst. t2 = [[atom\ x']]lst. t2')$   
 $\langle proof \rangle$

**lemma** *lst-head-cons-pair*:

**fixes**  $y1::'a::at$  **and**  $y2::'a::at$  **and**  $x1::'b::fs$  **and**  $x2::'b::fs$  **and**  $xs1::('b::fs)\ list$  **and**  $xs2::('b::fs)\ list$   
**assumes**  $([[atom\ y1]]lst. (x1 \# xs1) = [[atom\ y2]]lst. (x2 \# xs2))$   
**shows**  $([[atom\ y1]]lst. (x1, xs1) = [[atom\ y2]]lst. (x2, xs2))$   
 $\langle proof \rangle$

**lemma** *lst-head-cons-neq-nil*:

**fixes**  $y1::'a::at$  **and**  $y2::'a::at$  **and**  $x1::'b::fs$  **and**  $x2::'b::fs$  **and**  $xs1::('b::fs)\ list$  **and**  $xs2::('b::fs)\ list$   
**assumes**  $([[atom\ y1]]lst. (x1 \# xs1) = [[atom\ y2]]lst. (xs2))$   
**shows**  $xs2 \neq []$   
 $\langle proof \rangle$

**lemma** *lst-head-cons*:

**fixes**  $y1::'a::at$  **and**  $y2::'a::at$  **and**  $x1::'b::fs$  **and**  $x2::'b::fs$  **and**  $xs1::('b::fs)\ list$  **and**  $xs2::('b::fs)\ list$   
**assumes**  $([[atom\ y1]]lst. (x1 \# xs1) = [[atom\ y2]]lst. (x2 \# xs2))$   
**shows**  $([[atom\ y1]]lst. x1 = [[atom\ y2]]lst. x2 \text{ and } [[atom\ y1]]lst. xs1 = [[atom\ y2]]lst. xs2)$   
 $\langle proof \rangle$

**lemma** *lst-pure*:

**fixes**  $x1::'a::at$  **and**  $t1::'b::pure$  **and**  $x2::'a::at$  **and**  $t2::'b::pure$   
**assumes**  $([[atom\ x1]]lst. t1 = [[atom\ x2]]lst. t2)$   
**shows**  $t1 = t2$   
 $\langle proof \rangle$

**lemma** *lst-supp*:

**assumes**  $([[atom\ x1]]lst. t1 = [[atom\ x2]]lst. t2)$   
**shows**  $supp\ t1 - \{atom\ x1\} = supp\ t2 - \{atom\ x2\}$   
 $\langle proof \rangle$

**lemma** *lst-supp-subset*:

**assumes**  $[[atom\ x1]]lst.\ t1 = [[atom\ x2]]lst.\ t2$  **and**  $supp\ t1 \subseteq \{atom\ x1\} \cup B$

**shows**  $supp\ t2 \subseteq \{atom\ x2\} \cup B$

*<proof>*

**lemma** *projl-inl-eqvt*:

**fixes**  $\pi :: perm$

**shows**  $\pi \cdot (projl\ (Inl\ x)) = projl\ (Inl\ (\pi \cdot x))$

*<proof>*

**end**

# Chapter 2

## Syntax

Syntax of MiniSail programs and the contexts we use in judgements.

### 2.1 Program Syntax

#### 2.1.1 AST Datatypes

**type-synonym**  $num-nat = nat$

**atom-decl**  $x$

**atom-decl**  $u$

**atom-decl**  $bv$

**type-synonym**  $f = string$

**type-synonym**  $dc = string$

**type-synonym**  $tyid = string$

Basic types. Types without refinement constraints

**nominal-datatype**  $b =$

$B-int \mid B-bool \mid B-id \ tyid$   
 $\mid B-pair \ b \ b \ (\langle [-, -]^b \rangle)$   
 $\mid B-unit \mid B-bitvec \mid B-var \ bv$   
 $\mid B-app \ tyid \ b$

**nominal-datatype**  $bit = BitOne \mid BitZero$

Literals

**nominal-datatype**  $l =$

$L-num \ int \mid L-true \mid L-false \mid L-unit \mid L-bitvec \ bit \ list$

Values. We include a type identifier,  $tyid$ , in the literal for constructors to make typing and well-formedness checking easier

**nominal-datatype**  $v =$

$V-lit \ l \ (\langle [-]^v \rangle)$   
 $\mid V-var \ x \ (\langle [-]^v \rangle)$   
 $\mid V-pair \ v \ v \ (\langle [-, -]^v \rangle)$   
 $\mid V-cons \ tyid \ dc \ v$

| *V-consp tyid dc b v*

## Binary Operations

**nominal-datatype** *opp* = *Plus* (⟨*plus*⟩) | *LEq* (⟨*leq*⟩) | *Eq* (⟨*eq*⟩)

## Expressions

**nominal-datatype** *e* =  
*AE-val v* (⟨[-]<sup>*e*</sup> )  
| *AE-app f v* (⟨[- (-)]<sup>*e*</sup> )  
| *AE-appP f b v* (⟨[- [-] (-)]<sup>*e*</sup> )  
| *AE-op opp v v* (⟨[- - -]<sup>*e*</sup> )  
| *AE-concat v v* (⟨[- @@ -]<sup>*e*</sup> )  
| *AE-fst v* (⟨[#1-]<sup>*e*</sup> )  
| *AE-snd v* (⟨[#2-]<sup>*e*</sup> )  
| *AE-mvar u* (⟨[-]<sup>*e*</sup> )  
| *AE-len v* (⟨[| - |]<sup>*e*</sup> )  
| *AE-split v v* (⟨[- / -]<sup>*e*</sup> )

## Expressions for constraints

**nominal-datatype** *ce* =  
*CE-val v* (⟨[-]<sup>*ce*</sup> )  
| *CE-op opp ce ce* (⟨[- - -]<sup>*ce*</sup> )  
| *CE-concat ce ce* (⟨[- @@ -]<sup>*ce*</sup> )  
| *CE-fst ce* (⟨[#1-]<sup>*ce*</sup> )  
| *CE-snd ce* (⟨[#2-]<sup>*ce*</sup> )  
| *CE-len ce* (⟨[| - |]<sup>*ce*</sup> )

## Constraints

**nominal-datatype** *c* =  
*C-true* (⟨*TRUE*⟩ [] 50 )  
| *C-false* (⟨*FALSE*⟩ [] 50 )  
| *C-conj c c* (⟨*AND* - > [50, 50] 50 )  
| *C-disj c c* (⟨*OR* - > [50, 50] 50 )  
| *C-not c* (⟨*¬* - > [] 50 )  
| *C-imp c c* (⟨*IMP* - > [50, 50] 50 )  
| *C-eq ce ce* (⟨*==* - > [50, 50] 50 )

## Refined types

**nominal-datatype** *τ* =  
*T-refined-type x::x b c::c binds x in c* (⟨{ - : - | - }⟩ [50, 50] 1000)

## Statements

**nominal-datatype**  
*s* =  
*AS-val v* (⟨[-]<sup>*s*</sup> )  
| *AS-let x::x e s::s binds x in s* (⟨(*LET* - = - *IN* -)⟩ )  
| *AS-let2 x::x τ s::s binds x in s* (⟨(*LET* - : - = - *IN* -)⟩ )  
| *AS-if v s s* (⟨(*IF* - *THEN* - *ELSE* -)⟩ [0, 61, 0] 61 )  
| *AS-var u::u τ v s::s binds u in s* (⟨(*VAR* - : - = - *IN* -)⟩ )  
| *AS-assign u v* (⟨(- ::= -)⟩ )  
| *AS-match v branch-list* (⟨(*MATCH* - *WITH* { - })⟩ )

```

| AS-while s s (⟨(WHILE - DO { - } )⟩ [0, 0] 61)
| AS-seq s s (⟨(- ;; -)⟩ [1000, 61] 61)
| AS-assert c s (⟨(ASSERT - IN -)⟩)
  and branch-s =
  AS-branch dc x::x s::s binds x in s (⟨(- - ⇒ -)⟩)
  and branch-list =
  AS-final branch-s (⟨{ - }⟩)
| AS-cons branch-s branch-list (⟨(- | - )⟩)

```

Function and union type definitions

```

nominal-datatype fun-typ =
  AF-fun-typ x::x b c::c τ::τ s::s binds x in c τ s

```

```

nominal-datatype fun-typ-q =
  AF-fun-typ-some bv::bv ft::fun-typ binds bv in ft
| AF-fun-typ-none fun-typ

```

```

nominal-datatype fun-def = AF-fundef f fun-typ-q

```

```

nominal-datatype type-def =
  AF-typedef string (string * τ) list
| AF-typedef-poly string bv::bv dclist::(string * τ) list binds bv in dclist

```

**lemma** *check-typedef-poly*:

```

  AF-typedef-poly "option" bv [ ("None", { zz : B-unit | TRUE } ), ("Some", { zz : B-var bv | TRUE
} ) ] =
  AF-typedef-poly "option" bv2 [ ("None", { zz : B-unit | TRUE } ), ("Some", { zz : B-var bv2 |
TRUE } ) ]
⟨proof⟩

```

```

nominal-datatype var-def = AV-def u τ v

```

Programs

```

nominal-datatype p =
  AP-prog type-def list fun-def list var-def list s (⟨PROG - - - -⟩)

```

```

declare l.supp [simp] v.supp [simp] e.supp [simp] s-branch-s-branch-list.supp [simp] τ.supp [simp]
c.supp [simp] b.supp[simp]

```

## 2.1.2 Lemmas

These lemmas deal primarily with freshness and alpha-equivalence

### Atoms

```

lemma x-not-in-u-atoms[simp]:
  fixes u::u and x::x and us::u set
  shows atom x ∉ atom'us
⟨proof⟩

```

```

lemma x-fresh-u[simp]:
  fixes u::u and x::x

```

**shows**  $atom\ x \# u$   
 $\langle proof \rangle$

**lemma**  $x\text{-not-in-}b\text{-set}[simp]$ :  
**fixes**  $x::x$  **and**  $bs::bv\ fset$   
**shows**  $atom\ x \notin\ supp\ bs$   
 $\langle proof \rangle$

**lemma**  $x\text{-fresh-}b[simp]$ :  
**fixes**  $x::x$  **and**  $b::b$   
**shows**  $atom\ x \# b$   
 $\langle proof \rangle$

**lemma**  $x\text{-fresh-}bv[simp]$ :  
**fixes**  $x::x$  **and**  $bv::bv$   
**shows**  $atom\ x \# bv$   
 $\langle proof \rangle$

**lemma**  $u\text{-not-in-}x\text{-atoms}[simp]$ :  
**fixes**  $u::u$  **and**  $x::x$  **and**  $xs::x\ set$   
**shows**  $atom\ u \notin\ atom'xs$   
 $\langle proof \rangle$

**lemma**  $bv\text{-not-in-}x\text{-atoms}[simp]$ :  
**fixes**  $bv::bv$  **and**  $x::x$  **and**  $xs::x\ set$   
**shows**  $atom\ bv \notin\ atom'xs$   
 $\langle proof \rangle$

**lemma**  $u\text{-not-in-}b\text{-atoms}[simp]$ :  
**fixes**  $b::b$  **and**  $u::u$   
**shows**  $atom\ u \notin\ supp\ b$   
 $\langle proof \rangle$

**lemma**  $u\text{-not-in-}b\text{-set}[simp]$ :  
**fixes**  $u::u$  **and**  $bs::bv\ fset$   
**shows**  $atom\ u \notin\ supp\ bs$   
 $\langle proof \rangle$

**lemma**  $u\text{-fresh-}b[simp]$ :  
**fixes**  $x::u$  **and**  $b::b$   
**shows**  $atom\ x \# b$   
 $\langle proof \rangle$

**lemma**  $supp\text{-}b\text{-}v\text{-disjoint}$ :  
**fixes**  $x::x$  **and**  $bv::bv$   
**shows**  $supp\ (V\text{-var}\ x) \cap\ supp\ (B\text{-var}\ bv) = \{\}$   
 $\langle proof \rangle$

**lemma**  $supp\text{-}b\text{-}u\text{-disjoint}[simp]$ :  
**fixes**  $b::b$  **and**  $u::u$   
**shows**  $supp\ u \cap\ supp\ b = \{\}$   
 $\langle proof \rangle$

**lemma** *u-fresh-bv*[simp]:  
**fixes**  $u::u$  **and**  $b::bv$   
**shows**  $\text{atom } u \# b$   
 $\langle \text{proof} \rangle$

## Basic Types

**nominal-function** *b-of*  $:: \tau \Rightarrow b$  **where**  
 $b\text{-of } \{ z : b \mid c \} = b$   
 $\langle \text{proof} \rangle$

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

**lemma** *supp-b-empty*[simp]:  
**fixes**  $b::b$  **and**  $x::x$   
**shows**  $\text{atom } x \notin \text{supp } b$   
 $\langle \text{proof} \rangle$

**lemma** *flip-b-id*[simp]:  
**fixes**  $x::x$  **and**  $b::b$   
**shows**  $(x \leftrightarrow x') \cdot b = b$   
 $\langle \text{proof} \rangle$

**lemma** *flip-x-b-cancel*[simp]:  
**fixes**  $x::x$  **and**  $y::x$  **and**  $b::b$  **and**  $bv::bv$   
**shows**  $(x \leftrightarrow y) \cdot b = b$  **and**  $(x \leftrightarrow y) \cdot bv = bv$   
 $\langle \text{proof} \rangle$

**lemma** *flip-bv-x-cancel*[simp]:  
**fixes**  $bv::bv$  **and**  $z::bv$  **and**  $x::x$   
**shows**  $(bv \leftrightarrow z) \cdot x = x$   $\langle \text{proof} \rangle$

**lemma** *flip-bv-u-cancel*[simp]:  
**fixes**  $bv::bv$  **and**  $z::bv$  **and**  $x::u$   
**shows**  $(bv \leftrightarrow z) \cdot x = x$   $\langle \text{proof} \rangle$

## Literals

**lemma** *supp-bitvec-empty*:  
**fixes**  $bv::\text{bit list}$   
**shows**  $\text{supp } bv = \{\}$   
 $\langle \text{proof} \rangle$

**lemma** *bitvec-pure*[simp]:  
**fixes**  $bv::\text{bit list}$  **and**  $x::x$   
**shows**  $\text{atom } x \# bv$   $\langle \text{proof} \rangle$

**lemma** *supp-l-empty*[simp]:  
**fixes**  $l::l$   
**shows**  $\text{supp } (V\text{-lit } l) = \{\}$   
 $\langle \text{proof} \rangle$

**lemma** *type-l-nosupp*[simp]:

**fixes**  $x::x$  **and**  $l::l$   
**shows**  $atom\ x \notin\ supp\ (\{\{z : b \mid [[z]^v]^{ce} == [[l]^v]^{ce}\}\})$   
 $\langle proof \rangle$

**lemma** *flip-bitvec0*:  
**fixes**  $x::bit\ list$   
**assumes**  $atom\ c \# (z, x, z')$   
**shows**  $(z \leftrightarrow c) \cdot x = (z' \leftrightarrow c) \cdot x$   
 $\langle proof \rangle$

**lemma** *flip-bitvec*:  
**assumes**  $atom\ c \# (z, L-bitvec\ x, z')$   
**shows**  $(z \leftrightarrow c) \cdot x = (z' \leftrightarrow c) \cdot x$   
 $\langle proof \rangle$

**lemma** *type-l-eq*:  
**shows**  $\{\{z : b \mid [[z]^v]^{ce} == [V-lit\ l]^{ce}\}\} = (\{\{z' : b \mid [[z']^v]^{ce} == [V-lit\ l]^{ce}\}\})$   
 $\langle proof \rangle$

**lemma** *flip-l-eq*:  
**fixes**  $x::l$   
**shows**  $(z \leftrightarrow c) \cdot x = (z' \leftrightarrow c) \cdot x$   
 $\langle proof \rangle$

**lemma** *flip-l-eq1*:  
**fixes**  $x::l$   
**assumes**  $(z \leftrightarrow c) \cdot x = (z' \leftrightarrow c) \cdot x'$   
**shows**  $x' = x$   
 $\langle proof \rangle$

## Types

**lemma** *flip-base-eq*:  
**fixes**  $b::b$  **and**  $x::x$  **and**  $y::x$   
**shows**  $(x \leftrightarrow y) \cdot b = b$   
 $\langle proof \rangle$

Obtain an alpha-equivalent type where the bound variable is fresh in some term t

**lemma** *has-fresh-z0*:  
**fixes**  $t::'b::fs$   
**shows**  $\exists z. atom\ z \# (c',t) \wedge (\{\{z' : b \mid c'\}\}) = (\{\{z : b \mid (z \leftrightarrow z') \cdot c'\}\})$   
 $\langle proof \rangle$

**lemma** *has-fresh-z*:  
**fixes**  $t::'b::fs$   
**shows**  $\exists z\ b\ c. atom\ z \# t \wedge \tau = \{\{z : b \mid c\}\}$   
 $\langle proof \rangle$

**lemma** *obtain-fresh-z*:  
**fixes**  $t::'b::fs$   
**obtains**  $z$  **and**  $b$  **and**  $c$  **where**  $atom\ z \# t \wedge \tau = \{\{z : b \mid c\}\}$   
 $\langle proof \rangle$

**lemma** *has-fresh-z2*:

**fixes**  $t::'b::fs$

**shows**  $\exists z c. \text{atom } z \# t \wedge \tau = \{ \{ z : \text{b-of } \tau \mid c \} \}$

$\langle \text{proof} \rangle$

**lemma** *obtain-fresh-z2*:

**fixes**  $t::'b::fs$

**obtains**  $z$  **and**  $c$  **where**  $\text{atom } z \# t \wedge \tau = \{ \{ z : \text{b-of } \tau \mid c \} \}$

$\langle \text{proof} \rangle$

## Values

**lemma** *u-notin-supp-v[simp]*:

**fixes**  $u::u$  **and**  $v::v$

**shows**  $\text{atom } u \notin \text{supp } v$

$\langle \text{proof} \rangle$

**lemma** *u-fresh-xv[simp]*:

**fixes**  $u::u$  **and**  $x::x$  **and**  $v::v$

**shows**  $\text{atom } u \# (x, v)$

$\langle \text{proof} \rangle$

Part of an effort to make the proofs across inductive cases more uniform by distilling the non-uniform parts into lemmas like this

**lemma** *v-flip-eq*:

**fixes**  $v::v$  **and**  $va::v$  **and**  $x::x$  **and**  $c::x$

**assumes**  $\text{atom } c \# (v, va)$  **and**  $\text{atom } c \# (x, xa, v, va)$  **and**  $(x \leftrightarrow c) \cdot v = (xa \leftrightarrow c) \cdot va$

**shows**  $((v = V\text{-lit } l \longrightarrow (\exists l'. va = V\text{-lit } l' \wedge (x \leftrightarrow c) \cdot l = (xa \leftrightarrow c) \cdot l')) \wedge$

$((v = V\text{-var } y \longrightarrow (\exists y'. va = V\text{-var } y' \wedge (x \leftrightarrow c) \cdot y = (xa \leftrightarrow c) \cdot y')) \wedge$

$((v = V\text{-pair } vone \ vtwo \longrightarrow (\exists v1' \ v2'. va = V\text{-pair } v1' \ v2' \wedge (x \leftrightarrow c) \cdot vone = (xa \leftrightarrow c) \cdot v1' \wedge (x \leftrightarrow c) \cdot vtwo = (xa \leftrightarrow c) \cdot v2')) \wedge$

$((v = V\text{-cons } tyid \ dc \ vone \longrightarrow (\exists v1'. va = V\text{-cons } tyid \ dc \ v1' \wedge (x \leftrightarrow c) \cdot vone = (xa \leftrightarrow c) \cdot v1')) \wedge$

$((v = V\text{-consp } tyid \ dc \ b \ vone \longrightarrow (\exists v1'. va = V\text{-consp } tyid \ dc \ b \ v1' \wedge (x \leftrightarrow c) \cdot vone = (xa \leftrightarrow c) \cdot v1'))$

$\langle \text{proof} \rangle$

**lemma** *flip-eq*:

**fixes**  $x::x$  **and**  $xa::x$  **and**  $s::'a::fs$  **and**  $sa::'a::fs$

**assumes**  $(\forall c. \text{atom } c \# (s, sa) \longrightarrow \text{atom } c \# (x, xa, s, sa) \longrightarrow (x \leftrightarrow c) \cdot s = (xa \leftrightarrow c) \cdot sa)$  **and**  $x \neq xa$

**shows**  $(x \leftrightarrow xa) \cdot s = sa$

$\langle \text{proof} \rangle$

**lemma** *swap-v-supp*:

**fixes**  $v::v$  **and**  $d::x$  **and**  $z::x$

**assumes**  $\text{atom } d \# v$

**shows**  $\text{supp } ((z \leftrightarrow d) \cdot v) \subseteq \text{supp } v - \{ \text{atom } z \} \cup \{ \text{atom } d \}$

$\langle \text{proof} \rangle$

## Expressions

**lemma** *swap-e-supp*:

**fixes**  $e::e$  **and**  $d::x$  **and**  $z::x$   
**assumes**  $atom\ d \# e$   
**shows**  $supp\ ((z \leftrightarrow d) \cdot e) \subseteq supp\ e - \{atom\ z\} \cup \{atom\ d\}$   
 $\langle proof \rangle$

**lemma** *swap-ce-supp*:  
**fixes**  $e::ce$  **and**  $d::x$  **and**  $z::x$   
**assumes**  $atom\ d \# e$   
**shows**  $supp\ ((z \leftrightarrow d) \cdot e) \subseteq supp\ e - \{atom\ z\} \cup \{atom\ d\}$   
 $\langle proof \rangle$

**lemma** *swap-c-supp*:  
**fixes**  $c::c$  **and**  $d::x$  **and**  $z::x$   
**assumes**  $atom\ d \# c$   
**shows**  $supp\ ((z \leftrightarrow d) \cdot c) \subseteq supp\ c - \{atom\ z\} \cup \{atom\ d\}$   
 $\langle proof \rangle$

**lemma** *type-e-eq*:  
**assumes**  $atom\ z \# e$  **and**  $atom\ z' \# e$   
**shows**  $\{z : b \mid [[z]^v]^{ce} == e\} = (\{z' : b \mid [[z']^v]^{ce} == e\})$   
 $\langle proof \rangle$

**lemma** *type-e-eq2*:  
**assumes**  $atom\ z \# e$  **and**  $atom\ z' \# e$  **and**  $b=b'$   
**shows**  $\{z : b \mid [[z]^v]^{ce} == e\} = (\{z' : b' \mid [[z']^v]^{ce} == e\})$   
 $\langle proof \rangle$

**lemma** *e-flip-eq*:  
**fixes**  $e::e$  **and**  $ea::e$   
**assumes**  $atom\ c \# (e, ea)$  **and**  $atom\ c \# (x, xa, e, ea)$  **and**  $(x \leftrightarrow c) \cdot e = (xa \leftrightarrow c) \cdot ea$   
**shows**  $(e = AE-val\ w \longrightarrow (\exists w'. ea = AE-val\ w' \wedge (x \leftrightarrow c) \cdot w = (xa \leftrightarrow c) \cdot w')) \vee$   
 $(e = AE-op\ opp\ v1\ v2 \longrightarrow (\exists v1'\ v2'. ea = AS-op\ opp\ v1'\ v2' \wedge (x \leftrightarrow c) \cdot v1 = (xa \leftrightarrow c) \cdot v1'))$   
 $\wedge (x \leftrightarrow c) \cdot v2 = (xa \leftrightarrow c) \cdot v2') \vee$   
 $(e = AE-fst\ v \longrightarrow (\exists v'. ea = AE-fst\ v' \wedge (x \leftrightarrow c) \cdot v = (xa \leftrightarrow c) \cdot v')) \vee$   
 $(e = AE-snd\ v \longrightarrow (\exists v'. ea = AE-snd\ v' \wedge (x \leftrightarrow c) \cdot v = (xa \leftrightarrow c) \cdot v')) \vee$   
 $(e = AE-len\ v \longrightarrow (\exists v'. ea = AE-len\ v' \wedge (x \leftrightarrow c) \cdot v = (xa \leftrightarrow c) \cdot v')) \vee$   
 $(e = AE-concat\ v1\ v2 \longrightarrow (\exists v1'\ v2'. ea = AS-concat\ v1'\ v2' \wedge (x \leftrightarrow c) \cdot v1 = (xa \leftrightarrow c) \cdot v1'))$   
 $\wedge (x \leftrightarrow c) \cdot v2 = (xa \leftrightarrow c) \cdot v2') \vee$   
 $(e = AE-app\ f\ v \longrightarrow (\exists v'. ea = AE-app\ f\ v' \wedge (x \leftrightarrow c) \cdot v = (xa \leftrightarrow c) \cdot v'))$   
 $\langle proof \rangle$

**lemma** *fresh-opp-all*:  
**fixes**  $opp::opp$   
**shows**  $z \# opp$   
 $\langle proof \rangle$

**lemma** *fresh-e-opp-all*:  
**shows**  $(z \# v1 \wedge z \# v2) = z \# AE-op\ opp\ v1\ v2$   
 $\langle proof \rangle$

**lemma** *fresh-e-opp*:  
**fixes**  $z::x$

**assumes**  $atom\ z \# v1 \wedge atom\ z \# v2$   
**shows**  $atom\ z \# AE-op\ opp\ v1\ v2$   
 $\langle proof \rangle$

## Statements

**lemma** *branch-s-flip-eq*:

**fixes**  $v::v$  **and**  $va::v$

**assumes**  $atom\ c \# (v, va)$  **and**  $atom\ c \# (x, xa, v, va)$  **and**  $(x \leftrightarrow c) \cdot s = (xa \leftrightarrow c) \cdot sa$

**shows**  $(s = AS-val\ w \longrightarrow (\exists w'. sa = AS-val\ w' \wedge (x \leftrightarrow c) \cdot w = (xa \leftrightarrow c) \cdot w')) \vee$

$(s = AS-seq\ s1\ s2 \longrightarrow (\exists s1'\ s2'. sa = AS-seq\ s1'\ s2' \wedge (x \leftrightarrow c) \cdot s1 = (xa \leftrightarrow c) \cdot s1') \wedge (x \leftrightarrow c) \cdot s2 = (xa \leftrightarrow c) \cdot s2') \vee$

$(s = AS-if\ v\ s1\ s2 \longrightarrow (\exists v'\ s1'\ s2'. sa = AS-if\ seq\ s1'\ s2' \wedge (x \leftrightarrow c) \cdot s1 = (xa \leftrightarrow c) \cdot s1') \wedge (x \leftrightarrow c) \cdot s2 = (xa \leftrightarrow c) \cdot s2' \wedge (x \leftrightarrow c) \cdot c = (xa \leftrightarrow c) \cdot v')$

$\langle proof \rangle$

## 2.2 Context Syntax

### 2.2.1 Datatypes

Type and function/type definition contexts

**type-synonym**  $\Phi = fun-def\ list$

**type-synonym**  $\Theta = type-def\ list$

**type-synonym**  $\mathcal{B} = bv\ fset$

**datatype**  $\Gamma =$

$GNil$

|  $GCons\ x*b*c\ \Gamma$  (**infixr**  $\langle \#_{\Gamma} \rangle$  65)

**datatype**  $\Delta =$

$DNil$  ( $\langle \#_{\Delta} \rangle$ )

|  $DCons\ u*\tau\ \Delta$  (**infixr**  $\langle \#_{\Delta} \rangle$  65)

### 2.2.2 Functions and Lemmas

**lemma**  $\Gamma$ -*induct* [*case-names*  $GNil\ GCons$ ] :  $P\ GNil \Longrightarrow (\bigwedge x\ b\ c\ \Gamma'. P\ \Gamma' \Longrightarrow P\ ((x,b,c) \#_{\Gamma} \Gamma')) \Longrightarrow P\ \Gamma$

$\langle proof \rangle$

**instantiation**  $\Delta :: pt$

**begin**

**primrec** *permute- $\Delta$*

**where**

$DNil$ -*eqvt*:  $p \cdot DNil = DNil$

|  $DCons$ -*eqvt*:  $p \cdot (x \#_{\Delta} xs) = p \cdot x \#_{\Delta} p \cdot (xs::\Delta)$

**instance**  $\langle proof \rangle$

**end**

**lemmas** [*eqvt*] = *permute- $\Delta$ .simps*

**lemma**  $\Delta$ -induct [case-names DNil DCons] : P DNil  $\implies$  ( $\bigwedge u t \Delta'. P \Delta' \implies P ((u,t) \#_{\Delta} \Delta')$ )  $\implies P \Delta$   
 ⟨proof⟩

**lemma**  $\Phi$ -induct [case-names PNil PConsNone PConsSome] : P []  $\implies$  ( $\bigwedge f x b c \tau s' \Phi'. P \Phi' \implies P ((AF-fundef f (AF-fun-typ-none (AF-fun-typ x b c \tau s')) \# \Phi')$ )  $\implies$   
 ( $\bigwedge bv x b c \tau s' \Phi'. P \Phi' \implies P ((AF-fundef f (AF-fun-typ-some bv (AF-fun-typ x b c \tau s')) \# \Phi')$ )  $\implies P \Phi$   
 ⟨proof⟩

**lemma**  $\Theta$ -induct [case-names TNil AF-typedef AF-typedef-poly] : P []  $\implies$  ( $\bigwedge tid dclist \Theta'. P \Theta' \implies P ((AF-typedef tid dclist) \# \Theta')$ )  $\implies$   
 ( $\bigwedge tid bv dclist \Theta'. P \Theta' \implies P ((AF-typedef-poly tid bv dclist) \# \Theta')$ )  $\implies P \Theta$   
 ⟨proof⟩

**instantiation**  $\Gamma :: pt$   
**begin**

**primrec** permute- $\Gamma$   
**where**

$GNil$ -eqvt:  $p \cdot GNil = GNil$   
 |  $GCons$ -eqvt:  $p \cdot (x \#_{\Gamma} xs) = p \cdot x \#_{\Gamma} p \cdot (xs :: \Gamma)$

**instance** ⟨proof⟩  
**end**

**lemmas** [eqvt] = permute- $\Gamma$ .simps

**lemma**  $G$ -cons-eqvt[simp]:  
**fixes**  $\Gamma :: \Gamma$   
**shows**  $p \cdot ((x,b,c) \#_{\Gamma} \Gamma) = ((p \cdot x, p \cdot b, p \cdot c) \#_{\Gamma} (p \cdot \Gamma))$  (is ?A = ?B)  
 ⟨proof⟩

**lemma**  $G$ -cons-flip[simp]:  
**fixes**  $x :: x$  **and**  $\Gamma :: \Gamma$   
**shows**  $(x \leftrightarrow x') \cdot ((x'',b,c) \#_{\Gamma} \Gamma) = (((x \leftrightarrow x') \cdot x'', b, (x \leftrightarrow x') \cdot c) \#_{\Gamma} ((x \leftrightarrow x') \cdot \Gamma))$   
 ⟨proof⟩

**lemma**  $G$ -cons-flip-fresh[simp]:  
**fixes**  $x :: x$  **and**  $\Gamma :: \Gamma$   
**assumes**  $atom\ x \# (c, \Gamma)$  **and**  $atom\ x' \# (c, \Gamma)$   
**shows**  $(x \leftrightarrow x') \cdot ((x',b,c) \#_{\Gamma} \Gamma) = ((x, b, c) \#_{\Gamma} \Gamma)$   
 ⟨proof⟩

**lemma**  $G$ -cons-flip-fresh2[simp]:  
**fixes**  $x :: x$  **and**  $\Gamma :: \Gamma$   
**assumes**  $atom\ x \# (c, \Gamma)$  **and**  $atom\ x' \# (c, \Gamma)$   
**shows**  $(x \leftrightarrow x') \cdot ((x,b,c) \#_{\Gamma} \Gamma) = ((x', b, c) \#_{\Gamma} \Gamma)$   
 ⟨proof⟩

**lemma**  $G$ -cons-flip-fresh3[simp]:

**fixes**  $x::x$  and  $\Gamma::\Gamma$   
**assumes**  $atom\ x \# \Gamma$  and  $atom\ x' \# \Gamma$   
**shows**  $(x \leftrightarrow x') \cdot ((x', b, c) \#_{\Gamma} \Gamma) = ((x, b, (x \leftrightarrow x') \cdot c) \#_{\Gamma} \Gamma)$   
 $\langle proof \rangle$

**lemma** *neq-GNil-conv*:  $(xs \neq GNil) = (\exists y\ ys.\ xs = y \#_{\Gamma} ys)$   
 $\langle proof \rangle$

**nominal-function** *toList* ::  $\Gamma \Rightarrow (x*b*c)$  list **where**  
 $toList\ GNil = []$   
 $| toList\ (GCons\ xbc\ G) = xbc \# (toList\ G)$   
 $\langle proof \rangle$   
**nominal-termination** (*eqvt*)  
 $\langle proof \rangle$

**nominal-function** *toSet* ::  $\Gamma \Rightarrow (x*b*c)$  set **where**  
 $toSet\ GNil = \{\}$   
 $| toSet\ (GCons\ xbc\ G) = \{xbc\} \cup (toSet\ G)$   
 $\langle proof \rangle$   
**nominal-termination** (*eqvt*)  
 $\langle proof \rangle$

**nominal-function** *append-g* ::  $\Gamma \Rightarrow \Gamma \Rightarrow \Gamma$  (**infixr**  $\langle @ \rangle$  65) **where**  
 $append-g\ GNil\ g = g$   
 $| append-g\ (xbc \#_{\Gamma} g1)\ g2 = (xbc \#_{\Gamma} (g1 @ g2))$   
 $\langle proof \rangle$   
**nominal-termination** (*eqvt*)  $\langle proof \rangle$

**nominal-function** *dom* ::  $\Gamma \Rightarrow x$  set **where**  
 $dom\ \Gamma = (fst' (toSet\ \Gamma))$   
 $\langle proof \rangle$   
**nominal-termination** (*eqvt*)  $\langle proof \rangle$

Use of this is sometimes mixed in with use of freshness and support for the context however it makes it clear that for immutable variables, the context is ‘self-supporting’

**nominal-function** *atom-dom* ::  $\Gamma \Rightarrow atom$  set **where**  
 $atom-dom\ \Gamma = atom'(dom\ \Gamma)$   
 $\langle proof \rangle$   
**nominal-termination** (*eqvt*)  $\langle proof \rangle$

### 2.2.3 Immutable Variable Context Lemmas

**lemma** *append-GNil[simp]*:  
 $GNil @ G = G$   
 $\langle proof \rangle$

**lemma** *append-g-toSetU [simp]*:  $toSet\ (G1 @ G2) = toSet\ G1 \cup toSet\ G2$   
 $\langle proof \rangle$

**lemma** *supp-GNil*:  
**shows**  $supp\ GNil = \{\}$   
 $\langle proof \rangle$

**lemma** *supp-GCons*:

**fixes**  $xs::\Gamma$

**shows**  $\text{supp } (x \#_{\Gamma} xs) = \text{supp } x \cup \text{supp } xs$

$\langle \text{proof} \rangle$

**lemma** *atom-dom-eq[simp]*:

**fixes**  $G::\Gamma$

**shows**  $\text{atom-dom } ((x, b, c) \#_{\Gamma} G) = \text{atom-dom } ((x, b, c') \#_{\Gamma} G)$

$\langle \text{proof} \rangle$

**lemma** *dom-append[simp]*:

$\text{atom-dom } (\Gamma @ \Gamma') = \text{atom-dom } \Gamma \cup \text{atom-dom } \Gamma'$

$\langle \text{proof} \rangle$

**lemma** *dom-cons[simp]*:

$\text{atom-dom } ((x, b, c) \#_{\Gamma} G) = \{ \text{atom } x \} \cup \text{atom-dom } G$

$\langle \text{proof} \rangle$

**lemma** *fresh-GNil[ms-fresh]*:

**shows**  $a \# GNil$

$\langle \text{proof} \rangle$

**lemma** *fresh-GCons[ms-fresh]*:

**fixes**  $xs::\Gamma$

**shows**  $a \# (x \#_{\Gamma} xs) \longleftrightarrow a \# x \wedge a \# xs$

$\langle \text{proof} \rangle$

**lemma** *dom-supp-g[simp]*:

$\text{atom-dom } G \subseteq \text{supp } G$

$\langle \text{proof} \rangle$

**lemma** *fresh-append-g[ms-fresh]*:

**fixes**  $xs::\Gamma$

**shows**  $a \# (xs @ ys) \longleftrightarrow a \# xs \wedge a \# ys$

$\langle \text{proof} \rangle$

**lemma** *append-g-assoc*:

**fixes**  $xs::\Gamma$

**shows**  $(xs @ ys) @ zs = xs @ (ys @ zs)$

$\langle \text{proof} \rangle$

**lemma** *append-g-inside*:

**fixes**  $xs::\Gamma$

**shows**  $xs @ (x \#_{\Gamma} ys) = (xs @ (x \#_{\Gamma} GNil)) @ ys$

$\langle \text{proof} \rangle$

**lemma** *finite-Γ*:

*finite* (*toSet*  $\Gamma$ )

$\langle \text{proof} \rangle$

**lemma** *supp-Γ*:

$supp \Gamma = supp (toSet \Gamma)$   
 $\langle proof \rangle$

**lemma** *supp-of-subset*:  
**fixes**  $G::('a::fs \text{ set})$   
**assumes** *finite G and finite G' and  $G \subseteq G'$*   
**shows**  $supp G \subseteq supp G'$   
 $\langle proof \rangle$

**lemma** *supp-weakening*:  
**assumes**  $toSet G \subseteq toSet G'$   
**shows**  $supp G \subseteq supp G'$   
 $\langle proof \rangle$

**lemma** *fresh-weakening[ms-fresh]*:  
**assumes**  $toSet G \subseteq toSet G'$  **and**  $x \# G'$   
**shows**  $x \# G$   
 $\langle proof \rangle$

**instance**  $\Gamma :: fs$   
 $\langle proof \rangle$

**lemma** *fresh-gamma-elem*:  
**fixes**  $\Gamma::\Gamma$   
**assumes**  $a \# \Gamma$   
**and**  $e \in toSet \Gamma$   
**shows**  $a \# e$   
 $\langle proof \rangle$

**lemma** *fresh-gamma-append*:  
**fixes**  $xs::\Gamma$   
**shows**  $a \# (xs @ ys) \longleftrightarrow a \# xs \wedge a \# ys$   
 $\langle proof \rangle$

**lemma** *supp-triple[simp]*:  
**shows**  $supp (x, y, z) = supp x \cup supp y \cup supp z$   
 $\langle proof \rangle$

**lemma** *supp-append-g*:  
**fixes**  $xs::\Gamma$   
**shows**  $supp (xs @ ys) = supp xs \cup supp ys$   
 $\langle proof \rangle$

**lemma** *fresh-in-g[simp]*:  
**fixes**  $\Gamma::\Gamma$  **and**  $x'::x$   
**shows**  $atom x' \# \Gamma' @ (x, b0, c0) \#_{\Gamma} \Gamma = (atom x' \notin supp \Gamma' \cup supp x \cup supp b0 \cup supp c0 \cup supp \Gamma)$   
 $\langle proof \rangle$

**lemma** *fresh-suffix[ms-fresh]*:  
**fixes**  $\Gamma::\Gamma$   
**assumes**  $atom x \# \Gamma' @ \Gamma$

**shows**  $atom\ x \# \Gamma$   
*<proof>*

**lemma** *not-GCons-self* [*simp*]:  
**fixes**  $xs::\Gamma$   
**shows**  $xs \neq x \#_{\Gamma} xs$   
*<proof>*

**lemma** *not-GCons-self2* [*simp*]:  
**fixes**  $xs::\Gamma$   
**shows**  $x \#_{\Gamma} xs \neq xs$   
*<proof>*

**lemma** *fresh-restrict*:  
**fixes**  $y::x$  **and**  $\Gamma::\Gamma$   
**assumes**  $atom\ y \# (\Gamma' @ (x, b, c) \#_{\Gamma} \Gamma)$   
**shows**  $atom\ y \# (\Gamma' @ \Gamma)$   
*<proof>*

**lemma** *fresh-dom-free*:  
**assumes**  $atom\ x \# \Gamma$   
**shows**  $(x, b, c) \notin toSet\ \Gamma$   
*<proof>*

**lemma**  $\Gamma$ -*set-intros*:  $x \in toSet\ (x \#_{\Gamma} xs)$  **and**  $y \in toSet\ xs \implies y \in toSet\ (x \#_{\Gamma} xs)$   
*<proof>*

**lemma** *fresh-dom-free2*:  
**assumes**  $atom\ x \notin atom-dom\ \Gamma$   
**shows**  $(x, b, c) \notin toSet\ \Gamma$   
*<proof>*

## 2.2.4 Mutable Variable Context Lemmas

**lemma** *supp-DNil*:  
**shows**  $supp\ DNil = \{\}$   
*<proof>*

**lemma** *supp-DCons*:  
**fixes**  $xs::\Delta$   
**shows**  $supp\ (x \#_{\Delta} xs) = supp\ x \cup supp\ xs$   
*<proof>*

**lemma** *fresh-DNil[ms-fresh]*:  
**shows**  $a \# DNil$   
*<proof>*

**lemma** *fresh-DCons[ms-fresh]*:  
**fixes**  $xs::\Delta$   
**shows**  $a \# (x \#_{\Delta} xs) \longleftrightarrow a \# x \wedge a \# xs$   
*<proof>*

**instance**  $\Delta :: fs$

$\langle \text{proof} \rangle$

## 2.2.5 Lookup Functions

**nominal-function**  $\text{lookup} :: \Gamma \Rightarrow x \Rightarrow (b*c) \text{ option}$  **where**

$\text{lookup } GNil \ x = None$

|  $\text{lookup } ((x,b,c)\#_{\Gamma} G) \ y = (\text{if } x=y \text{ then } Some \ (b,c) \ \text{else } \text{lookup } G \ y)$

$\langle \text{proof} \rangle$

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

**nominal-function**  $\text{replace-in-g} :: \Gamma \Rightarrow x \Rightarrow c \Rightarrow \Gamma \ \langle \text{[-+→-]} \rangle \ [1000,0,0] \ 200$  **where**

$\text{replace-in-g } GNil \ - = GNil$

|  $\text{replace-in-g } ((x,b,c)\#_{\Gamma} G) \ x' \ c' = (\text{if } x=x' \ \text{then } ((x,b,c')\#_{\Gamma} G) \ \text{else } (x,b,c)\#_{\Gamma} (\text{replace-in-g } G \ x' \ c'))$

$\langle \text{proof} \rangle$

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

Functions for looking up data-constructors in the Pi context

**nominal-function**  $\text{lookup-fun} :: \Phi \Rightarrow f \Rightarrow \text{fun-def option}$  **where**

$\text{lookup-fun } [] \ g = None$

|  $\text{lookup-fun } ((AF\text{-fundef } f \ ft)\#_{\Pi}) \ g = (\text{if } (f=g) \ \text{then } Some \ (AF\text{-fundef } f \ ft) \ \text{else } \text{lookup-fun } \Pi \ g)$

$\langle \text{proof} \rangle$

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

**nominal-function**  $\text{lookup-td} :: \Theta \Rightarrow \text{string} \Rightarrow \text{type-def option}$  **where**

$\text{lookup-td } [] \ g = None$

|  $\text{lookup-td } ((AF\text{-typedef } s \ lst) \# (\Theta::\Theta)) \ g = (\text{if } (s = g) \ \text{then } Some \ (AF\text{-typedef } s \ lst) \ \text{else } \text{lookup-td } \Theta \ g)$

|  $\text{lookup-td } ((AF\text{-typedef-poly } s \ bv \ lst) \# (\Theta::\Theta)) \ g = (\text{if } (s = g) \ \text{then } Some \ (AF\text{-typedef-poly } s \ bv \ lst) \ \text{else } \text{lookup-td } \Theta \ g)$

$\langle \text{proof} \rangle$

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

**nominal-function**  $\text{name-of-type} :: \text{type-def} \Rightarrow f$  **where**

$\text{name-of-type } (AF\text{-typedef } f \ -) = f$

|  $\text{name-of-type } (AF\text{-typedef-poly } f \ -) = f$

$\langle \text{proof} \rangle$

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

**nominal-function**  $\text{name-of-fun} :: \text{fun-def} \Rightarrow f$  **where**

$\text{name-of-fun } (AF\text{-fundef } f \ ft) = f$

$\langle \text{proof} \rangle$

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

**nominal-function**  $\text{remove2} :: 'a::\text{pt} \Rightarrow 'a \ \text{list} \Rightarrow 'a \ \text{list}$  **where**

$\text{remove2 } x \ [] = []$

$\text{remove2 } x \ (y \# \ xs) = (\text{if } x = y \ \text{then } \ xs \ \text{else } y \# \ \text{remove2 } x \ \ xs)$

$\langle \text{proof} \rangle$

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

**nominal-function**  $\text{base-for-lit} :: l \Rightarrow b$  **where**

$\text{base-for-lit } (L\text{-true}) = B\text{-bool}$

|  $\text{base-for-lit } (L\text{-false}) = B\text{-bool}$

|  $\text{base-for-lit } (L\text{-num } n) = B\text{-int}$

| *base-for-lit* (*L-unit*) = *B-unit*  
| *base-for-lit* (*L-bitvec v*) = *B-bitvec*  
⟨*proof*⟩

**nominal-termination** (*eqvt*) ⟨*proof*⟩

**lemma** *neg-DNil-conv*: (*xs* ≠ *DNil*) = ( $\exists y$  *ys*. *xs* = *y* #<sub>Δ</sub> *ys*)  
⟨*proof*⟩

**nominal-function** *setD* :: Δ ⇒ (*u*\*τ) *set* **where**

*setD* *DNil* = {}  
| *setD* (*DCons xbc G*) = {*xbc*} ∪ (*setD G*)  
⟨*proof*⟩

**nominal-termination** (*eqvt*) ⟨*proof*⟩

**lemma** *eqvt-triple*:

**fixes** *y*::'a::at **and** *ya*::'a::at **and** *xa*::'c::at **and** *va*::'d::fs **and** *s*::s **and** *sa*::s **and** *f*::s\*'c\*'d ⇒ *s*  
**assumes** *atom y* # (*xa*, *va*) **and** *atom ya* # (*xa*, *va*) **and**  
∀ *c*. *atom c* # (*s*, *sa*) → *atom c* # (*y*, *ya*, *s*, *sa*) → (*y* ↔ *c*) · *s* = (*ya* ↔ *c*) · *sa*  
**and** *eqvt-at f* (*s*,*xa*,*va*) **and** *eqvt-at f* (*sa*,*xa*,*va*) **and**  
*atom c* # (*s*, *va*, *xa*, *sa*) **and** *atom c* # (*y*, *ya*, *f* (*s*, *xa*, *va*), *f* (*sa*, *xa*, *va*))  
**shows** (*y* ↔ *c*) · *f* (*s*, *xa*, *va*) = (*ya* ↔ *c*) · *f* (*sa*, *xa*, *va*)  
⟨*proof*⟩

## 2.3 Functions for bit list/vectors

**inductive** *split* :: int ⇒ bit list ⇒ bit list \* bit list ⇒ bool **where**

*split* 0 *xs* ([], *xs*)  
| *split* *m* *xs* (*ys*,*zs*) ⇒ *split* (*m*+1) (*x*#*xs*) ((*x* # *ys*), *zs*)

**equivariance** *split*

**nominal-inductive** *split* ⟨*proof*⟩

**lemma** *split-concat*:

**assumes** *split n v* (*v1*,*v2*)  
**shows** *v* = *append v1 v2*  
⟨*proof*⟩

**lemma** *split-n*:

**assumes** *split n v* (*v1*,*v2*)  
**shows** 0 ≤ *n* ∧ *n* ≤ int (*length v*)  
⟨*proof*⟩

**lemma** *split-length*:

**assumes** *split n v* (*v1*,*v2*)  
**shows** *n* = int (*length v1*)  
⟨*proof*⟩

**lemma** *obtain-split*:

**assumes** 0 ≤ *n* **and** *n* ≤ int (*length bv*)  
**shows** ∃ *bv1 bv2*. *split n bv* (*bv1* , *bv2*)  
⟨*proof*⟩

**end**

## Chapter 3

# Immutable Variable Substitution

Substitution involving immutable variables. We define a class and instances for all of the term forms

### 3.1 Class

```
class has-subst-v = fs +
  fixes subst-v :: 'a::fs ⇒ x ⇒ v ⇒ 'a::fs  (⟨[-::=-]⟩ [1000,50,50] 1000)
  assumes fresh-subst-v-if:  y ‡ (subst-v a x v) ⟷ (atom x ‡ a ∧ y ‡ a) ∨ (y ‡ v ∧ (y ‡ a ∨ y = atom
x))
  and forget-subst-v[simp]:  atom x ‡ a ⟹ subst-v a x v = a
  and subst-v-id[simp]:      subst-v a x (V-var x) = a
  and eqvt[simp,eqvt]:      (p::perm) · (subst-v a x v) = (subst-v (p · a) (p · x) (p · v))
  and flip-subst-v[simp]:   atom x ‡ c ⟹ ((x ↔ z) · c) = c[z::=[x]v]v
  and subst-v-simple-commute[simp]:  atom x ‡ c ⟹ (c[z::=[x]v]v)[x::=b]v = c[z::=b]v
begin
```

**lemma** *subst-v-flip-eq-one*:

```
fixes z1::x and z2::x and x1::x and x2::x
assumes [[atom z1]]lst. c1 = [[atom z2]]lst. c2
  and atom x1 ‡ (z1,z2,c1,c2)
shows (c1[z1::=[x1]v]v) = (c2[z2::=[x1]v]v)
⟨proof⟩
```

**lemma** *subst-v-flip-eq-two*:

```
fixes z1::x and z2::x and x1::x and x2::x
assumes [[atom z1]]lst. c1 = [[atom z2]]lst. c2
shows (c1[z1::=b]v) = (c2[z2::=b]v)
⟨proof⟩
```

**lemma** *subst-v-flip-eq-three*:

```
assumes [[atom z1]]lst. c1 = [[atom z1^]]lst. c1' and atom x ‡ c1 and atom x' ‡ (x,z1,z1', c1, c1^)
shows (x ↔ x^) · (c1[z1::=[x]v]v) = c1'[z1^::=[x^]v]v
⟨proof⟩
```

**end**

## 3.2 Values

### nominal-function

$subst\text{-}vv :: v \Rightarrow x \Rightarrow v \Rightarrow v$  **where**  
 $subst\text{-}vv (V\text{-lit } l) x v = V\text{-lit } l$   
 $| subst\text{-}vv (V\text{-var } y) x v = (if\ x = y\ then\ v\ else\ V\text{-var } y)$   
 $| subst\text{-}vv (V\text{-cons } tyid\ c\ v') x v = V\text{-cons } tyid\ c\ (subst\text{-}vv\ v'\ x\ v)$   
 $| subst\text{-}vv (V\text{-consp } tyid\ c\ b\ v') x v = V\text{-consp } tyid\ c\ b\ (subst\text{-}vv\ v'\ x\ v)$   
 $| subst\text{-}vv (V\text{-pair } v1\ v2) x v = V\text{-pair } (subst\text{-}vv\ v1\ x\ v)\ (subst\text{-}vv\ v2\ x\ v)$   
 $\langle proof \rangle$

**nominal-termination** (*eqvt*)  $\langle proof \rangle$

### abbreviation

$subst\text{-}vv\text{-}abbrev :: v \Rightarrow x \Rightarrow v \Rightarrow v$  ( $\langle [-::=]_{vv} \rangle [1000, 50, 50]$  1000)

**where**

$v[x::=v']_{vv} \equiv subst\text{-}vv\ v\ x\ v'$

**lemma** *fresh-subst-vv-if* [*simp*]:

$j \# t[i::=x]_{vv} = ((atom\ i \# t \wedge j \# t) \vee (j \# x \wedge (j \# t \vee j = atom\ i)))$   
 $\langle proof \rangle$

**lemma** *forget-subst-vv* [*simp*]:  $atom\ a \# tm \implies tm[a::=x]_{vv} = tm$

$\langle proof \rangle$

**lemma** *subst-vv-id* [*simp*]:  $tm[a::=V\text{-var } a]_{vv} = tm$

$\langle proof \rangle$

**lemma** *subst-vv-commute* [*simp*]:

$atom\ j \# tm \implies tm[i::=t]_{vv}[j::=u]_{vv} = tm[i::=t][j::=u]_{vv}$   
 $\langle proof \rangle$

**lemma** *subst-vv-commute-full* [*simp*]:

$atom\ j \# t \implies atom\ i \# u \implies i \neq j \implies tm[i::=t]_{vv}[j::=u]_{vv} = tm[j::=u]_{vv}[i::=t]_{vv}$   
 $\langle proof \rangle$

**lemma** *subst-vv-var-flip* [*simp*]:

**fixes**  $v::v$

**assumes**  $atom\ y \# v$

**shows**  $(y \leftrightarrow x) \cdot v = v[x::=V\text{-var } y]_{vv}$

$\langle proof \rangle$

**instantiation**  $v :: has\text{-}subst\text{-}v$

**begin**

**definition**

$subst\text{-}v = subst\text{-}vv$

**instance**  $\langle proof \rangle$

**end**

### 3.3 Expressions

**nominal-function**  $subst-ev :: e \Rightarrow x \Rightarrow v \Rightarrow e$  **where**

$subst-ev \ ( (AE-val \ v') ) \ x \ v = ( (AE-val \ (subst-vv \ v' \ x \ v)) )$   
 $| \ subst-ev \ ( (AE-app \ f \ v') ) \ x \ v = ( (AE-app \ f \ (subst-vv \ v' \ x \ v)) )$   
 $| \ subst-ev \ ( (AE-appP \ f \ b \ v') ) \ x \ v = ( (AE-appP \ f \ b \ (subst-vv \ v' \ x \ v)) )$   
 $| \ subst-ev \ ( (AE-op \ opp \ v1 \ v2) ) \ x \ v = ( (AE-op \ opp \ (subst-vv \ v1 \ x \ v) \ (subst-vv \ v2 \ x \ v)) )$   
 $| \ subst-ev \ [\#1 \ v]^e \ x \ v = [\#1 \ (subst-vv \ v' \ x \ v)]^e$   
 $| \ subst-ev \ [\#2 \ v]^e \ x \ v = [\#2 \ (subst-vv \ v' \ x \ v)]^e$   
 $| \ subst-ev \ ( (AE-mvar \ u) ) \ x \ v = AE-mvar \ u$   
 $| \ subst-ev \ [ [ v' ] ]^e \ x \ v = [ [ (subst-vv \ v' \ x \ v) ] ]^e$   
 $| \ subst-ev \ ( AE-concat \ v1 \ v2 ) \ x \ v = AE-concat \ (subst-vv \ v1 \ x \ v) \ (subst-vv \ v2 \ x \ v)$   
 $| \ subst-ev \ ( AE-split \ v1 \ v2 ) \ x \ v = AE-split \ (subst-vv \ v1 \ x \ v) \ (subst-vv \ v2 \ x \ v)$   
 $\langle proof \rangle$

**nominal-termination**  $(eqvt)$   $\langle proof \rangle$

**abbreviation**

$subst-ev-abbrev :: e \Rightarrow x \Rightarrow v \Rightarrow e \ (\langle [-::=]_{ev} \rangle [1000,50,50] \ 500)$

**where**

$e[x::=v]_{ev} \equiv subst-ev \ e \ x \ v'$

**lemma**  $size-subst-ev$   $[simp]$ :  $size \ ( \ subst-ev \ A \ i \ x ) = size \ A$

$\langle proof \rangle$

**lemma**  $forget-subst-ev$   $[simp]$ :  $atom \ a \ \# \ A \Longrightarrow \ subst-ev \ A \ a \ x = A$

$\langle proof \rangle$

**lemma**  $subst-ev-id$   $[simp]$ :  $subst-ev \ A \ a \ (V-var \ a) = A$

$\langle proof \rangle$

**lemma**  $fresh-subst-ev-if$   $[simp]$ :

$j \ \# \ (subst-ev \ A \ i \ x) = ((atom \ i \ \# \ A \ \wedge \ j \ \# \ A) \vee (j \ \# \ x \ \wedge \ (j \ \# \ A \ \vee \ j = atom \ i)))$

$\langle proof \rangle$

**lemma**  $subst-ev-commute$   $[simp]$ :

$atom \ j \ \# \ A \Longrightarrow (A[i::=t]_{ev})[j::=u]_{ev} = A[i::=t][j::=u]_{vv} ]_{ev}$

$\langle proof \rangle$

**lemma**  $subst-ev-var-flip$   $[simp]$ :

**fixes**  $e::e$  **and**  $y::x$  **and**  $x::x$

**assumes**  $atom \ y \ \# \ e$

**shows**  $(y \leftrightarrow x) \cdot e = e [x::=V-var \ y]_{ev}$

$\langle proof \rangle$

**lemma**  $subst-ev-flip$ :

**fixes**  $e::e$  **and**  $ea::e$  **and**  $c::x$

**assumes**  $atom \ c \ \# \ (e, \ ea)$  **and**  $atom \ c \ \# \ (x, \ xa, \ e, \ ea)$  **and**  $(x \leftrightarrow c) \cdot e = (xa \leftrightarrow c) \cdot ea$

**shows**  $e[x::=v]_{ev} = ea[xa::=v]_{ev}$

$\langle proof \rangle$

**lemma**  $subst-ev-var$   $[simp]$ :

$(AE-val \ (V-var \ x))[x::=[z]^v]_{ev} = AE-val \ (V-var \ z)$

$\langle proof \rangle$

**instantiation**  $e :: has\text{-}subst\text{-}v$

**begin**

**definition**

$subst\text{-}v = subst\text{-}ev$

**instance**  $\langle proof \rangle$

**end**

**lemma** *subst-ev-commute-full*:

**fixes**  $e::e$  **and**  $w::v$  **and**  $v::v$

**assumes**  $atom\ z \# v$  **and**  $atom\ x \# w$  **and**  $x \neq z$

**shows**  $subst\text{-}ev\ (e[z::=w]_{ev})\ x\ v = subst\text{-}ev\ (e[x::=v]_{ev})\ z\ w$

$\langle proof \rangle$

**lemma** *subst-ev-v-flip1* [*simp*]:

**fixes**  $e::e$

**assumes**  $atom\ z1 \# (z,e)$  **and**  $atom\ z1' \# (z,e)$

**shows**  $(z1 \leftrightarrow z1') \cdot e[z::=v]_{ev} = e[z::=((z1 \leftrightarrow z1') \cdot v)]_{ev}$

$\langle proof \rangle$

### 3.4 Expressions in Constraints

**nominal-function** *subst-cev*  $:: ce \Rightarrow x \Rightarrow v \Rightarrow ce$  **where**

$subst\text{-}cev\ ((CE\text{-}val\ v'))\ x\ v = ((CE\text{-}val\ (subst\text{-}vv\ v'\ x\ v))\ )$

|  $subst\text{-}cev\ ((CE\text{-}op\ opp\ v1\ v2)\ )\ x\ v = ((CE\text{-}op\ opp\ (subst\text{-}cev\ v1\ x\ v)\ (subst\text{-}cev\ v2\ x\ v))\ )$

|  $subst\text{-}cev\ ((CE\text{-}fst\ v'))\ x\ v = CE\text{-}fst\ (subst\text{-}cev\ v'\ x\ v)$

|  $subst\text{-}cev\ ((CE\text{-}snd\ v'))\ x\ v = CE\text{-}snd\ (subst\text{-}cev\ v'\ x\ v)$

|  $subst\text{-}cev\ ((CE\text{-}len\ v'))\ x\ v = CE\text{-}len\ (subst\text{-}cev\ v'\ x\ v)$

|  $subst\text{-}cev\ (CE\text{-}concat\ v1\ v2)\ x\ v = CE\text{-}concat\ (subst\text{-}cev\ v1\ x\ v)\ (subst\text{-}cev\ v2\ x\ v)$

$\langle proof \rangle$

**nominal-termination** (*eqvt*)  $\langle proof \rangle$

**abbreviation**

$subst\text{-}cev\text{-}abbrev :: ce \Rightarrow x \Rightarrow v \Rightarrow ce$  ( $\lambda[-::=]_{cev}$ ) [*1000,50,50*] *500*)

**where**

$e[x::=v]_{cev} \equiv subst\text{-}cev\ e\ x\ v'$

**lemma** *size-subst-cev* [*simp*]:  $size\ (subst\text{-}cev\ A\ i\ x) = size\ A$

$\langle proof \rangle$

**lemma** *forget-subst-cev* [*simp*]:  $atom\ a \# A \Longrightarrow subst\text{-}cev\ A\ a\ x = A$

$\langle proof \rangle$

**lemma** *subst-cev-id* [*simp*]:  $subst\text{-}cev\ A\ a\ (V\text{-}var\ a) = A$

$\langle proof \rangle$

**lemma** *fresh-subst-cev-if* [*simp*]:

$j \# (subst\text{-}cev\ A\ i\ x) = ((atom\ i \# A \wedge j \# A) \vee (j \# x \wedge (j \# A \vee j = atom\ i)))$

$\langle proof \rangle$

**lemma** *subst-cev-commute* [simp]:

$atom\ j \# A \implies (subst\text{-}cev\ (subst\text{-}cev\ A\ i\ t)\ j\ u) = subst\text{-}cev\ A\ i\ (subst\text{-}vv\ t\ j\ u)$

$\langle proof \rangle$

**lemma** *subst-cev-var-flip*[simp]:

**fixes**  $e::ce$  **and**  $y::x$  **and**  $x::x$

**assumes**  $atom\ y \# e$

**shows**  $(y \leftrightarrow x) \cdot e = e [x ::= V\text{-}var\ y]_{cev}$

$\langle proof \rangle$

**lemma** *subst-cev-flip*:

**fixes**  $e::ce$  **and**  $ea::ce$  **and**  $c::x$

**assumes**  $atom\ c \# (e, ea)$  **and**  $atom\ c \# (x, xa, e, ea)$  **and**  $(x \leftrightarrow c) \cdot e = (xa \leftrightarrow c) \cdot ea$

**shows**  $e[x ::= v]_{cev} = ea[xa ::= v]_{cev}$

$\langle proof \rangle$

**lemma** *subst-cev-var*[simp]:

**fixes**  $z::x$  **and**  $x::x$

**shows**  $[[x]^v]^{ce} [x ::= [z]^v]_{cev} = [[z]^v]^{ce}$

$\langle proof \rangle$

**instantiation**  $ce :: has\text{-}subst\text{-}v$

**begin**

**definition**

$subst\text{-}v = subst\text{-}cev$

**instance**  $\langle proof \rangle$

**end**

**lemma** *subst-cev-commute-full*:

**fixes**  $e::ce$  **and**  $w::v$  **and**  $v::v$

**assumes**  $atom\ z \# v$  **and**  $atom\ x \# w$  **and**  $x \neq z$

**shows**  $subst\text{-}cev\ (e[z ::= w]_{cev})\ x\ v = subst\text{-}cev\ (e[x ::= v]_{cev})\ z\ w$

$\langle proof \rangle$

**lemma** *subst-cev-v-flip1*[simp]:

**fixes**  $e::ce$

**assumes**  $atom\ z1 \# (z, e)$  **and**  $atom\ z1' \# (z, e)$

**shows**  $(z1 \leftrightarrow z1') \cdot e[z ::= v]_{cev} = e[z ::= ((z1 \leftrightarrow z1') \cdot v)]_{cev}$

$\langle proof \rangle$

## 3.5 Constraints

**nominal-function**  $subst\text{-}cv :: c \Rightarrow x \Rightarrow v \Rightarrow c$  **where**

$subst\text{-}cv\ (C\text{-}true)\ x\ v = C\text{-}true$

|  $subst\text{-}cv\ (C\text{-}false)\ x\ v = C\text{-}false$

|  $subst\text{-}cv\ (C\text{-}conj\ c1\ c2)\ x\ v = C\text{-}conj\ (subst\text{-}cv\ c1\ x\ v)\ (subst\text{-}cv\ c2\ x\ v)$

$| \text{subst-cv } (C\text{-disj } c1 \ c2) \ x \ v = C\text{-disj } (\text{subst-cv } c1 \ x \ v) \ (\text{subst-cv } c2 \ x \ v)$   
 $| \text{subst-cv } (C\text{-imp } c1 \ c2) \ x \ v = C\text{-imp } (\text{subst-cv } c1 \ x \ v) \ (\text{subst-cv } c2 \ x \ v)$   
 $| \text{subst-cv } (e1 == e2) \ x \ v = ((\text{subst-cev } e1 \ x \ v) == (\text{subst-cev } e2 \ x \ v))$   
 $| \text{subst-cv } (C\text{-not } c) \ x \ v = C\text{-not } (\text{subst-cv } c \ x \ v)$   
 $\langle \text{proof} \rangle$

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

**abbreviation**

$\text{subst-cv-abbrev} :: c \Rightarrow x \Rightarrow v \Rightarrow c \langle \text{[-::=]}_{cv} \rangle [1000, 50, 50] \ 1000$

**where**

$c[x::=v]_{cv} \equiv \text{subst-cv } c \ x \ v'$

**lemma** *size-subst-cv* [*simp*]:  $\text{size } (\text{subst-cv } A \ i \ x) = \text{size } A$   
 $\langle \text{proof} \rangle$

**lemma** *forget-subst-cv* [*simp*]:  $\text{atom } a \ \# \ A \Longrightarrow \text{subst-cv } A \ a \ x = A$   
 $\langle \text{proof} \rangle$

**lemma** *subst-cv-id* [*simp*]:  $\text{subst-cv } A \ a \ (V\text{-var } a) = A$   
 $\langle \text{proof} \rangle$

**lemma** *fresh-subst-cv-if* [*simp*]:  
 $j \ \# \ (\text{subst-cv } A \ i \ x) \longleftrightarrow (\text{atom } i \ \# \ A \wedge j \ \# \ A) \vee (j \ \# \ x \wedge (j \ \# \ A \vee j = \text{atom } i))$   
 $\langle \text{proof} \rangle$

**lemma** *subst-cv-commute* [*simp*]:  
 $\text{atom } j \ \# \ A \Longrightarrow (\text{subst-cv } (\text{subst-cv } A \ i \ t) \ j \ u) = \text{subst-cv } A \ i \ (\text{subst-vv } t \ j \ u)$   
 $\langle \text{proof} \rangle$

**lemma** *let-s-size* [*simp*]:  $\text{size } s \leq \text{size } (AS\text{-let } x \ e \ s)$   
 $\langle \text{proof} \rangle$

**lemma** *subst-cv-var-flip*[*simp*]:  
**fixes**  $c::c$   
**assumes**  $\text{atom } y \ \# \ c$   
**shows**  $(y \leftrightarrow x) \cdot c = c[x::=V\text{-var } y]_{cv}$   
 $\langle \text{proof} \rangle$

**instantiation**  $c :: \text{has-subst-v}$

**begin**

**definition**

$\text{subst-v} = \text{subst-cv}$

**instance**  $\langle \text{proof} \rangle$

**end**

**lemma** *subst-cv-var-flip1*[*simp*]:  
**fixes**  $c::c$   
**assumes**  $\text{atom } y \ \# \ c$   
**shows**  $(x \leftrightarrow y) \cdot c = c[x::=V\text{-var } y]_{cv}$

$\langle proof \rangle$

**lemma** *subst-cv-v-flip3*[simp]:

**fixes**  $c::c$

**assumes**  $atom\ z1 \# c$  **and**  $atom\ z1' \# c$

**shows**  $(z1 \leftrightarrow z1') \cdot c[z::=[z1]^v]_{cv} = c[z::=[z1']^v]_{cv}$

$\langle proof \rangle$

**lemma** *subst-cv-v-flip*[simp]:

**fixes**  $c::c$

**assumes**  $atom\ x \# c$

**shows**  $((x \leftrightarrow z) \cdot c)[x::=v]_{cv} = c[z::=v]_{cv}$

$\langle proof \rangle$

**lemma** *subst-cv-commute-full*:

**fixes**  $c::c$

**assumes**  $atom\ z \# v$  **and**  $atom\ x \# w$  **and**  $x \neq z$

**shows**  $(c[z::=w]_{cv})[x::=v]_{cv} = (c[x::=v]_{cv})[z::=w]_{cv}$

$\langle proof \rangle$

**lemma** *subst-cv-eq*[simp]:

**assumes**  $atom\ z1 \# e1$

**shows**  $(CE\text{-val}\ (V\text{-var}\ z1) == e1)[z1::=[x]^v]_{cv} = (CE\text{-val}\ (V\text{-var}\ x) == e1) \text{ (is ?A = ?B)}$

$\langle proof \rangle$

## 3.6 Variable Context

The idea of this substitution is to remove  $x$  from the context. We really want to add the condition that  $x$  is fresh in  $v$  but this causes problems with proofs.

**nominal-function** *subst-gv*  $:: \Gamma \Rightarrow x \Rightarrow v \Rightarrow \Gamma$  **where**

*subst-gv*  $GNil\ x\ v = GNil$

| *subst-gv*  $((y,b,c) \#_{\Gamma} \Gamma)\ x\ v = (if\ x = y\ then\ \Gamma\ else\ ((y,b,c[x::=v]_{cv}) \#_{\Gamma} (subst-gv\ \Gamma\ x\ v)))$

$\langle proof \rangle$

**nominal-termination** (*eqvt*)  $\langle proof \rangle$

**abbreviation**

*subst-gv-abbrev*  $:: \Gamma \Rightarrow x \Rightarrow v \Rightarrow \Gamma\ (\langle \cdot [::=]_{\Gamma v} \rangle [1000,50,50]\ 1000)$

**where**

$g[x::=v]_{\Gamma v} \equiv subst-gv\ g\ x\ v$

**lemma** *size-subst-gv* [simp]:  $size\ (subst-gv\ G\ i\ x) \leq size\ G$

$\langle proof \rangle$

**lemma** *forget-subst-gv* [simp]:  $atom\ a \# G \Longrightarrow subst-gv\ G\ a\ x = G$

$\langle proof \rangle$

**lemma** *fresh-subst-gv*:  $atom\ a \# G \Longrightarrow atom\ a \# v \Longrightarrow atom\ a \# subst-gv\ G\ x\ v$

$\langle proof \rangle$

**lemma** *subst-gv-flip*:

**fixes**  $x::x$  **and**  $xa::x$  **and**  $z::x$  **and**  $c::c$  **and**  $b::b$  **and**  $\Gamma::\Gamma$

**assumes**  $\text{atom } xa \# ((x, b, c[z::=[x]^v]_{cv}) \#_{\Gamma} \Gamma)$  **and**  $\text{atom } xa \# \Gamma$  **and**  $\text{atom } x \# \Gamma$  **and**  $\text{atom } x \# (z,$   
 $c)$  **and**  $\text{atom } xa \# (z, c)$   
**shows**  $(x \leftrightarrow xa) \cdot ((x, b, c[z::=[x]^v]_{cv}) \#_{\Gamma} \Gamma) = (xa, b, c[z::=V\text{-var } xa]_{cv}) \#_{\Gamma} \Gamma$   
 $\langle \text{proof} \rangle$

### 3.7 Types

**nominal-function**  $\text{subst-tv} :: \tau \Rightarrow x \Rightarrow v \Rightarrow \tau$  **where**  
 $\text{atom } z \# (x, v) \Longrightarrow \text{subst-tv } \{ z : b \mid c \} x v = \{ z : b \mid c[x::=v]_{cv} \}$   
 $\langle \text{proof} \rangle$

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

**abbreviation**

$\text{subst-tv-abbrev} :: \tau \Rightarrow x \Rightarrow v \Rightarrow \tau \langle \text{[-::=]}_{\tau v} \rangle [1000, 50, 50] 1000$

**where**

$t[x::=v]_{\tau v} \equiv \text{subst-tv } t x v$

**lemma** *size-subst-tv* [*simp*]:  $\text{size } (\text{subst-tv } A i x) = \text{size } A$   
 $\langle \text{proof} \rangle$

**lemma** *forget-subst-tv* [*simp*]:  $\text{atom } a \# A \Longrightarrow \text{subst-tv } A a x = A$   
 $\langle \text{proof} \rangle$

**lemma** *subst-tv-id* [*simp*]:  $\text{subst-tv } A a (V\text{-var } a) = A$   
 $\langle \text{proof} \rangle$

**lemma** *fresh-subst-tv-if* [*simp*]:  
 $j \# (\text{subst-tv } A i x) \longleftrightarrow (\text{atom } i \# A \wedge j \# A) \vee (j \# x \wedge (j \# A \vee j = \text{atom } i))$   
 $\langle \text{proof} \rangle$

**lemma** *subst-tv-commute* [*simp*]:  
 $\text{atom } y \# \tau \Longrightarrow (\tau[x::=t]_{\tau v})[y::=v]_{\tau v} = \tau[x::=t[y::=v]_{vv}]_{\tau v}$   
 $\langle \text{proof} \rangle$

**lemma** *subst-tv-var-flip* [*simp*]:  
**fixes**  $x::x$  **and**  $xa::x$  **and**  $\tau::\tau$   
**assumes**  $\text{atom } xa \# \tau$   
**shows**  $(x \leftrightarrow xa) \cdot \tau = \tau[x::=V\text{-var } xa]_{\tau v}$   
 $\langle \text{proof} \rangle$

**instantiation**  $\tau :: \text{has-subst-v}$

**begin**

**definition**

$\text{subst-v} = \text{subst-tv}$

**instance**  $\langle \text{proof} \rangle$

**end**

**lemma** *subst-tv-commute-full*:

**fixes**  $c::\tau$   
**assumes**  $\text{atom } z \# v$  **and**  $\text{atom } x \# w$  **and**  $x \neq z$   
**shows**  $(c[z::=w]_{\tau v})[x::=v]_{\tau v} = (c[x::=v]_{\tau v})[z::=w]_{\tau v}$   
 $\langle \text{proof} \rangle$

**lemma** *type-eq-subst-eq*:  
**fixes**  $v::v$  **and**  $c1::c$   
**assumes**  $\{ z1 : b1 \mid c1 \} = \{ z2 : b2 \mid c2 \}$   
**shows**  $c1[z1::=v]_{cv} = c2[z2::=v]_{cv}$   
 $\langle \text{proof} \rangle$

Extract constraint from a type. We cannot just project out the constraint as this would mean alpha-equivalent types give different answers

**nominal-function**  $c\text{-of} :: \tau \Rightarrow x \Rightarrow c$  **where**  
 $\text{atom } z \# x \Longrightarrow c\text{-of } (T\text{-refined-type } z \ b \ c) \ x = c[z::=[x]^v]_{cv}$   
 $\langle \text{proof} \rangle$

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

**lemma** *c-of-eq*:  
**shows**  $c\text{-of } \{ x : b \mid c \} \ x = c$   
 $\langle \text{proof} \rangle$

**lemma** *obtain-fresh-z-c-of*:  
**fixes**  $t::'b::fs$   
**obtains**  $z$  **where**  $\text{atom } z \# t \wedge \tau = \{ z : b\text{-of } \tau \mid c\text{-of } \tau \ z \}$   
 $\langle \text{proof} \rangle$

**lemma** *c-of-fresh*:  
**fixes**  $x::x$   
**assumes**  $\text{atom } x \# (t, z)$   
**shows**  $\text{atom } x \# c\text{-of } t \ z$   
 $\langle \text{proof} \rangle$

**lemma** *c-of-switch*:  
**fixes**  $z::x$   
**assumes**  $\text{atom } z \# t$   
**shows**  $(c\text{-of } t \ z)[z::=V\text{-var } x]_{cv} = c\text{-of } t \ x$   
 $\langle \text{proof} \rangle$

**lemma** *type-eq-subst-eq1*:  
**fixes**  $v::v$  **and**  $c1::c$   
**assumes**  $\{ z1 : b1 \mid c1 \} = (\{ z2 : b2 \mid c2 \})$  **and**  $\text{atom } z1 \# c2$   
**shows**  $c1[z1::=v]_{cv} = c2[z2::=v]_{cv}$  **and**  $b1=b2$  **and**  $c1 = (z1 \leftrightarrow z2) \cdot c2$   
 $\langle \text{proof} \rangle$

**lemma** *type-eq-subst-eq2*:  
**fixes**  $v::v$  **and**  $c1::c$   
**assumes**  $\{ z1 : b1 \mid c1 \} = (\{ z2 : b2 \mid c2 \})$   
**shows**  $c1[z1::=v]_{cv} = c2[z2::=v]_{cv}$  **and**  $b1=b2$  **and**  $[[\text{atom } z1]]\text{lst. } c1 = [[\text{atom } z2]]\text{lst. } c2$   
 $\langle \text{proof} \rangle$

**lemma** *type-eq-subst-eq3*:

**fixes**  $v::v$  **and**  $c1::c$

**assumes**  $\{ \{ z1 : b1 \mid c1 \} = \{ \{ z2 : b2 \mid c2 \} \}$  **and**  $\text{atom } z1 \# c2$

**shows**  $c1 = c2[z2 ::= V\text{-var } z1]_{cv}$  **and**  $b1 = b2$

$\langle \text{proof} \rangle$

**lemma** *type-eq-flip*:

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

**shows**  $\{ \{ z : b \mid c \} = \{ \{ x : b \mid (x \leftrightarrow z) \cdot c \} \}$

$\langle \text{proof} \rangle$

**lemma** *c-of-true*:

*c-of*  $\{ \{ z' : B\text{-bool} \mid TRUE \} \} x = C\text{-true}$

$\langle \text{proof} \rangle$

**lemma** *type-eq-subst*:

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

**shows**  $\{ \{ z : b \mid c \} = \{ \{ x : b \mid c[z ::= [x]^v]_{cv} \} \}$

$\langle \text{proof} \rangle$

**lemma** *type-e-subst-fresh*:

**fixes**  $x::x$  **and**  $z::x$

**assumes**  $\text{atom } z \# (x, v)$  **and**  $\text{atom } x \# e$

**shows**  $\{ \{ z : b \mid CE\text{-val } (V\text{-var } z) == e \} [x ::= v]_{\tau v} = \{ \{ z : b \mid CE\text{-val } (V\text{-var } z) == e \} \}$

$\langle \text{proof} \rangle$

**lemma** *type-v-subst-fresh*:

**fixes**  $x::x$  **and**  $z::x$

**assumes**  $\text{atom } z \# (x, v)$  **and**  $\text{atom } x \# v'$

**shows**  $\{ \{ z : b \mid CE\text{-val } (V\text{-var } z) == CE\text{-val } v' \} [x ::= v]_{\tau v} = \{ \{ z : b \mid CE\text{-val } (V\text{-var } z) == CE\text{-val } v' \} \}$

$\langle \text{proof} \rangle$

**lemma** *subst-tbase-eq*:

*b-of*  $\tau = \text{b-of } \tau[x ::= v]_{\tau v}$

$\langle \text{proof} \rangle$

**lemma** *subst-tv-if*:

**assumes**  $\text{atom } z1 \# (x, v)$  **and**  $\text{atom } z' \# (x, v)$

**shows**  $\{ \{ z1 : b \mid CE\text{-val } (v'[x ::= v]_{vv}) == CE\text{-val } (V\text{-lit } l) \text{ IMP } (c'[x ::= v]_{cv})[z' ::= [z1]^v]_{cv} \} = \{ \{ z1 : b \mid CE\text{-val } v' == CE\text{-val } (V\text{-lit } l) \text{ IMP } c'[z' ::= [z1]^v]_{cv} \} [x ::= v]_{\tau v} \}$

$\langle \text{proof} \rangle$

**lemma** *subst-tv-tid*:

**assumes**  $\text{atom } za \# (x, v)$

**shows**  $\{ \{ za : B\text{-id } tid \mid TRUE \} = \{ \{ za : B\text{-id } tid \mid TRUE \} [x ::= v]_{\tau v} \}$

$\langle \text{proof} \rangle$

**lemma** *b-of-subst*:

*b-of*  $(\tau[x ::= v]_{\tau v}) = \text{b-of } \tau$

$\langle \text{proof} \rangle$

**lemma** *subst-tv-flip*:

**assumes**  $\tau'[x::=v]_{\tau v} = \tau$  **and** *atom*  $x \# (v, \tau)$  **and** *atom*  $x' \# (v, \tau)$   
**shows**  $((x' \leftrightarrow x) \cdot \tau')[x'::=v]_{\tau v} = \tau$

*<proof>*

**lemma** *subst-cv-true*:

$\{ z : B\text{-id tid} \mid \text{TRUE} \} = \{ z : B\text{-id tid} \mid \text{TRUE} \}[x::=v]_{\tau v}$

*<proof>*

**lemma** *t-eg-supp*:

**assumes**  $(\{ z : b \mid c \}) = (\{ z1 : b1 \mid c1 \})$   
**shows**  $\text{supp } c - \{ \text{atom } z \} = \text{supp } c1 - \{ \text{atom } z1 \}$

*<proof>*

**lemma** *fresh-t-eg*:

**fixes**  $x::x$

**assumes**  $(\{ z : b \mid c \}) = (\{ zz : b \mid cc \})$  **and** *atom*  $x \# c$  **and**  $x \neq zz$   
**shows** *atom*  $x \# cc$

*<proof>*

## 3.8 Mutable Variable Context

**nominal-function** *subst-dv*  $:: \Delta \Rightarrow x \Rightarrow v \Rightarrow \Delta$  **where**

*subst-dv*  $\text{DNil } x \ v = \text{DNil}$

$\mid \text{subst-dv } ((u, t) \#_{\Delta} \Delta) \ x \ v = ((u, t[x::=v]_{\tau v}) \#_{\Delta} (\text{subst-dv } \Delta \ x \ v))$

*<proof>*

**nominal-termination** (*eqvt*) *<proof>*

**abbreviation**

*subst-dv-abbrev*  $:: \Delta \Rightarrow x \Rightarrow v \Rightarrow \Delta \ (\langle \cdot [-::=]_{\Delta v} \rangle [1000, 50, 50] \ 1000)$

**where**

$\Delta[x::=v]_{\Delta v} \equiv \text{subst-dv } \Delta \ x \ v$

**nominal-function** *dmap*  $:: (u * \tau \Rightarrow u * \tau) \Rightarrow \Delta \Rightarrow \Delta$  **where**

*dmap*  $f \ \text{DNil} = \text{DNil}$

$\mid \text{dmap } f \ ((u, t) \#_{\Delta} \Delta) = (f \ (u, t) \#_{\Delta} (\text{dmap } f \ \Delta))$

*<proof>*

**nominal-termination** (*eqvt*) *<proof>*

**lemma** *subst-dv-iff*:

$\Delta[x::=v]_{\Delta v} = \text{dmap } (\lambda(u, t). (u, t[x::=v]_{\tau v})) \ \Delta$

*<proof>*

**lemma** *size-subst-dv [simp]*:  $\text{size } (\text{subst-dv } G \ i \ x) \leq \text{size } G$

*<proof>*

**lemma** *forget-subst-dv [simp]*:  $\text{atom } a \ \# \ G \Longrightarrow \text{subst-dv } G \ a \ x = G$

*<proof>*

**lemma** *subst-dv-member*:

**assumes**  $(u, \tau) \in \text{setD } \Delta$

**shows**  $(u, \tau[x::=v]_{\tau v}) \in \text{setD } (\Delta[x::=v]_{\Delta v})$   
 $\langle \text{proof} \rangle$

**lemma** *fresh-subst-dv*:

**fixes**  $x::x$   
**assumes**  $\text{atom } xa \# \Delta$  **and**  $\text{atom } xa \# v$   
**shows**  $\text{atom } xa \# \Delta[x::=v]_{\Delta v}$   
 $\langle \text{proof} \rangle$

**lemma** *fresh-subst-dv-if*:

**fixes**  $j::\text{atom}$  **and**  $i::x$  **and**  $x::v$  **and**  $t::\Delta$   
**assumes**  $j \# t \wedge j \# x$   
**shows**  $(j \# \text{subst-dv } t \ i \ x)$   
 $\langle \text{proof} \rangle$

### 3.9 Statements

Using ideas from proofs at top of AFP/Launchbury/Substitution.thy. Subproofs borrowed from there; hence the apply style proofs.

**nominal-function** (*default case-sum*  $(\lambda x. \text{Inl undefined})$  (*case-sum*  $(\lambda x. \text{Inl undefined})$   $(\lambda x. \text{Inr undefined})$ ))

$\text{subst-sv} :: s \Rightarrow x \Rightarrow v \Rightarrow s$

**and**  $\text{subst-branchv} :: \text{branch-s} \Rightarrow x \Rightarrow v \Rightarrow \text{branch-s}$

**and**  $\text{subst-branchlv} :: \text{branch-list} \Rightarrow x \Rightarrow v \Rightarrow \text{branch-list}$  **where**

$\text{subst-sv } ( (\text{AS-val } v') ) \ x \ v = (\text{AS-val } (\text{subst-vv } v' \ x \ v ))$

|  $\text{atom } y \# (x, v) \Longrightarrow \text{subst-sv } (\text{AS-let } y \ e \ s) \ x \ v = (\text{AS-let } y \ (e[x::=v]_{ev}) \ (\text{subst-sv } s \ x \ v ))$

|  $\text{atom } y \# (x, v) \Longrightarrow \text{subst-sv } (\text{AS-let2 } y \ t \ s1 \ s2) \ x \ v = (\text{AS-let2 } y \ (t[x::=v]_{\tau v}) \ (\text{subst-sv } s1 \ x \ v) \ (\text{subst-sv } s2 \ x \ v ))$

|  $\text{subst-sv } (\text{AS-match } v' \ cs) \ x \ v = \text{AS-match } (v'[x::=v]_{vv}) \ (\text{subst-branchlv } cs \ x \ v)$

|  $\text{subst-sv } (\text{AS-assign } y \ v') \ x \ v = \text{AS-assign } y \ (\text{subst-vv } v' \ x \ v)$

|  $\text{subst-sv } ( (\text{AS-if } v' \ s1 \ s2) ) \ x \ v = (\text{AS-if } (\text{subst-vv } v' \ x \ v) \ (\text{subst-sv } s1 \ x \ v) \ (\text{subst-sv } s2 \ x \ v) )$

|  $\text{atom } u \# (x, v) \Longrightarrow \text{subst-sv } (\text{AS-var } u \ \tau \ v' \ s) \ x \ v = \text{AS-var } u \ (\text{subst-tv } \tau \ x \ v) \ (\text{subst-vv } v' \ x \ v) \ (\text{subst-sv } s \ x \ v)$

|  $\text{subst-sv } (\text{AS-while } s1 \ s2) \ x \ v = \text{AS-while } (\text{subst-sv } s1 \ x \ v) \ (\text{subst-sv } s2 \ x \ v)$

|  $\text{subst-sv } (\text{AS-seq } s1 \ s2) \ x \ v = \text{AS-seq } (\text{subst-sv } s1 \ x \ v) \ (\text{subst-sv } s2 \ x \ v)$

|  $\text{subst-sv } (\text{AS-assert } c \ s) \ x \ v = \text{AS-assert } (\text{subst-cv } c \ x \ v) \ (\text{subst-sv } s \ x \ v)$

|  $\text{atom } x1 \# (x, v) \Longrightarrow \text{subst-branchv } (\text{AS-branch } dc \ x1 \ s1) \ x \ v = \text{AS-branch } dc \ x1 \ (\text{subst-sv } s1 \ x \ v)$

|  $\text{subst-branchlv } (\text{AS-final } cs) \ x \ v = \text{AS-final } (\text{subst-branchv } cs \ x \ v)$

|  $\text{subst-branchlv } (\text{AS-cons } cs \ css) \ x \ v = \text{AS-cons } (\text{subst-branchv } cs \ x \ v) \ (\text{subst-branchlv } css \ x \ v)$

$\langle \text{proof} \rangle$

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

**abbreviation**

$\text{subst-sv-abbrev} :: s \Rightarrow x \Rightarrow v \Rightarrow s \ (\langle [-::=]_{sv} \rangle [1000, 50, 50] \ 1000)$

**where**

$s[x::=v]_{sv} \equiv \text{subst-sv } s \ x \ v$

**abbreviation**

$\text{subst-branchv-abbrev} :: \text{branch-s} \Rightarrow x \Rightarrow v \Rightarrow \text{branch-s} \ (\langle [-::=]_{sv} \rangle [1000, 50, 50] \ 1000)$

**where**

$s[x::=v]_{sv} \equiv \text{subst-branchv } s \ x \ v$

**lemma** *size-subst-sv* [simp]:  $\text{size } (\text{subst-sv } A \ i \ x) = \text{size } A$  **and**  $\text{size } (\text{subst-branchv } B \ i \ x) = \text{size } B$   
**and**  $\text{size } (\text{subst-branchlv } C \ i \ x) = \text{size } C$   
 ⟨proof⟩

**lemma** *forget-subst-sv* [simp]: **shows**  $\text{atom } a \ \# \ A \implies \text{subst-sv } A \ a \ x = A$  **and**  $\text{atom } a \ \# \ B \implies$   
 $\text{subst-branchv } B \ a \ x = B$  **and**  $\text{atom } a \ \# \ C \implies \text{subst-branchlv } C \ a \ x = C$   
 ⟨proof⟩

**lemma** *subst-sv-id* [simp]:  $\text{subst-sv } A \ a \ (V\text{-var } a) = A$  **and**  $\text{subst-branchv } B \ a \ (V\text{-var } a) = B$  **and**  
 $\text{subst-branchlv } C \ a \ (V\text{-var } a) = C$   
 ⟨proof⟩

**lemma** *fresh-subst-sv-if-rl*:

**shows**

$(\text{atom } x \ \# \ s \wedge j \ \# \ s) \vee (j \ \# \ v \wedge (j \ \# \ s \vee j = \text{atom } x)) \implies j \ \# \ (\text{subst-sv } s \ x \ v)$  **and**  
 $(\text{atom } x \ \# \ cs \wedge j \ \# \ cs) \vee (j \ \# \ v \wedge (j \ \# \ cs \vee j = \text{atom } x)) \implies j \ \# \ (\text{subst-branchv } cs \ x \ v)$  **and**  
 $(\text{atom } x \ \# \ css \wedge j \ \# \ css) \vee (j \ \# \ v \wedge (j \ \# \ css \vee j = \text{atom } x)) \implies j \ \# \ (\text{subst-branchlv } css \ x \ v)$   
 ⟨proof⟩

**lemma** *fresh-subst-sv-if-lr*:

**shows**

$j \ \# \ (\text{subst-sv } s \ x \ v) \implies (\text{atom } x \ \# \ s \wedge j \ \# \ s) \vee (j \ \# \ v \wedge (j \ \# \ s \vee j = \text{atom } x))$  **and**  
 $j \ \# \ (\text{subst-branchv } cs \ x \ v) \implies (\text{atom } x \ \# \ cs \wedge j \ \# \ cs) \vee (j \ \# \ v \wedge (j \ \# \ cs \vee j = \text{atom } x))$  **and**  
 $j \ \# \ (\text{subst-branchlv } css \ x \ v) \implies (\text{atom } x \ \# \ css \wedge j \ \# \ css) \vee (j \ \# \ v \wedge (j \ \# \ css \vee j = \text{atom } x))$   
 ⟨proof⟩

**lemma** *fresh-subst-sv-if*[simp]:

**fixes**  $x::x$  **and**  $v::v$

**shows**  $j \ \# \ (\text{subst-sv } s \ x \ v) \longleftrightarrow (\text{atom } x \ \# \ s \wedge j \ \# \ s) \vee (j \ \# \ v \wedge (j \ \# \ s \vee j = \text{atom } x))$  **and**  
 $j \ \# \ (\text{subst-branchv } cs \ x \ v) \longleftrightarrow (\text{atom } x \ \# \ cs \wedge j \ \# \ cs) \vee (j \ \# \ v \wedge (j \ \# \ cs \vee j = \text{atom } x))$   
 ⟨proof⟩

**lemma** *subst-sv-commute* [simp]:

**fixes**  $A::s$  **and**  $t::v$  **and**  $j::x$  **and**  $i::x$

**shows**  $\text{atom } j \ \# \ A \implies (\text{subst-sv } (\text{subst-sv } A \ i \ t) \ j \ u) = \text{subst-sv } A \ i \ (\text{subst-vv } t \ j \ u)$  **and**  
 $\text{atom } j \ \# \ B \implies (\text{subst-branchv } (\text{subst-branchv } B \ i \ t) \ j \ u) = \text{subst-branchv } B \ i \ (\text{subst-vv } t \ j \ u)$  **and**  
 $\text{atom } j \ \# \ C \implies (\text{subst-branchlv } (\text{subst-branchlv } C \ i \ t) \ j \ u) = \text{subst-branchlv } C \ i \ (\text{subst-vv } t \ j \ u)$   
 ⟨proof⟩

**lemma** *c-eq-perm*:

**assumes**  $(\text{atom } z) \iff (\text{atom } z')$  **and**  $c = c'$  **and**  $\text{atom } z' \ \# \ c$   
**shows**  $\llbracket z : b \mid c \rrbracket = \llbracket z' : b \mid c' \rrbracket$   
 ⟨proof⟩

**lemma** *subst-sv-flip*:

**fixes**  $s::s$  **and**  $sa::s$  **and**  $v'::v$

**assumes**  $\text{atom } c \ \# \ (s, sa)$  **and**  $\text{atom } c \ \# \ (v', x, xa, s, sa)$   $\text{atom } x \ \# \ v'$  **and**  $\text{atom } xa \ \# \ v'$  **and**  $(x \leftrightarrow c) \cdot$   
 $s = (xa \leftrightarrow c) \cdot sa$   
**shows**  $s[x::=v']_{sv} = sa[xa::=v']_{sv}$   
 ⟨proof⟩

**lemma** *if-type-eq*:

**fixes**  $\Gamma::\Gamma$  **and**  $v::v$  **and**  $z1::x$   
**assumes**  $atom\ z1' \# (v, ca, (x, b, c) \#_{\Gamma} \Gamma, (CE-val\ v == CE-val\ (V-lit\ ll)\ IMP\ ca[za::=[z1]^v]_{cv}$   
 $)$  **and**  $atom\ z1 \# v$   
**and**  $atom\ z1 \# (za, ca)$  **and**  $atom\ z1' \# (za, ca)$   
**shows**  $(\{ z1' : ba \mid CE-val\ v == CE-val\ (V-lit\ ll)\ IMP\ ca[za::=[z1']^v]_{cv} \}) = \{ z1 : ba \mid CE-val\ v == CE-val\ (V-lit\ ll)\ IMP\ ca[za::=[z1]^v]_{cv} \}$   
 $\langle proof \rangle$

**lemma** *subst-sv-var-flip*:

**fixes**  $x::x$  **and**  $s::s$  **and**  $z::x$   
**shows**  $atom\ x \# s \implies ((x \leftrightarrow z) \cdot s) = s[z::=[x]^v]_{sv}$  **and**  
 $atom\ x \# cs \implies ((x \leftrightarrow z) \cdot cs) = subst-branchv\ cs\ z\ [x]^v$  **and**  
 $atom\ x \# css \implies ((x \leftrightarrow z) \cdot css) = subst-branchlv\ css\ z\ [x]^v$   
 $\langle proof \rangle$

**instantiation**  $s :: has-subst-v$

**begin**

**definition**

$subst-v = subst-sv$

**instance**  $\langle proof \rangle$

**end**

### 3.10 Type Definition

**nominal-function**  $subst-ft-v :: fun-tyt \Rightarrow x \Rightarrow v \Rightarrow fun-tyt$  **where**

$atom\ z \# (x, v) \implies subst-ft-v\ (AF-fun-tyt\ z\ b\ c\ t\ (s::s))\ x\ v = AF-fun-tyt\ z\ b\ c\ [x::=v]_{cv}\ t\ [x::=v]_{\tau v}$   
 $s[x::=v]_{sv}$   
 $\langle proof \rangle$

**nominal-termination**  $(eqvt)\ \langle proof \rangle$

**nominal-function**  $subst-ftq-v :: fun-tyt-q \Rightarrow x \Rightarrow v \Rightarrow fun-tyt-q$  **where**

$atom\ bv \# (x, v) \implies subst-ftq-v\ (AF-fun-tyt-some\ bv\ ft)\ x\ v = (AF-fun-tyt-some\ bv\ (subst-ft-v\ ft\ x\ v))$   
 $\mid\ subst-ftq-v\ (AF-fun-tyt-none\ ft)\ x\ v = (AF-fun-tyt-none\ (subst-ft-v\ ft\ x\ v))$   
 $\langle proof \rangle$

**nominal-termination**  $(eqvt)\ \langle proof \rangle$

**lemma**  $size-subst-ft[simp]$ :  $size\ (subst-ft-v\ A\ x\ v) = size\ A$

$\langle proof \rangle$

**lemma**  $forget-subst-ft\ [simp]$ : **shows**  $atom\ x \# A \implies subst-ft-v\ A\ x\ a = A$

$\langle proof \rangle$

**lemma**  $subst-ft-id\ [simp]$ :  $subst-ft-v\ A\ a\ (V-var\ a) = A$

$\langle proof \rangle$

**instantiation**  $fun-tyt :: has-subst-v$

**begin**

**definition**

$subst-v = subst-ft-v$

**instance**  $\langle proof \rangle$

**end**

**instantiation**  $fun-typ-q :: has-subst-v$

**begin**

**definition**

$subst-v = subst-ftq-v$

**instance**  $\langle proof \rangle$

**end**

### 3.11 Variable Context

**lemma**  $subst-dv-fst-eq$ :

$fst \text{ ' } setD (\Delta[x::=v]_{\Delta v}) = fst \text{ ' } setD \Delta$   
 $\langle proof \rangle$

**lemma**  $subst-gv-member-iff$ :

**fixes**  $x'::x$  **and**  $x::x$  **and**  $v::v$  **and**  $c'::c$   
**assumes**  $(x', b', c') \in toSet \Gamma$  **and**  $atom\ x \notin atom-dom\ \Gamma$   
**shows**  $(x', b', c'[x::=v]_{cv}) \in toSet\ \Gamma[x::=v]_{\Gamma v}$   
 $\langle proof \rangle$

**lemma**  $fresh-subst-gv-if$ :

**fixes**  $j::atom$  **and**  $i::x$  **and**  $x::v$  **and**  $t::\Gamma$   
**assumes**  $j \# t \wedge j \# x$   
**shows**  $(j \# subst-gv\ t\ i\ x)$   
 $\langle proof \rangle$

### 3.12 Lookup

**lemma**  $set-GConsD$ :  $y \in toSet\ (x \#_{\Gamma}\ xs) \implies y=x \vee y \in toSet\ xs$

$\langle proof \rangle$

**lemma**  $subst-g-assoc-cons$ :

**assumes**  $x \neq x'$   
**shows**  $((x', b', c') \#_{\Gamma}\ \Gamma'[x::=v]_{\Gamma v} \ @\ G) = ((x', b', c'[x::=v]_{cv}) \#_{\Gamma}\ ((\Gamma'[x::=v]_{\Gamma v}) \ @\ G))$   
 $\langle proof \rangle$

**end**

## Chapter 4

# Basic Type Variable Substitution

### 4.1 Class

```
class has-subst-b = fs +
  fixes subst-b :: 'a::fs ⇒ bv ⇒ b ⇒ 'a::fs (⟦-⟦[::=]⟧b) [1000,50,50] 1000)

assumes fresh-subst-if: j # (t[i::=x]b) ⟷ (atom i # t ∧ j # t) ∨ (j # x ∧ (j # t ∨ j = atom i))
and forget-subst[simp]: atom a # tm ⟹ tm[a::=x]b = tm
and subst-id[simp]: tm[a::=(B-var a)]b = tm
and eqvt[simp,eqvt]: (p::perm) • (subst-b t1 x1 v) = (subst-b (p • t1) (p • x1) (p • v))
and flip-subst[simp]: atom bv # c ⟹ ((bv ↔ z) • c) = c[z::=B-var bv]b
and flip-subst-subst[simp]: atom bv # c ⟹ ((bv ↔ z) • c)[bv::=v]b = c[z::=v]b
begin
```

```
lemmas flip-subst-b = flip-subst-subst
```

```
lemma subst-b-simple-commute:
  fixes x::bv
  assumes atom x # c
  shows (c[z::=B-var x]b)[x::=b]b = c[z::=b]b
⟨proof⟩
```

```
lemma subst-b-flip-eq-one:
  fixes z1::bv and z2::bv and x1::bv and x2::bv
  assumes [[atom z1]]lst. c1 = [[atom z2]]lst. c2
  and atom x1 # (z1,z2,c1,c2)
  shows (c1[z1::=B-var x1]b) = (c2[z2::=B-var x1]b)
⟨proof⟩
```

```
lemma subst-b-flip-eq-two:
  fixes z1::bv and z2::bv and x1::bv and x2::bv
  assumes [[atom z1]]lst. c1 = [[atom z2]]lst. c2
  shows (c1[z1::=b]b) = (c2[z2::=b]b)
⟨proof⟩
```

```
lemma subst-b-fresh-x:
  fixes tm::'a::fs and x::x
  shows atom x # tm = atom x # tm[bv::=b]b
```

⟨proof⟩

**lemma** *subst-b-x-flip*[simp]:

**fixes**  $x'::x$  **and**  $x::x$  **and**  $bv::bv$

**shows**  $((x' \leftrightarrow x) \cdot tm)[bv::=b]_b = (x' \leftrightarrow x) \cdot (tm[bv::=b]_b)$

⟨proof⟩

**end**

## 4.2 Base Type

**nominal-function** *subst-bb* ::  $b \Rightarrow bv \Rightarrow b \Rightarrow b$  **where**

*subst-bb* (*B-var*  $bv2$ )  $bv1$   $b$  = (if  $bv1 = bv2$  then  $b$  else (*B-var*  $bv2$ ))

| *subst-bb* *B-int*  $bv1$   $b$  = *B-int*

| *subst-bb* *B-bool*  $bv1$   $b$  = *B-bool*

| *subst-bb* (*B-id*  $s$ )  $bv1$   $b$  = *B-id*  $s$

| *subst-bb* (*B-pair*  $b1$   $b2$ )  $bv1$   $b$  = *B-pair* (*subst-bb*  $b1$   $bv1$   $b$ ) (*subst-bb*  $b2$   $bv1$   $b$ )

| *subst-bb* *B-unit*  $bv1$   $b$  = *B-unit*

| *subst-bb* *B-bitvec*  $bv1$   $b$  = *B-bitvec*

| *subst-bb* (*B-app*  $s$   $b2$ )  $bv1$   $b$  = *B-app*  $s$  (*subst-bb*  $b2$   $bv1$   $b$ )

⟨proof⟩

**nominal-termination** (*eqvt*) ⟨proof⟩

**abbreviation**

*subst-bb-abbrev* ::  $b \Rightarrow bv \Rightarrow b \Rightarrow b$  ( $\langle[-::=]_{bb}\rangle$  [1000,50,50] 1000)

**where**

$b[bv::=b]_{bb} \equiv \text{subst-bb } b \text{ } bv \text{ } b'$

**instantiation**  $b$  :: *has-subst-b*

**begin**

**definition** *subst-b* = *subst-bb*

**instance** ⟨proof⟩

**end**

**lemma** *subst-bb-inject*:

**assumes**  $b1 = b2[bv::=b]_{bb}$  **and**  $b2 \neq B\text{-var } bv$

**shows**

$b1 = B\text{-int} \implies b2 = B\text{-int}$  **and**

$b1 = B\text{-bool} \implies b2 = B\text{-bool}$  **and**

$b1 = B\text{-unit} \implies b2 = B\text{-unit}$  **and**

$b1 = B\text{-bitvec} \implies b2 = B\text{-bitvec}$  **and**

$b1 = B\text{-pair } b11 \text{ } b12 \implies (\exists b11' \text{ } b12'. b11 = b11'[bv::=b]_{bb} \wedge b12 = b12'[bv::=b]_{bb} \wedge b2 = B\text{-pair } b11' \text{ } b12')$  **and**

$b1 = B\text{-var } bv' \implies b2 = B\text{-var } bv'$  **and**

$b1 = B\text{-id } tyid \implies b2 = B\text{-id } tyid$  **and**

$b1 = B\text{-app } tyid \text{ } b11 \implies (\exists b11'. b11 = b11'[bv::=b]_{bb} \wedge b2 = B\text{-app } tyid \text{ } b11')$

⟨proof⟩

**lemma** *flip-b-subst4*:

**fixes**  $b1::b$  **and**  $bv1::bv$  **and**  $c::bv$  **and**  $b::b$

**assumes** *atom*  $c \# (b1, bv1)$

**shows**  $b1[bv1::=b]_{bb} = ((bv1 \leftrightarrow c) \cdot b1)[c ::= b]_{bb}$   
 ⟨proof⟩

**lemma** *subst-bb-flip-sym*:

**fixes**  $b1::b$  **and**  $b2::b$

**assumes**  $atom\ c \# b$  **and**  $atom\ c \# (bv1, bv2, b1, b2)$  **and**  $(bv1 \leftrightarrow c) \cdot b1 = (bv2 \leftrightarrow c) \cdot b2$

**shows**  $b1[bv1::=b]_{bb} = b2[bv2::=b]_{bb}$

⟨proof⟩

### 4.3 Value

**nominal-function** *subst-vb*  $:: v \Rightarrow bv \Rightarrow b \Rightarrow v$  **where**

*subst-vb* (V-lit l) x v = V-lit l

| *subst-vb* (V-var y) x v = V-var y

| *subst-vb* (V-cons tyid c v') x v = V-cons tyid c (subst-vb v' x v)

| *subst-vb* (V-consp tyid c b v') x v = V-consp tyid c (subst-bb b x v) (subst-vb v' x v)

| *subst-vb* (V-pair v1 v2) x v = V-pair (subst-vb v1 x v) (subst-vb v2 x v)

⟨proof⟩

**nominal-termination** (eqvt) ⟨proof⟩

**abbreviation**

*subst-vb-abbrev*  $:: v \Rightarrow bv \Rightarrow b \Rightarrow v$  ( $\langle \cdot [- ::= -]_{vb} \rangle$  [1000,50,50] 500)

**where**

$e[bv::=b]_{vb} \equiv \text{subst-vb } e\ bv\ b$

**instantiation**  $v :: \text{has-subst-b}$

**begin**

**definition**  $\text{subst-b} = \text{subst-vb}$

**instance** ⟨proof⟩

**end**

### 4.4 Constraints Expressions

**nominal-function** *subst-ceb*  $:: ce \Rightarrow bv \Rightarrow b \Rightarrow ce$  **where**

*subst-ceb* ( (CE-val v') ) bv b = ( CE-val (subst-vb v' bv b) )

| *subst-ceb* ( (CE-op opp v1 v2) ) bv b = ( (CE-op opp (subst-ceb v1 bv b)(subst-ceb v2 bv b)) )

| *subst-ceb* ( (CE-fst v') ) bv b = CE-fst (subst-ceb v' bv b)

| *subst-ceb* ( (CE-snd v') ) bv b = CE-snd (subst-ceb v' bv b)

| *subst-ceb* ( (CE-len v') ) bv b = CE-len (subst-ceb v' bv b)

| *subst-ceb* ( CE-concat v1 v2 ) bv b = CE-concat (subst-ceb v1 bv b) (subst-ceb v2 bv b)

⟨proof⟩

**nominal-termination** (eqvt) ⟨proof⟩

**abbreviation**

*subst-ceb-abbrev*  $:: ce \Rightarrow bv \Rightarrow b \Rightarrow ce$  ( $\langle \cdot [- ::= -]_{ceb} \rangle$  [1000,50,50] 500)

**where**

$ce[bv::=b]_{ceb} \equiv \text{subst-ceb } ce\ bv\ b$

**instantiation**  $ce :: \text{has-subst-b}$

**begin**

**definition**  $subst-b = subst-ceb$

**instance**  $\langle proof \rangle$   
**end**

## 4.5 Constraints

**nominal-function**  $subst-cb :: c \Rightarrow bv \Rightarrow b \Rightarrow c$  **where**

$subst-cb (C-true) x v = C-true$   
|  $subst-cb (C-false) x v = C-false$   
|  $subst-cb (C-conj c1 c2) x v = C-conj (subst-cb c1 x v) (subst-cb c2 x v)$   
|  $subst-cb (C-disj c1 c2) x v = C-disj (subst-cb c1 x v) (subst-cb c2 x v)$   
|  $subst-cb (C-imp c1 c2) x v = C-imp (subst-cb c1 x v) (subst-cb c2 x v)$   
|  $subst-cb (C-eq e1 e2) x v = C-eq (subst-ceb e1 x v) (subst-ceb e2 x v)$   
|  $subst-cb (C-not c) x v = C-not (subst-cb c x v)$   
 $\langle proof \rangle$

**nominal-termination**  $(eqvt) \langle proof \rangle$

**abbreviation**

$subst-cb-abbrev :: c \Rightarrow bv \Rightarrow b \Rightarrow c \langle \langle [-::=]_{cb} \rangle [1000,50,50] 500 \rangle$   
**where**  
 $c[bv::=b]_{cb} \equiv subst-cb c bv b$

**instantiation**  $c :: has-subst-b$

**begin**

**definition**  $subst-b = subst-cb$

**instance**  $\langle proof \rangle$   
**end**

## 4.6 Types

**nominal-function**  $subst-tb :: \tau \Rightarrow bv \Rightarrow b \Rightarrow \tau$  **where**

$subst-tb (\{ z : b2 \mid c \}) bv1 b1 = \{ z : b2[bv1::=b1]_{bb} \mid c[bv1::=b1]_{cb} \}$   
 $\langle proof \rangle$

**nominal-termination**  $(eqvt) \langle proof \rangle$

**abbreviation**

$subst-tb-abbrev :: \tau \Rightarrow bv \Rightarrow b \Rightarrow \tau \langle \langle [-::=]_{tb} \rangle [1000,50,50] 1000 \rangle$   
**where**  
 $t[bv::=b]_{tb} \equiv subst-tb t bv b'$

**instantiation**  $\tau :: has-subst-b$

**begin**

**definition**  $subst-b = subst-tb$

**instance**  $\langle proof \rangle$   
**end**

**lemma**  $subst-bb-commute [simp]:$

$atom\ j\ \# \ A \implies (subst\text{-}bb\ (subst\text{-}bb\ A\ i\ t)\ j\ u) = subst\text{-}bb\ A\ i\ (subst\text{-}bb\ t\ j\ u)$   
 $\langle proof \rangle$

**lemma** *subst-vb-commute* [simp]:

$atom\ j\ \# \ A \implies (subst\text{-}vb\ (subst\text{-}vb\ A\ i\ t))\ j\ u = subst\text{-}vb\ A\ i\ (subst\text{-}bb\ t\ j\ u)$   
 $\langle proof \rangle$

**lemma** *subst-ceb-commute* [simp]:

$atom\ j\ \# \ A \implies (subst\text{-}ceb\ (subst\text{-}ceb\ A\ i\ t))\ j\ u = subst\text{-}ceb\ A\ i\ (subst\text{-}bb\ t\ j\ u)$   
 $\langle proof \rangle$

**lemma** *subst-cb-commute* [simp]:

$atom\ j\ \# \ A \implies (subst\text{-}cb\ (subst\text{-}cb\ A\ i\ t))\ j\ u = subst\text{-}cb\ A\ i\ (subst\text{-}bb\ t\ j\ u)$   
 $\langle proof \rangle$

**lemma** *subst-tb-commute* [simp]:

$atom\ j\ \# \ A \implies (subst\text{-}tb\ (subst\text{-}tb\ A\ i\ t))\ j\ u = subst\text{-}tb\ A\ i\ (subst\text{-}bb\ t\ j\ u)$   
 $\langle proof \rangle$

## 4.7 Expressions

**nominal-function** *subst-eb* ::  $e \Rightarrow bv \Rightarrow b \Rightarrow e$  **where**

$subst\text{-}eb\ (AE\text{-}val\ v')\ bv\ b = (AE\text{-}val\ (subst\text{-}vb\ v'\ bv\ b))$   
 $| subst\text{-}eb\ (AE\text{-}app\ f\ v')\ bv\ b = (AE\text{-}app\ f\ (subst\text{-}vb\ v'\ bv\ b))$   
 $| subst\text{-}eb\ (AE\text{-}appP\ f\ b'\ v')\ bv\ b = (AE\text{-}appP\ f\ (b'[bv::=b]_{bb})\ (subst\text{-}vb\ v'\ bv\ b))$   
 $| subst\text{-}eb\ (AE\text{-}op\ opp\ v1\ v2)\ bv\ b = (AE\text{-}op\ opp\ (subst\text{-}vb\ v1\ bv\ b)\ (subst\text{-}vb\ v2\ bv\ b))$   
 $| subst\text{-}eb\ (AE\text{-}fst\ v')\ bv\ b = AE\text{-}fst\ (subst\text{-}vb\ v'\ bv\ b)$   
 $| subst\text{-}eb\ (AE\text{-}snd\ v')\ bv\ b = AE\text{-}snd\ (subst\text{-}vb\ v'\ bv\ b)$   
 $| subst\text{-}eb\ (AE\text{-}mvar\ u)\ bv\ b = AE\text{-}mvar\ u$   
 $| subst\text{-}eb\ (AE\text{-}len\ v')\ bv\ b = AE\text{-}len\ (subst\text{-}vb\ v'\ bv\ b)$   
 $| subst\text{-}eb\ (AE\text{-}concat\ v1\ v2)\ bv\ b = AE\text{-}concat\ (subst\text{-}vb\ v1\ bv\ b)\ (subst\text{-}vb\ v2\ bv\ b)$   
 $| subst\text{-}eb\ (AE\text{-}split\ v1\ v2)\ bv\ b = AE\text{-}split\ (subst\text{-}vb\ v1\ bv\ b)\ (subst\text{-}vb\ v2\ bv\ b)$   
 $\langle proof \rangle$

**nominal-termination** (*eqvt*)  $\langle proof \rangle$

**abbreviation**

$subst\text{-}eb\text{-}abbrev :: e \Rightarrow bv \Rightarrow b \Rightarrow e\ (\lambda\cdot[-::=]_{eb})\ [1000,50,50]\ 500$

**where**

$e[bv::=b]_{eb} \equiv subst\text{-}eb\ e\ bv\ b$

**instantiation**  $e :: has\text{-}subst\text{-}b$

**begin**

**definition**  $subst\text{-}b = subst\text{-}eb$

**instance**  $\langle proof \rangle$

**end**

## 4.8 Statements

**nominal-function** (*default case-sum* ( $\lambda x. Inl\ undefined$ )) (*case-sum* ( $\lambda x. Inl\ undefined$ )) ( $\lambda x. Inr\ unde\text{-}fined$ ))

$subst\text{-}sb :: s \Rightarrow bv \Rightarrow b \Rightarrow s$

**and**  $subst\text{-}branchb :: branch\text{-}s \Rightarrow bv \Rightarrow b \Rightarrow branch\text{-}s$

**and**  $subst\text{-}branchlb :: branch\text{-}list \Rightarrow bv \Rightarrow b \Rightarrow branch\text{-}list$

**where**

$subst\text{-}sb (AS\text{-}val\ v')\ bv\ b = (AS\text{-}val\ (subst\text{-}vb\ v'\ bv\ b))$   
 $| subst\text{-}sb (AS\text{-}let\ y\ e\ s)\ bv\ b = (AS\text{-}let\ y\ (e[bv::=b]_{eb})\ (subst\text{-}sb\ s\ bv\ b))$   
 $| subst\text{-}sb (AS\text{-}let2\ y\ t\ s1\ s2)\ bv\ b = (AS\text{-}let2\ y\ (subst\text{-}tb\ t\ bv\ b)\ (subst\text{-}sb\ s1\ bv\ b)\ (subst\text{-}sb\ s2\ bv\ b))$   
 $| subst\text{-}sb (AS\text{-}match\ v'\ cs)\ bv\ b = AS\text{-}match\ (subst\text{-}vb\ v'\ bv\ b)\ (subst\text{-}branchlb\ cs\ bv\ b)$   
 $| subst\text{-}sb (AS\text{-}assign\ y\ v')\ bv\ b = AS\text{-}assign\ y\ (subst\text{-}vb\ v'\ bv\ b)$   
 $| subst\text{-}sb (AS\text{-}if\ v'\ s1\ s2)\ bv\ b = (AS\text{-}if\ (subst\text{-}vb\ v'\ bv\ b)\ (subst\text{-}sb\ s1\ bv\ b)\ (subst\text{-}sb\ s2\ bv\ b))$   
 $| subst\text{-}sb (AS\text{-}var\ u\ \tau\ v'\ s)\ bv\ b = AS\text{-}var\ u\ (subst\text{-}tb\ \tau\ bv\ b)\ (subst\text{-}vb\ v'\ bv\ b)\ (subst\text{-}sb\ s\ bv\ b)$   
 $| subst\text{-}sb (AS\text{-}while\ s1\ s2)\ bv\ b = AS\text{-}while\ (subst\text{-}sb\ s1\ bv\ b)\ (subst\text{-}sb\ s2\ bv\ b)$   
 $| subst\text{-}sb (AS\text{-}seq\ s1\ s2)\ bv\ b = AS\text{-}seq\ (subst\text{-}sb\ s1\ bv\ b)\ (subst\text{-}sb\ s2\ bv\ b)$   
 $| subst\text{-}sb (AS\text{-}assert\ c\ s)\ bv\ b = AS\text{-}assert\ (subst\text{-}cb\ c\ bv\ b)\ (subst\text{-}sb\ s\ bv\ b)$

$| subst\text{-}branchb (AS\text{-}branch\ dc\ x1\ s')\ bv\ b = AS\text{-}branch\ dc\ x1\ (subst\text{-}sb\ s'\ bv\ b)$

$| subst\text{-}branchlb (AS\text{-}final\ sb)\ bv\ b = AS\text{-}final\ (subst\text{-}branchb\ sb\ bv\ b)$

$| subst\text{-}branchlb (AS\text{-}cons\ sb\ ssb)\ bv\ b = AS\text{-}cons\ (subst\text{-}branchb\ sb\ bv\ b)\ (subst\text{-}branchlb\ ssb\ bv\ b)$

$\langle proof \rangle$

**nominal-termination** (*eqvt*)  $\langle proof \rangle$

**abbreviation**

$subst\text{-}sb\text{-}abbrev :: s \Rightarrow bv \Rightarrow b \Rightarrow s \ (\langle \text{[-::=]}_{sb} \rangle [1000,50,50] 1000)$

**where**

$b[bv::=b]_{sb} \equiv subst\text{-}sb\ b\ bv\ b'$

**lemma** *fresh-subst-sb-if* [*simp*]:

$(j \# (subst\text{-}sb\ A\ i\ x)) = ((atom\ i \# A \wedge j \# A) \vee (j \# x \wedge (j \# A \vee j = atom\ i)))$  **and**  
 $(j \# (subst\text{-}branchb\ B\ i\ x)) = ((atom\ i \# B \wedge j \# B) \vee (j \# x \wedge (j \# B \vee j = atom\ i)))$  **and**  
 $(j \# (subst\text{-}branchlb\ C\ i\ x)) = ((atom\ i \# C \wedge j \# C) \vee (j \# x \wedge (j \# C \vee j = atom\ i)))$

$\langle proof \rangle$

**lemma**

*forget-subst-sb*[*simp*]:  $atom\ a \# A \Longrightarrow subst\text{-}sb\ A\ a\ x = A$  **and**

*forget-subst-branchb* [*simp*]:  $atom\ a \# B \Longrightarrow subst\text{-}branchb\ B\ a\ x = B$  **and**

*forget-subst-branchlb*[*simp*]:  $atom\ a \# C \Longrightarrow subst\text{-}branchlb\ C\ a\ x = C$

$\langle proof \rangle$

**lemma** *subst-sb-id*:  $subst\text{-}sb\ A\ a\ (B\text{-}var\ a) = A$  **and**

*subst-branchb-id* [*simp*]:  $subst\text{-}branchb\ B\ a\ (B\text{-}var\ a) = B$  **and**

*subst-branchlb-id*:  $subst\text{-}branchlb\ C\ a\ (B\text{-}var\ a) = C$

$\langle proof \rangle$

**lemma** *flip-subst-s*:

**fixes**  $bv::bv$  **and**  $s::s$  **and**  $cs::branch\text{-}s$  **and**  $z::bv$

**shows**  $atom\ bv \# s \Longrightarrow ((bv \leftrightarrow z) \cdot s) = s[z::=B\text{-}var\ bv]_{sb}$  **and**

$atom\ bv \# cs \Longrightarrow ((bv \leftrightarrow z) \cdot cs) = subst\text{-}branchb\ cs\ z\ (B\text{-}var\ bv)$  **and**

$atom\ bv \# css \Longrightarrow ((bv \leftrightarrow z) \cdot css) = subst\text{-}branchlb\ css\ z\ (B\text{-}var\ bv)$

*<proof>*

**lemma** *flip-subst-subst-s*:

**fixes** *bv::bv and s::s and cs::branch-s and z::bv*

**shows** *atom bv # s  $\implies ((bv \leftrightarrow z) \cdot s)[bv::=v]_{sb} = s[z::=v]_{sb}$  and*

*atom bv # cs  $\implies subst-branchb ((bv \leftrightarrow z) \cdot cs) bv v = subst-branchb cs z v$  and*

*atom bv # css  $\implies subst-branchlb ((bv \leftrightarrow z) \cdot css) bv v = subst-branchlb css z v$*

*<proof>*

**instantiation** *s :: has-subst-b*

**begin**

**definition** *subst-b = ( $\lambda s bv b. subst-sb s bv b$ )*

**instance** *<proof>*

**end**

## 4.9 Function Type

**nominal-function** *subst-ft-b :: fun-typ  $\Rightarrow$  bv  $\Rightarrow$  b  $\Rightarrow$  fun-typ where*

*subst-ft-b ( AF-fun-typ z b c t (s::s) ) x v = AF-fun-typ z (subst-bb b x v) (subst-cb c x v) t[x::=v] <sub>$\tau b$</sub>*

*s[x::=v] <sub>$sb$</sub>*

*<proof>*

**nominal-termination** (*eqvt*) *<proof>*

**nominal-function** *subst-ftq-b :: fun-typ-q  $\Rightarrow$  bv  $\Rightarrow$  b  $\Rightarrow$  fun-typ-q where*

*atom bv # (x,v)  $\implies subst-ftq-b (AF-fun-typ-some bv ft) x v = (AF-fun-typ-some bv (subst-ft-b ft x v))$*

| *subst-ftq-b (AF-fun-typ-none ft) x v = (AF-fun-typ-none (subst-ft-b ft x v))*

*<proof>*

**nominal-termination** (*eqvt*) *<proof>*

**instantiation** *fun-typ :: has-subst-b*

**begin**

**definition** *subst-b = subst-ft-b*

Note: Using just `simp` in the second `apply` unpacks and gives a single goal whereas `auto` gives 43 non-intuitive goals. These goals are easier to solve and tedious, however they that it clear if a mistake is made in the definition of the function. For example, I saw that one of the goals was going through with `metis` and the next wasn't. It turned out the definition of the function itself was wrong

**instance** *<proof>*

**end**

**instantiation** *fun-typ-q :: has-subst-b*

**begin**

**definition** *subst-b = subst-ftq-b*

**instance** *<proof>*

**end**

## 4.10 Contexts

### 4.10.1 Immutable Variables

**nominal-function**  $subst\text{-}gb :: \Gamma \Rightarrow bv \Rightarrow b \Rightarrow \Gamma$  **where**

$subst\text{-}gb\ GNil\ -\ - = GNil$   
|  $subst\text{-}gb\ ((y,b',c)\#\Gamma)\ bv\ b = ((y,b'[bv::=b]_{bb},c[bv::=b]_{cb})\#\Gamma\ (subst\text{-}gb\ \Gamma\ bv\ b))$   
 $\langle proof \rangle$

**nominal-termination**  $(eqvt)\ \langle proof \rangle$

**abbreviation**

$subst\text{-}gb\text{-}abbrev :: \Gamma \Rightarrow bv \Rightarrow b \Rightarrow \Gamma\ (\langle[-::=]_{\Gamma b}\rangle\ [1000,50,50]\ 1000)$

**where**

$g[bv::=b]_{\Gamma b} \equiv subst\text{-}gb\ g\ bv\ b'$

**instantiation**  $\Gamma :: has\text{-}subst\text{-}b$

**begin**

**definition**  $subst\text{-}b = subst\text{-}gb$

**instance**  $\langle proof \rangle$

**end**

**lemma**  $subst\text{-}b\text{-}base\text{-}for\text{-}lit$ :

$(base\text{-}for\text{-}lit\ l)[bv::=b]_{bb} = base\text{-}for\text{-}lit\ l$   
 $\langle proof \rangle$

**lemma**  $subst\text{-}b\text{-}lookup$ :

**assumes**  $Some\ (b,\ c) = lookup\ \Gamma\ x$   
**shows**  $Some\ (b[bv::=b]_{bb},\ c[bv::=b]_{cb}) = lookup\ \Gamma[bv::=b]_{\Gamma b}\ x$   
 $\langle proof \rangle$

**lemma**  $subst\text{-}g\text{-}b\text{-}x\text{-}fresh$ :

**fixes**  $x::x$  **and**  $b::b$  **and**  $\Gamma::\Gamma$  **and**  $bv::bv$   
**assumes**  $atom\ x\ \#\ \Gamma$   
**shows**  $atom\ x\ \#\ \Gamma[bv::=b]_{\Gamma b}$   
 $\langle proof \rangle$

### 4.10.2 Mutable Variables

**nominal-function**  $subst\text{-}db :: \Delta \Rightarrow bv \Rightarrow b \Rightarrow \Delta$  **where**

$subst\text{-}db\ []_{\Delta}\ -\ - = []_{\Delta}$   
|  $subst\text{-}db\ ((u,t)\#\Delta)\ bv\ b = ((u,t[bv::=b]_{\tau b})\#\Delta\ (subst\text{-}db\ \Delta\ bv\ b))$   
 $\langle proof \rangle$

**nominal-termination**  $(eqvt)\ \langle proof \rangle$

**abbreviation**

$subst\text{-}db\text{-}abbrev :: \Delta \Rightarrow bv \Rightarrow b \Rightarrow \Delta\ (\langle[-::=]_{\Delta b}\rangle\ [1000,50,50]\ 1000)$

**where**

$\Delta[bv::=b]_{\Delta b} \equiv subst\text{-}db\ \Delta\ bv\ b$

**instantiation**  $\Delta :: has\text{-}subst\text{-}b$

**begin**

**definition**  $subst\text{-}b = subst\text{-}db$

**instance**  $\langle proof \rangle$   
**end**

**lemma** *subst-d-b-member*:  
**assumes**  $(u, \tau) \in \text{setD } \Delta$   
**shows**  $(u, \tau[bv::=b]_{\tau b}) \in \text{setD } \Delta[bv::=b]_{\Delta b}$   
 $\langle proof \rangle$

**lemmas** *ms-fresh-all* = *e.fresh s-branch-s-branch-list.fresh  $\tau$ .fresh c.fresh ce.fresh v.fresh l.fresh fresh-at-base opp.fresh pure-fresh ms-fresh*

**lemmas** *fresh-intros[intro]* = *fresh-GNil x-not-in-b-set x-not-in-u-atoms x-fresh-b u-not-in-x-atoms bv-not-in-x-atoms u-not-in-b-atoms*

**lemmas** *subst-b-simps* = *subst-tb.simps subst-cb.simps subst-ceb.simps subst-vb.simps subst-bb.simps subst-eb.simps subst-branchb.simps subst-sb.simps*

**lemma** *subst-d-b-x-fresh*:  
**fixes**  $x::x$  **and**  $b::b$  **and**  $\Delta::\Delta$  **and**  $bv::bv$   
**assumes**  $\text{atom } x \# \Delta$   
**shows**  $\text{atom } x \# \Delta[bv::=b]_{\Delta b}$   
 $\langle proof \rangle$

**lemma** *subst-b-fresh-x*:  
**fixes**  $x::x$   
**shows**  $\text{atom } x \# v \implies \text{atom } x \# v[bv::=b]_{vb}$  **and**  
 $\text{atom } x \# ce \implies \text{atom } x \# ce[bv::=b]_{ceb}$  **and**  
 $\text{atom } x \# e \implies \text{atom } x \# e[bv::=b]_{eb}$  **and**  
 $\text{atom } x \# c \implies \text{atom } x \# c[bv::=b]_{cb}$  **and**  
 $\text{atom } x \# t \implies \text{atom } x \# t[bv::=b]_{\tau b}$  **and**  
 $\text{atom } x \# d \implies \text{atom } x \# d[bv::=b]_{\Delta b}$  **and**  
 $\text{atom } x \# g \implies \text{atom } x \# g[bv::=b]_{\Gamma b}$  **and**  
 $\text{atom } x \# s \implies \text{atom } x \# s[bv::=b]_{sb}$   
 $\langle proof \rangle$

**lemma** *subst-b-fresh-u-cls*:  
**fixes**  $tm::'a::\text{has-subst-b}$  **and**  $x::u$   
**shows**  $\text{atom } x \# tm = \text{atom } x \# tm[bv::=b]_b$   
 $\langle proof \rangle$

**lemma** *subst-g-b-u-fresh*:  
**fixes**  $x::u$  **and**  $b::b$  **and**  $\Gamma::\Gamma$  **and**  $bv::bv$   
**assumes**  $\text{atom } x \# \Gamma$   
**shows**  $\text{atom } x \# \Gamma[bv::=b]_{\Gamma b}$   
 $\langle proof \rangle$

**lemma** *subst-d-b-u-fresh*:  
**fixes**  $x::u$  **and**  $b::b$  **and**  $\Gamma::\Delta$  **and**  $bv::bv$   
**assumes**  $\text{atom } x \# \Gamma$   
**shows**  $\text{atom } x \# \Gamma[bv::=b]_{\Delta b}$   
 $\langle proof \rangle$

**lemma** *subst-b-fresh-u*:

**fixes**  $x::u$

**shows**  $atom\ x \# v \implies atom\ x \# v[bv::=b]_{vb}$  **and**

$atom\ x \# ce \implies atom\ x \# ce[bv::=b]_{ceb}$  **and**

$atom\ x \# e \implies atom\ x \# e[bv::=b]_{eb}$  **and**

$atom\ x \# c \implies atom\ x \# c[bv::=b]_{cb}$  **and**

$atom\ x \# t \implies atom\ x \# t[bv::=b]_{\tau b}$  **and**

$atom\ x \# d \implies atom\ x \# d[bv::=b]_{\Delta b}$  **and**

$atom\ x \# g \implies atom\ x \# g[bv::=b]_{\Gamma b}$  **and**

$atom\ x \# s \implies atom\ x \# s[bv::=b]_{sb}$

$\langle proof \rangle$

**lemma** *subst-db-u-fresh*:

**fixes**  $u::u$  **and**  $b::b$  **and**  $D::\Delta$

**assumes**  $atom\ u \# D$

**shows**  $atom\ u \# D[bv::=b]_{\Delta b}$

$\langle proof \rangle$

**lemma** *flip-bt-subst4*:

**fixes**  $t::\tau$  **and**  $bv::bv$

**assumes**  $atom\ bv \# t$

**shows**  $t[bv'::=b]_{\tau b} = ((bv' \leftrightarrow bv) \cdot t)[bv::=b]_{\tau b}$

$\langle proof \rangle$

**lemma** *subst-bt-flip-sym*:

**fixes**  $t1::\tau$  **and**  $t2::\tau$

**assumes**  $atom\ bv \# b$  **and**  $atom\ bv \# (bv1, bv2, t1, t2)$  **and**  $(bv1 \leftrightarrow bv) \cdot t1 = (bv2 \leftrightarrow bv) \cdot t2$

**shows**  $t1[bv1::=b]_{\tau b} = t2[bv2::=b]_{\tau b}$

$\langle proof \rangle$

**end**

# Chapter 5

## Wellformed Terms

We require that expressions and values are well-formed. This includes a notion of well-sortedness. We identify a sort with a basic type and define the judgement as two clusters of mutually recursive inductive predicates. Some of the proofs are across all of the predicates and although they seemed at first to be daunting, they have all worked out well.

**named-theorems** *ms-wb Facts for helping with well-sortedness*

### 5.1 Definitions

**inductive**  $wfV :: \Theta \Rightarrow \mathcal{B} \Rightarrow \Gamma \Rightarrow v \Rightarrow b \Rightarrow \text{bool} (\langle - ; - ; - \vdash_{wf} - : - \rangle [50,50,50] 50)$  **and**  
 $wfC :: \Theta \Rightarrow \mathcal{B} \Rightarrow \Gamma \Rightarrow c \Rightarrow \text{bool} (\langle - ; - ; - \vdash_{wf} - \rangle [50,50] 50)$  **and**  
 $wfG :: \Theta \Rightarrow \mathcal{B} \Rightarrow \Gamma \Rightarrow \text{bool} (\langle - ; - \vdash_{wf} - \rangle [50,50] 50)$  **and**  
 $wfT :: \Theta \Rightarrow \mathcal{B} \Rightarrow \Gamma \Rightarrow \tau \Rightarrow \text{bool} (\langle - ; - ; - \vdash_{wf} - \rangle [50,50] 50)$  **and**  
 $wfTs :: \Theta \Rightarrow \mathcal{B} \Rightarrow \Gamma \Rightarrow (\text{string} * \tau) \text{ list} \Rightarrow \text{bool} (\langle - ; - ; - \vdash_{wf} - \rangle [50,50] 50)$  **and**  
 $wfTh :: \Theta \Rightarrow \text{bool} (\langle \vdash_{wf} - \rangle [50] 50)$  **and**  
 $wfB :: \Theta \Rightarrow \mathcal{B} \Rightarrow b \Rightarrow \text{bool} (\langle - ; - \vdash_{wf} - \rangle [50,50] 50)$  **and**  
 $wfCE :: \Theta \Rightarrow \mathcal{B} \Rightarrow \Gamma \Rightarrow ce \Rightarrow b \Rightarrow \text{bool} (\langle - ; - ; - \vdash_{wf} - : - \rangle [50,50,50] 50)$  **and**  
 $wfTD :: \Theta \Rightarrow \text{type-def} \Rightarrow \text{bool} (\langle - \vdash_{wf} - \rangle [50,50] 50)$   
**where**

$wfB\text{-intI}: \vdash_{wf} \Theta \Longrightarrow \Theta; \mathcal{B} \vdash_{wf} B\text{-int}$   
 $| wfB\text{-boolI}: \vdash_{wf} \Theta \Longrightarrow \Theta; \mathcal{B} \vdash_{wf} B\text{-bool}$   
 $| wfB\text{-unitI}: \vdash_{wf} \Theta \Longrightarrow \Theta; \mathcal{B} \vdash_{wf} B\text{-unit}$   
 $| wfB\text{-bitvecI}: \vdash_{wf} \Theta \Longrightarrow \Theta; \mathcal{B} \vdash_{wf} B\text{-bitvec}$   
 $| wfB\text{-pairI}: \llbracket \Theta; \mathcal{B} \vdash_{wf} b1 ; \Theta; \mathcal{B} \vdash_{wf} b2 \rrbracket \Longrightarrow \Theta; \mathcal{B} \vdash_{wf} B\text{-pair } b1 \ b2$

$| wfB\text{-consI}: \llbracket$   
 $\vdash_{wf} \Theta;$   
 $(AF\text{-typedef } s \text{ dclist}) \in \text{set } \Theta$   
 $\rrbracket \Longrightarrow$   
 $\Theta; \mathcal{B} \vdash_{wf} B\text{-id } s$

$| wfB\text{-appI}: \llbracket$   
 $\vdash_{wf} \Theta;$   
 $\Theta; \mathcal{B} \vdash_{wf} b;$   
 $(AF\text{-typedef-poly } s \text{ bv dclist}) \in \text{set } \Theta$   
 $\rrbracket \Longrightarrow$

$\Theta; \mathcal{B} \vdash_{wf} B\text{-app } s \ b$

|  $wfV\text{-varI}: \llbracket \Theta; \mathcal{B} \vdash_{wf} \Gamma; \text{Some } (b,c) = \text{lookup } \Gamma \ x \rrbracket \implies \Theta; \mathcal{B}; \Gamma \vdash_{wf} V\text{-var } x : b$   
 |  $wfV\text{-litI}: \Theta; \mathcal{B} \vdash_{wf} \Gamma \implies \Theta; \mathcal{B}; \Gamma \vdash_{wf} V\text{-lit } l : \text{base-for-lit } l$

|  $wfV\text{-pairI}: \llbracket$   
 $\Theta; \mathcal{B}; \Gamma \vdash_{wf} v1 : b1 ;$   
 $\Theta; \mathcal{B}; \Gamma \vdash_{wf} v2 : b2$   
 $\rrbracket \implies$   
 $\Theta; \mathcal{B}; \Gamma \vdash_{wf} (V\text{-pair } v1 \ v2) : B\text{-pair } b1 \ b2$

|  $wfV\text{-consI}: \llbracket$   
 $AF\text{-typedef } s \ dclist \in \text{set } \Theta;$   
 $(dc, \{ \! \{ x : b' \mid c \} \! \}) \in \text{set } dclist ;$   
 $\Theta; \mathcal{B}; \Gamma \vdash_{wf} v : b'$   
 $\rrbracket \implies$   
 $\Theta; \mathcal{B}; \Gamma \vdash_{wf} V\text{-cons } s \ dc \ v : B\text{-id } s$

|  $wfV\text{-conspI}: \llbracket$   
 $AF\text{-typedef-poly } s \ bv \ dclist \in \text{set } \Theta;$   
 $(dc, \{ \! \{ x : b' \mid c \} \! \}) \in \text{set } dclist ;$   
 $\Theta; \mathcal{B} \vdash_{wf} b;$   
 $\text{atom } bv \ \# \ (\Theta, \mathcal{B}, \Gamma, b, v);$   
 $\Theta; \mathcal{B}; \Gamma \vdash_{wf} v : b'[bv::=b]_{bb}$   
 $\rrbracket \implies$   
 $\Theta; \mathcal{B}; \Gamma \vdash_{wf} V\text{-consp } s \ dc \ b \ v : B\text{-app } s \ b$

|  $wfCE\text{-valI} : \llbracket$   
 $\Theta; \mathcal{B}; \Gamma \vdash_{wf} v : b$   
 $\rrbracket \implies$   
 $\Theta; \mathcal{B}; \Gamma \vdash_{wf} CE\text{-val } v : b$

|  $wfCE\text{-plusI}: \llbracket$   
 $\Theta; \mathcal{B}; \Gamma \vdash_{wf} v1 : B\text{-int};$   
 $\Theta; \mathcal{B}; \Gamma \vdash_{wf} v2 : B\text{-int}$   
 $\rrbracket \implies$   
 $\Theta; \mathcal{B}; \Gamma \vdash_{wf} CE\text{-op Plus } v1 \ v2 : B\text{-int}$

|  $wfCE\text{-leqI}: \llbracket$   
 $\Theta; \mathcal{B}; \Gamma \vdash_{wf} v1 : B\text{-int};$   
 $\Theta; \mathcal{B}; \Gamma \vdash_{wf} v2 : B\text{-int}$   
 $\rrbracket \implies$   
 $\Theta; \mathcal{B}; \Gamma \vdash_{wf} CE\text{-op LEq } v1 \ v2 : B\text{-bool}$

|  $wfCE\text{-eqI}: \llbracket$   
 $\Theta; \mathcal{B}; \Gamma \vdash_{wf} v1 : b;$   
 $\Theta; \mathcal{B}; \Gamma \vdash_{wf} v2 : b$   
 $\rrbracket \implies$   
 $\Theta; \mathcal{B}; \Gamma \vdash_{wf} CE\text{-op Eq } v1 \ v2 : B\text{-bool}$

|  $wfCE\text{-fstI}: \llbracket$   
 $\Theta; \mathcal{B}; \Gamma \vdash_{wf} v1 : B\text{-pair } b1 \ b2$

$$\begin{aligned} & \rceil \Longrightarrow \\ & \quad \Theta; \mathcal{B}; \Gamma \vdash_{wf} CE\text{-fst } v1 : b1 \\ \\ & | wfCE\text{-sndI}: \llbracket \\ & \quad \Theta; \mathcal{B}; \Gamma \vdash_{wf} v1 : B\text{-pair } b1 \ b2 \\ & \rceil \Longrightarrow \\ & \quad \Theta; \mathcal{B}; \Gamma \vdash_{wf} CE\text{-snd } v1 : b2 \\ \\ & | wfCE\text{-concatI}: \llbracket \\ & \quad \Theta; \mathcal{B}; \Gamma \vdash_{wf} v1 : B\text{-bitvec} ; \\ & \quad \Theta; \mathcal{B}; \Gamma \vdash_{wf} v2 : B\text{-bitvec} \\ & \rceil \Longrightarrow \\ & \quad \Theta; \mathcal{B}; \Gamma \vdash_{wf} CE\text{-concat } v1 \ v2 : B\text{-bitvec} \\ \\ & | wfCE\text{-lenI}: \llbracket \\ & \quad \Theta; \mathcal{B}; \Gamma \vdash_{wf} v1 : B\text{-bitvec} \\ & \rceil \Longrightarrow \\ & \quad \Theta; \mathcal{B}; \Gamma \vdash_{wf} CE\text{-len } v1 : B\text{-int} \\ \\ & | wfTI : \llbracket \\ & \quad atom \ z \ \sharp \ (\Theta, \mathcal{B}, \Gamma) ; \\ & \quad \Theta; \mathcal{B} \vdash_{wf} b ; \\ & \quad \Theta; \mathcal{B} ; (z, b, C\text{-true}) \#_{\Gamma} \Gamma \vdash_{wf} c \\ & \rceil \Longrightarrow \\ & \quad \Theta; \mathcal{B}; \Gamma \vdash_{wf} \{ z : b \mid c \} \\ \\ & | wfC\text{-eqI}: \llbracket \\ & \quad \Theta; \mathcal{B}; \Gamma \vdash_{wf} e1 : b ; \\ & \quad \Theta; \mathcal{B}; \Gamma \vdash_{wf} e2 : b \rceil \Longrightarrow \\ & \quad \Theta; \mathcal{B}; \Gamma \vdash_{wf} C\text{-eq } e1 \ e2 \\ & | wfC\text{-trueI}: \Theta; \mathcal{B} \vdash_{wf} \Gamma \Longrightarrow \Theta; \mathcal{B}; \Gamma \vdash_{wf} C\text{-true} \\ & | wfC\text{-falseI}: \Theta; \mathcal{B} \vdash_{wf} \Gamma \Longrightarrow \Theta; \mathcal{B}; \Gamma \vdash_{wf} C\text{-false} \\ \\ & | wfC\text{-conjI}: \llbracket \Theta; \mathcal{B}; \Gamma \vdash_{wf} c1 ; \Theta; \mathcal{B}; \Gamma \vdash_{wf} c2 \rceil \Longrightarrow \Theta; \mathcal{B}; \Gamma \vdash_{wf} C\text{-conj } c1 \ c2 \\ & | wfC\text{-disjI}: \llbracket \Theta; \mathcal{B}; \Gamma \vdash_{wf} c1 ; \Theta; \mathcal{B}; \Gamma \vdash_{wf} c2 \rceil \Longrightarrow \Theta; \mathcal{B}; \Gamma \vdash_{wf} C\text{-disj } c1 \ c2 \\ & | wfC\text{-notI}: \llbracket \Theta; \mathcal{B}; \Gamma \vdash_{wf} c1 \rceil \Longrightarrow \Theta; \mathcal{B}; \Gamma \vdash_{wf} C\text{-not } c1 \\ & | wfC\text{-impI}: \llbracket \Theta; \mathcal{B}; \Gamma \vdash_{wf} c1 ; \\ & \quad \Theta; \mathcal{B}; \Gamma \vdash_{wf} c2 \rceil \Longrightarrow \Theta; \mathcal{B}; \Gamma \vdash_{wf} C\text{-imp } c1 \ c2 \\ \\ & | wfG\text{-nilI}: \vdash_{wf} \Theta \Longrightarrow \Theta; \mathcal{B} \vdash_{wf} GNil \\ & | wfG\text{-cons1I}: \llbracket c \notin \{ TRUE, FALSE \} ; \\ & \quad \Theta; \mathcal{B} \vdash_{wf} \Gamma ; \\ & \quad atom \ x \ \sharp \ \Gamma ; \\ & \quad \Theta ; \mathcal{B} ; (x, b, C\text{-true}) \#_{\Gamma} \Gamma \vdash_{wf} c ; wfB \ \Theta \ \mathcal{B} \ b \\ & \rceil \Longrightarrow \Theta; \mathcal{B} \vdash_{wf} ((x, b, c) \#_{\Gamma} \Gamma) \\ & | wfG\text{-cons2I}: \llbracket c \in \{ TRUE, FALSE \} ; \\ & \quad \Theta; \mathcal{B} \vdash_{wf} \Gamma ; \\ & \quad atom \ x \ \sharp \ \Gamma ; \\ & \quad wfB \ \Theta \ \mathcal{B} \ b \\ & \rceil \Longrightarrow \Theta; \mathcal{B} \vdash_{wf} ((x, b, c) \#_{\Gamma} \Gamma) \\ \\ & | wfTh\text{-emptyI}: \vdash_{wf} \llbracket \end{aligned}$$

| *wfTh-consI*:  $\llbracket$   
 (name-of-type tdef)  $\notin$  name-of-type ' set  $\Theta$  ;  
 $\vdash_{wf} \Theta$  ;  
 $\Theta \vdash_{wf} \text{tdef} \rrbracket \implies \vdash_{wf} \text{tdef}\#\Theta$

| *wfTD-simpleI*:  $\llbracket$   
 $\Theta$  ;  $\{\|\}$  ;  $GNil \vdash_{wf} \text{lst}$   
 $\rrbracket \implies$   
 $\Theta \vdash_{wf} (AF\text{-typedef } s \text{ lst})$

| *wfTD-poly*:  $\llbracket$   
 $\Theta$  ;  $\{|bv|\}$  ;  $GNil \vdash_{wf} \text{lst}$   
 $\rrbracket \implies$   
 $\Theta \vdash_{wf} (AF\text{-typedef-poly } s \text{ bv lst})$

| *wfTs-nil*:  $\Theta$ ;  $\mathcal{B} \vdash_{wf} \Gamma \implies \Theta$ ;  $\mathcal{B}$ ;  $\Gamma \vdash_{wf} []::(\text{string}*\tau) \text{ list}$

| *wfTs-cons*:  $\llbracket \Theta$ ;  $\mathcal{B}$ ;  $\Gamma \vdash_{wf} \tau$  ;  
 $dc \notin \text{fst ' set } ts$  ;  
 $\Theta$ ;  $\mathcal{B}$ ;  $\Gamma \vdash_{wf} ts::(\text{string}*\tau) \text{ list} \rrbracket \implies \Theta$ ;  $\mathcal{B}$ ;  $\Gamma \vdash_{wf} ((dc,\tau)\#ts)$

**inductive-cases** *wfC-elim*s:

$\Theta$ ;  $\mathcal{B}$ ;  $\Gamma \vdash_{wf} C\text{-true}$   
 $\Theta$ ;  $\mathcal{B}$ ;  $\Gamma \vdash_{wf} C\text{-false}$   
 $\Theta$ ;  $\mathcal{B}$ ;  $\Gamma \vdash_{wf} C\text{-eq } e1 \ e2$   
 $\Theta$ ;  $\mathcal{B}$ ;  $\Gamma \vdash_{wf} C\text{-conj } c1 \ c2$   
 $\Theta$ ;  $\mathcal{B}$ ;  $\Gamma \vdash_{wf} C\text{-disj } c1 \ c2$   
 $\Theta$ ;  $\mathcal{B}$ ;  $\Gamma \vdash_{wf} C\text{-not } c1$   
 $\Theta$ ;  $\mathcal{B}$ ;  $\Gamma \vdash_{wf} C\text{-imp } c1 \ c2$

**inductive-cases** *wfV-elim*s:

$\Theta$ ;  $\mathcal{B}$ ;  $\Gamma \vdash_{wf} V\text{-var } x : b$   
 $\Theta$ ;  $\mathcal{B}$ ;  $\Gamma \vdash_{wf} V\text{-lit } l : b$   
 $\Theta$ ;  $\mathcal{B}$ ;  $\Gamma \vdash_{wf} V\text{-pair } v1 \ v2 : b$   
 $\Theta$ ;  $\mathcal{B}$ ;  $\Gamma \vdash_{wf} V\text{-cons } \text{tyid } dc \ v : b$   
 $\Theta$ ;  $\mathcal{B}$ ;  $\Gamma \vdash_{wf} V\text{-consp } \text{tyid } dc \ b \ v : b'$

**inductive-cases** *wfCE-elim*s:

$\Theta$ ;  $\mathcal{B}$ ;  $\Gamma \vdash_{wf} CE\text{-val } v : b$   
 $\Theta$ ;  $\mathcal{B}$ ;  $\Gamma \vdash_{wf} CE\text{-op Plus } v1 \ v2 : b$   
 $\Theta$ ;  $\mathcal{B}$ ;  $\Gamma \vdash_{wf} CE\text{-op LEq } v1 \ v2 : b$   
 $\Theta$ ;  $\mathcal{B}$ ;  $\Gamma \vdash_{wf} CE\text{-fst } v1 : b$   
 $\Theta$ ;  $\mathcal{B}$ ;  $\Gamma \vdash_{wf} CE\text{-snd } v1 : b$   
 $\Theta$ ;  $\mathcal{B}$ ;  $\Gamma \vdash_{wf} CE\text{-concat } v1 \ v2 : b$   
 $\Theta$ ;  $\mathcal{B}$ ;  $\Gamma \vdash_{wf} CE\text{-len } v1 : b$   
 $\Theta$ ;  $\mathcal{B}$ ;  $\Gamma \vdash_{wf} CE\text{-op opp } v1 \ v2 : b$   
 $\Theta$ ;  $\mathcal{B}$ ;  $\Gamma \vdash_{wf} CE\text{-op Eq } v1 \ v2 : b$

**inductive-cases** *wfT-elim*s:

$\Theta$ ;  $\mathcal{B}$  ;  $\Gamma \vdash_{wf} \tau::\tau$   
 $\Theta$ ;  $\mathcal{B}$  ;  $\Gamma \vdash_{wf} \llbracket z : b \mid c \rrbracket$

**inductive-cases** *wfG-elim*s:

- $\Theta; \mathcal{B} \vdash_{wf} GNil$
- $\Theta; \mathcal{B} \vdash_{wf} (x,b,c)\#_{\Gamma}\Gamma$
- $\Theta; \mathcal{B} \vdash_{wf} (x,b,TRUE)\#_{\Gamma}\Gamma$
- $\Theta; \mathcal{B} \vdash_{wf} (x,b,FALSE)\#_{\Gamma}\Gamma$

**inductive-cases** *wfTh-elim*s:

- $\vdash_{wf} []$
- $\vdash_{wf} td\#\Pi$

**inductive-cases** *wfTD-elim*s:

- $\Theta \vdash_{wf} (AF\text{-typedef } s \text{ } lst)$
- $\Theta \vdash_{wf} (AF\text{-typedef-poly } s \text{ } bv \text{ } lst)$

**inductive-cases** *wfTs-elim*s:

- $\Theta; \mathcal{B}; GNil \vdash_{wf} ([::((string*\tau) \text{ } list))$
- $\Theta; \mathcal{B}; GNil \vdash_{wf} ((t\#ts)::((string*\tau) \text{ } list))$

**inductive-cases** *wfB-elim*s:

- $\Theta; \mathcal{B} \vdash_{wf} B\text{-pair } b1 \text{ } b2$
- $\Theta; \mathcal{B} \vdash_{wf} B\text{-id } s$
- $\Theta; \mathcal{B} \vdash_{wf} B\text{-app } s \text{ } b$

**equivariance** *wfV*

This setup of 'avoiding' is not complete and for some of lemmas, such as weakening, do it the hard way

**nominal-inductive** *wfV*

**avoids** *wfV-conspI*: *bv* | *wfTI*: *z*  
*<proof>*

**inductive**

*wfE* ::  $\Theta \Rightarrow \Phi \Rightarrow \mathcal{B} \Rightarrow \Gamma \Rightarrow \Delta \Rightarrow e \Rightarrow b \Rightarrow \text{bool } (\langle - ; - ; - ; - ; - \vdash_{wf} - : - \rangle [50,50,50] 50)$  **and**  
*wfS* ::  $\Theta \Rightarrow \Phi \Rightarrow \mathcal{B} \Rightarrow \Gamma \Rightarrow \Delta \Rightarrow s \Rightarrow b \Rightarrow \text{bool } (\langle - ; - ; - ; - ; - \vdash_{wf} - : - \rangle [50,50,50] 50)$  **and**  
*wfCS* ::  $\Theta \Rightarrow \Phi \Rightarrow \mathcal{B} \Rightarrow \Gamma \Rightarrow \Delta \Rightarrow \text{tyid} \Rightarrow \text{string} \Rightarrow \tau \Rightarrow \text{branch-s} \Rightarrow b \Rightarrow \text{bool } (\langle - ; - ; - ; - ; - ; - ; - ; - \vdash_{wf} - : - \rangle [50,50,50,50,50] 50)$  **and**  
*wfCSS* ::  $\Theta \Rightarrow \Phi \Rightarrow \mathcal{B} \Rightarrow \Gamma \Rightarrow \Delta \Rightarrow \text{tyid} \Rightarrow (\text{string} * \tau) \text{ } list \Rightarrow \text{branch-list} \Rightarrow b \Rightarrow \text{bool } (\langle - ; - ; - ; - ; - ; - ; - ; - \vdash_{wf} - : - \rangle [50,50,50,50,50] 50)$  **and**  
*wfPhi* ::  $\Theta \Rightarrow \Phi \Rightarrow \text{bool } (\langle - \vdash_{wf} - \rangle [50,50] 50)$  **and**  
*wfD* ::  $\Theta \Rightarrow \mathcal{B} \Rightarrow \Gamma \Rightarrow \Delta \Rightarrow \text{bool } (\langle - ; - ; - \vdash_{wf} - \rangle [50,50] 50)$  **and**  
*wfFTQ* ::  $\Theta \Rightarrow \Phi \Rightarrow \text{fun-typ-q} \Rightarrow \text{bool } (\langle - ; - ; - \vdash_{wf} - \rangle [50] 50)$  **and**  
*wfFT* ::  $\Theta \Rightarrow \Phi \Rightarrow \mathcal{B} \Rightarrow \text{fun-typ} \Rightarrow \text{bool } (\langle - ; - ; - \vdash_{wf} - \rangle [50] 50)$  **where**

*wfE-valI* :  $\llbracket$   
 $\Theta \vdash_{wf} \Phi;$   
 $\Theta; \mathcal{B}; \Gamma \vdash_{wf} \Delta;$   
 $\Theta; \mathcal{B}; \Gamma \vdash_{wf} v : b$   
 $\rrbracket \Rightarrow$   
 $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash_{wf} AE\text{-val } v : b$

| *wfE-plusI*:  $\llbracket$   
 $\Theta \vdash_{wf} \Phi;$

$$\begin{array}{l}
\Theta; \mathcal{B}; \Gamma \vdash_{wf} \Delta; \\
\Theta; \mathcal{B}; \Gamma \vdash_{wf} v1 : B-int; \\
\Theta; \mathcal{B}; \Gamma \vdash_{wf} v2 : B-int \\
\] \Rightarrow \\
\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash_{wf} AE-op Plus v1 v2 : B-int \\
\\
| wfE-leqI: [ \\
\Theta \vdash_{wf} \Phi; \\
\Theta; \mathcal{B}; \Gamma \vdash_{wf} \Delta; \\
\Theta; \mathcal{B}; \Gamma \vdash_{wf} v1 : B-int; \\
\Theta; \mathcal{B}; \Gamma \vdash_{wf} v2 : B-int \\
\] \Rightarrow \\
\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash_{wf} AE-op LEq v1 v2 : B-bool \\
\\
| wfE-eqI: [ \\
\Theta \vdash_{wf} \Phi; \\
\Theta; \mathcal{B}; \Gamma \vdash_{wf} \Delta; \\
\Theta; \mathcal{B}; \Gamma \vdash_{wf} v1 : b; \\
\Theta; \mathcal{B}; \Gamma \vdash_{wf} v2 : b; \\
b \in \{ B-bool, B-int, B-unit \} \\
\] \Rightarrow \\
\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash_{wf} AE-op Eq v1 v2 : B-bool \\
\\
| wfE-fstI: [ \\
\Theta \vdash_{wf} \Phi; \\
\Theta; \mathcal{B}; \Gamma \vdash_{wf} \Delta; \\
\Theta; \mathcal{B}; \Gamma \vdash_{wf} v1 : B-pair b1 b2 \\
\] \Rightarrow \\
\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash_{wf} AE-fst v1 : b1 \\
\\
| wfE-sndI: [ \\
\Theta \vdash_{wf} \Phi; \\
\Theta; \mathcal{B}; \Gamma \vdash_{wf} \Delta; \\
\Theta; \mathcal{B}; \Gamma \vdash_{wf} v1 : B-pair b1 b2 \\
\] \Rightarrow \\
\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash_{wf} AE-snd v1 : b2 \\
\\
| wfE-concatI: [ \\
\Theta \vdash_{wf} \Phi; \\
\Theta; \mathcal{B}; \Gamma \vdash_{wf} \Delta; \\
\Theta; \mathcal{B}; \Gamma \vdash_{wf} v1 : B-bitvec; \\
\Theta; \mathcal{B}; \Gamma \vdash_{wf} v2 : B-bitvec \\
\] \Rightarrow \\
\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash_{wf} AE-concat v1 v2 : B-bitvec \\
\\
| wfE-splitI: [ \\
\Theta \vdash_{wf} \Phi; \\
\Theta; \mathcal{B}; \Gamma \vdash_{wf} \Delta; \\
\Theta; \mathcal{B}; \Gamma \vdash_{wf} v1 : B-bitvec; \\
\Theta; \mathcal{B}; \Gamma \vdash_{wf} v2 : B-int \\
\] \Rightarrow \\
\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash_{wf} AE-split v1 v2 : B-pair B-bitvec B-bitvec
\end{array}$$

$| \text{wfE-lenI}: \llbracket$   
 $\Theta \vdash_{wf} \Phi ;$   
 $\Theta; \mathcal{B}; \Gamma \vdash_{wf} \Delta;$   
 $\Theta; \mathcal{B}; \Gamma \vdash_{wf} v1 : B\text{-bitvec}$   
 $\rrbracket \Rightarrow$   
 $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash_{wf} AE\text{-len } v1 : B\text{-int}$

$| \text{wfE-appI}: \llbracket$   
 $\Theta \vdash_{wf} \Phi ;$   
 $\Theta; \mathcal{B}; \Gamma \vdash_{wf} \Delta;$   
 $\text{Some } (AF\text{-fundef } f \ (AF\text{-fun-typ-none } (AF\text{-fun-typ } x \ b \ c \ \tau \ s))) = \text{lookup-fun } \Phi \ f ;$   
 $\Theta; \mathcal{B}; \Gamma \vdash_{wf} v : b$   
 $\rrbracket \Rightarrow$   
 $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash_{wf} AE\text{-app } f \ v : b\text{-of } \tau$

$| \text{wfE-appPI}: \llbracket$   
 $\Theta \vdash_{wf} \Phi ;$   
 $\Theta; \mathcal{B}; \Gamma \vdash_{wf} \Delta;$   
 $\Theta; \mathcal{B} \vdash_{wf} b';$   
 $\text{atom } bv \ \# \ (\Phi, \Theta, \mathcal{B}, \Gamma, \Delta, b', v, (b\text{-of } \tau)[bv::=b]_b);$   
 $\text{Some } (AF\text{-fundef } f \ (AF\text{-fun-typ-some } bv \ (AF\text{-fun-typ } x \ b \ c \ \tau \ s))) = \text{lookup-fun } \Phi \ f ;$   
 $\Theta; \mathcal{B}; \Gamma \vdash_{wf} v : (b[bv::=b]_b)$   
 $\rrbracket \Rightarrow$   
 $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash_{wf} (AE\text{-appP } f \ b' \ v) : ((b\text{-of } \tau)[bv::=b]_b)$

$| \text{wfE-mvarI}: \llbracket$   
 $\Theta \vdash_{wf} \Phi ;$   
 $\Theta; \mathcal{B}; \Gamma \vdash_{wf} \Delta;$   
 $(u, \tau) \in \text{setD } \Delta$   
 $\rrbracket \Rightarrow$   
 $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash_{wf} AE\text{-mvar } u : b\text{-of } \tau$

$| \text{wfS-valI}: \llbracket$   
 $\Theta \vdash_{wf} \Phi ;$   
 $\Theta; \mathcal{B}; \Gamma \vdash_{wf} v : b ;$   
 $\Theta; \mathcal{B}; \Gamma \vdash_{wf} \Delta$   
 $\rrbracket \Rightarrow$   
 $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash_{wf} (AS\text{-val } v) : b$

$| \text{wfS-letI}: \llbracket$   
 $\text{wfE } \Theta \ \Phi \ \mathcal{B} \ \Gamma \ \Delta \ e \ b' ;$   
 $\Theta ; \Phi ; \mathcal{B} ; (x, b', C\text{-true}) \ \#_{\Gamma} \ \Gamma ; \Delta \vdash_{wf} s : b;$   
 $\Theta; \mathcal{B}; \Gamma \vdash_{wf} \Delta ;$   
 $\text{atom } x \ \# \ (\Phi, \Theta, \mathcal{B}, \Gamma, \Delta, e, b)$   
 $\rrbracket \Rightarrow$   
 $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash_{wf} LET \ x = e \ IN \ s : b$

$| \text{wfS-assertI}: \llbracket$   
 $\Theta ; \Phi ; \mathcal{B} ; (x, B\text{-bool}, c) \ \#_{\Gamma} \ \Gamma ; \Delta \vdash_{wf} s : b;$   
 $\Theta; \mathcal{B}; \Gamma \vdash_{wf} c ;$   
 $\Theta; \mathcal{B}; \Gamma \vdash_{wf} \Delta ;$

$$\begin{array}{l}
\text{atom } x \# (\Phi, \Theta, \mathcal{B}, \Gamma, \Delta, c, b, s) \\
\] \Rightarrow \\
\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash_{wf} \text{ASSERT } c \text{ IN } s : b \\
\\
| \text{wfs-let2I: } \llbracket \\
\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash_{wf} s1 : b\text{-of } \tau ; \\
\Theta; \mathcal{B}; \Gamma \vdash_{wf} \tau; \\
\Theta; \Phi; \mathcal{B}; (x, b\text{-of } \tau, C\text{-true}) \#_{\Gamma} \Gamma; \Delta \vdash_{wf} s2 : b ; \\
\text{atom } x \# (\Phi, \Theta, \mathcal{B}, \Gamma, \Delta, s1, b, \tau) \\
\] \Rightarrow \\
\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash_{wf} \text{LET } x : \tau = s1 \text{ IN } s2 : b \\
\\
| \text{wfs-ifi: } \llbracket \\
\Theta; \mathcal{B}; \Gamma \vdash_{wf} v : B\text{-bool}; \\
\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash_{wf} s1 : b ; \\
\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash_{wf} s2 : b \\
\] \Rightarrow \\
\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash_{wf} \text{IF } v \text{ THEN } s1 \text{ ELSE } s2 : b \\
\\
| \text{wfs-varI: } \llbracket \\
\text{wft } \Theta \mathcal{B} \Gamma \tau ; \\
\Theta; \mathcal{B}; \Gamma \vdash_{wf} v : b\text{-of } \tau; \\
\text{atom } u \# (\Phi, \Theta, \mathcal{B}, \Gamma, \Delta, \tau, v, b); \\
\Theta; \Phi; \mathcal{B}; \Gamma; (u, \tau) \#_{\Delta} \Delta \vdash_{wf} s : b \\
\] \Rightarrow \\
\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash_{wf} \text{VAR } u : \tau = v \text{ IN } s : b \\
\\
| \text{wfs-assignI: } \llbracket \\
(u, \tau) \in \text{setD } \Delta ; \\
\Theta; \mathcal{B}; \Gamma \vdash_{wf} \Delta ; \\
\Theta \vdash_{wf} \Phi ; \\
\Theta; \mathcal{B}; \Gamma \vdash_{wf} v : b\text{-of } \tau \\
\] \Rightarrow \\
\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash_{wf} u ::= v : B\text{-unit} \\
\\
| \text{wfs-whileI: } \llbracket \\
\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash_{wf} s1 : B\text{-bool} ; \\
\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash_{wf} s2 : b \\
\] \Rightarrow \\
\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash_{wf} \text{WHILE } s1 \text{ DO } \{ s2 \} : b \\
\\
| \text{wfs-seqI: } \llbracket \\
\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash_{wf} s1 : B\text{-unit} ; \\
\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash_{wf} s2 : b \\
\] \Rightarrow \\
\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash_{wf} s1 ;; s2 : b \\
\\
| \text{wfs-matchI: } \llbracket \\
\text{wfV } \Theta \mathcal{B} \Gamma v (B\text{-id tid}) ; \\
(AF\text{-typedef tid dclist}) \in \text{set } \Theta; \\
\text{wfD } \Theta \mathcal{B} \Gamma \Delta ; \\
\Theta \vdash_{wf} \Phi ;
\end{array}$$

$$\begin{array}{l}
\Theta ; \Phi ; \mathcal{B} ; \Gamma ; \Delta ; tid ; dclist \vdash_{wf} cs : b \\
\boxed{\Longrightarrow} \\
\Theta ; \Phi ; \mathcal{B} ; \Gamma ; \Delta \vdash_{wf} AS\text{-match } v \text{ } cs : b \\
\\
| \text{ wfS-branchI: } \boxed{\begin{array}{l}
\Theta ; \Phi ; \mathcal{B} ; (x, b\text{-of } \tau, C\text{-true}) \#_{\Gamma} \Gamma ; \Delta \vdash_{wf} s : b ; \\
atom \ x \ \# (\Phi, \Theta, \mathcal{B}, \Gamma, \Delta, \Gamma, \tau); \\
\Theta ; \mathcal{B} ; \Gamma \vdash_{wf} \Delta
\end{array}} \\
\boxed{\Longrightarrow} \\
\Theta ; \Phi ; \mathcal{B} ; \Gamma ; \Delta ; tid ; dc ; \tau \vdash_{wf} dc \ x \Rightarrow s : b \\
\\
| \text{ wfS-finalI: } \boxed{\begin{array}{l}
\Theta ; \Phi ; \mathcal{B} ; \Gamma ; \Delta ; tid ; dc ; t \vdash_{wf} cs : b \\
\Theta ; \Phi ; \mathcal{B} ; \Gamma ; \Delta ; tid ; [(dc, t)] \vdash_{wf} AS\text{-final } cs : b
\end{array}} \\
\\
| \text{ wfS-cons: } \boxed{\begin{array}{l}
\Theta ; \Phi ; \mathcal{B} ; \Gamma ; \Delta ; tid ; dc ; t \vdash_{wf} cs : b ; \\
\Theta ; \Phi ; \mathcal{B} ; \Gamma ; \Delta ; tid ; dclist \vdash_{wf} css : b \\
\Theta ; \Phi ; \mathcal{B} ; \Gamma ; \Delta ; tid ; (dc, t) \# dclist \vdash_{wf} AS\text{-cons } cs \text{ } css : b
\end{array}} \\
\\
| \text{ wfD-emptyI: } \Theta ; \mathcal{B} \vdash_{wf} \Gamma \Longrightarrow \Theta ; \mathcal{B} ; \Gamma \vdash_{wf} \boxed{\Delta} \\
\\
| \text{ wfD-cons: } \boxed{\begin{array}{l}
\Theta ; \mathcal{B} ; \Gamma \vdash_{wf} \Delta :: \Delta ; \\
\Theta ; \mathcal{B} ; \Gamma \vdash_{wf} \tau ; \\
u \notin fst \text{ ' } setD \ \Delta \\
\Theta ; \mathcal{B} ; \Gamma \vdash_{wf} ((u, \tau) \#_{\Delta} \Delta)
\end{array}} \\
\\
| \text{ wfPhi-emptyI: } \vdash_{wf} \Theta \Longrightarrow \Theta \vdash_{wf} \boxed{\phantom{\Delta}} \\
\\
| \text{ wfPhi-consI: } \boxed{\begin{array}{l}
f \notin name\text{-of-fun ' } set \ \Phi ; \\
\Theta ; \Phi \vdash_{wf} ft ; \\
\Theta \vdash_{wf} \Phi \\
\Theta \vdash_{wf} ((AF\text{-fundef } f \text{ } ft) \# \Phi)
\end{array}} \\
\\
| \text{ wfFTNone: } \Theta ; \Phi ; \{\|\} \vdash_{wf} ft \Longrightarrow \Theta ; \Phi \vdash_{wf} AF\text{-fun-typ-none } ft \\
| \text{ wfFTSome: } \Theta ; \Phi ; \{|\ bv \|\} \vdash_{wf} ft \Longrightarrow \Theta ; \Phi \vdash_{wf} AF\text{-fun-typ-some } bv \text{ } ft \\
\\
| \text{ wfFTI: } \boxed{\begin{array}{l}
\Theta ; B \vdash_{wf} b ; \\
supp \ s \subseteq \{atom \ x\} \cup supp \ B ; \\
supp \ c \subseteq \{atom \ x\} ; \\
\Theta ; B ; (x, b, c) \#_{\Gamma} GNil \vdash_{wf} \tau ; \\
\Theta \vdash_{wf} \Phi \\
\Theta ; \Phi ; B \vdash_{wf} (AF\text{-fun-typ } x \text{ } b \text{ } c \text{ } \tau \text{ } s)
\end{array}}
\end{array}$$

**inductive-cases** *wfE-elim*s:

$\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash_{wf} AE\text{-val } v : b$   
 $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash_{wf} AE\text{-op Plus } v1\ v2 : b$   
 $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash_{wf} AE\text{-op LEq } v1\ v2 : b$   
 $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash_{wf} AE\text{-fst } v1 : b$   
 $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash_{wf} AE\text{-snd } v1 : b$   
 $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash_{wf} AE\text{-concat } v1\ v2 : b$   
 $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash_{wf} AE\text{-len } v1 : b$   
 $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash_{wf} AE\text{-op opp } v1\ v2 : b$   
 $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash_{wf} AE\text{-app } f\ v : b$   
 $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash_{wf} AE\text{-appP } f\ b'\ v : b$   
 $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash_{wf} AE\text{-mvar } u : b$   
 $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash_{wf} AE\text{-op Eq } v1\ v2 : b$

**inductive-cases** *wfCS-elim*s:

$\Theta; \Phi; \mathcal{B}; \Gamma; \Delta; tid; dc; t \vdash_{wf} (cs::branch\text{-}s) : b$   
 $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta; tid; dc \vdash_{wf} (cs::branch\text{-}list) : b$

**inductive-cases** *wfPhi-elim*s:

$\Theta \vdash_{wf} []$   
 $\Theta \vdash_{wf} ((AF\text{-fundef } f\ ft)\#\Pi)$   
 $\Theta \vdash_{wf} (fd\#\Phi::\Phi)$

**declare**[[ *simproc del: alpha-1st*]]

**inductive-cases** *wfFTQ-elim*s:

$\Theta; \Phi \vdash_{wf} AF\text{-fun-typ-none } ft$   
 $\Theta; \Phi \vdash_{wf} AF\text{-fun-typ-some } bv\ ft$   
 $\Theta; \Phi \vdash_{wf} AF\text{-fun-typ-some } bv\ (AF\text{-fun-typ } x\ b\ c\ \tau\ s)$

**inductive-cases** *wfFT-elim*s:

$\Theta; \Phi; \mathcal{B} \vdash_{wf} AF\text{-fun-typ } x\ b\ c\ \tau\ s$

**declare**[[ *simproc add: alpha-1st*]]

**inductive-cases** *wfD-elim*s:

$\Pi; \mathcal{B}; (\Gamma::\Gamma) \vdash_{wf} []_{\Delta}$   
 $\Pi; \mathcal{B}; (\Gamma::\Gamma) \vdash_{wf} (u, \tau) \#_{\Delta} \Delta::\Delta$

**equivariance** *wfE*

**nominal-inductive** *wfE*

**avoids** *wfE-appPI: bv* | *wfS-varI: u* | *wfS-letI: x* | *wfS-let2I: x* | *wfS-branchI: x* | *wfS-assertI: x*

*<proof>*

**inductive** *wfVDs* :: *var-def list*  $\Rightarrow$  *bool* **where**

*wfVDs-nilI: wfVDs* []

| *wfVDs-consI: []*  
*atom u*  $\#$  *ts*;

```

wfV ( $\emptyset::\Theta$ )  $\{\emptyset\}$  GNil v (b-of  $\tau$ );
wfT ( $\emptyset::\Theta$ )  $\{\emptyset\}$  GNil  $\tau$ ;
wfVDs ts
 $\mathbb{I} \implies$  wfVDs  $((AV-def\ u\ \tau\ v)\#ts)$ 

```

**equivariance** *wfVDs*

**nominal-inductive** *wfVDs*  $\langle proof \rangle$

**end**

**hide-const** *Syntax.dom*

## Chapter 6

# Refinement Constraint Logic

Semantics for the logic we use in the refinement constraints. It is a multi-sorted, quantifier free logic with polymorphic datatypes and linear arithmetic. We could have modelled by using one of the encodings to FOL however we wanted to explore using a more direct model.

### 6.1 Evaluation and Satisfiability

#### 6.1.1 Valuation

Refinement constraint logic values.  $S_{\text{Ut}}$  is a value for the uninterpreted sort that corresponds to basic type variables. For now we only need one of these universes. We wrap an `smt_val` inside it during a process we call 'boxing' which is introduced in the `RCLLogicL` theory

**nominal-datatype**  $rcl\text{-}val = S_{\text{Bitvec}} \textit{bit list} \mid S_{\text{Num}} \textit{int} \mid S_{\text{Bool}} \textit{bool} \mid S_{\text{Pair}} \textit{rcl-val rcl-val} \mid$   
 $S_{\text{Cons}} \textit{tyid string rcl-val} \mid S_{\text{Consp}} \textit{tyid string b rcl-val} \mid$   
 $S_{\text{Unit}} \mid S_{\text{Ut}} \textit{rcl-val}$

RCL sorts. Represent our domains. The universe is the union of all of the these.  $S_{\text{Ut}}$  is the single uninterpreted sort. These map almost directly to basic type but we have them to distinguish syntax (basic types) and semantics (RCL sorts)

**nominal-datatype**  $rcl\text{-}sort = S\text{-}bool \mid S\text{-}int \mid S\text{-}unit \mid S\text{-}pair \textit{rcl-sort rcl-sort} \mid S\text{-}id \textit{tyid} \mid S\text{-}app \textit{tyid}$   
 $rcl\text{-}sort \mid S\text{-}bitvec \mid S\text{-}ut$

**type-synonym**  $valuation = (x, rcl\text{-}val) \textit{map}$

**type-synonym**  $type\text{-}valuation = (bv, rcl\text{-}sort) \textit{map}$

Well-sortedness for RCL values

**inductive**  $wfRCV:: \Theta \Rightarrow rcl\text{-}val \Rightarrow b \Rightarrow bool \ ( \langle - \vdash - : - \rangle [50,50] 50) \textbf{ where}$   
 $wfRCV\text{-}B_{\text{Bitvec}}I: P \vdash (S_{\text{Bitvec}} \textit{bv}) : B\text{-}bitvec$   
 $\mid wfRCV\text{-}B_{\text{Int}}I: P \vdash (S_{\text{Num}} \textit{n}) : B\text{-}int$   
 $\mid wfRCV\text{-}B_{\text{Bool}}I: P \vdash (S_{\text{Bool}} \textit{b}) : B\text{-}bool$   
 $\mid wfRCV\text{-}B_{\text{Pair}}I: \llbracket P \vdash s1 : b1 ; P \vdash s2 : b2 \rrbracket \Longrightarrow P \vdash (S_{\text{Pair}} \textit{s1 s2}) : (B\text{-}pair \textit{b1 b2})$   
 $\mid wfRCV\text{-}B_{\text{Cons}}I: \llbracket AF\text{-}typedef \textit{s dclist} \in \textit{set} \ \Theta;$   
 $\quad (dc, \{ \! \{ x : b \mid c \} \! \}) \in \textit{set} \ \textit{dclist} ;$   
 $\quad \Theta \vdash s1 : b \rrbracket \Longrightarrow \Theta \vdash (S_{\text{Cons}} \textit{s dc s1}) : (B\text{-}id \ \textit{s})$   
 $\mid wfRCV\text{-}B_{\text{ConsPI}}I: \llbracket AF\text{-}typedef\text{-}poly \ \textit{s bv dclist} \in \textit{set} \ \Theta;$

$(dc, \{ x : b \mid c \}) \in \text{set dclist} ;$   
 $\text{atom } bv \# (\Theta, S\text{Consp } s \text{ dc } b' \text{ s1}, B\text{-app } s \text{ b}') ;$   
 $\Theta \vdash s1 : b[bv ::= b']_{bb} \Longrightarrow \Theta \vdash (S\text{Consp } s \text{ dc } b' \text{ s1}) : (B\text{-app } s \text{ b}')$   
 $| \text{wfRCV-BUnitI: } P \vdash S\text{Unit} : B\text{-unit}$   
 $| \text{wfRCV-BVarI: } P \vdash (S\text{U}t \text{ n}) : (B\text{-var } bv)$   
**equivariance** *wfRCV*  
**nominal-inductive** *wfRCV*  
**avoids** *wfRCV-BConsPI: bv*  
 $\langle \text{proof} \rangle$

**inductive-cases** *wfRCV-elim* :

*wfRCV P s B-bitvec*  
*wfRCV P s (B-pair b1 b2)*  
*wfRCV P s (B-int)*  
*wfRCV P s (B-bool)*  
*wfRCV P s (B-id ss)*  
*wfRCV P s (B-var bv)*  
*wfRCV P s (B-unit)*  
*wfRCV P s (B-app tyid b)*  
*wfRCV P (SBitvec bv) b*  
*wfRCV P (SNum n) b*  
*wfRCV P (SBool n) b*  
*wfRCV P (SPair s1 s2) b*  
*wfRCV P (SCons s dc s1) b*  
*wfRCV P (SConsp s dc b' s1) b*  
*wfRCV P SUnit b*  
*wfRCV P (SUt s1) b*

Sometimes we want to assert  $P \vdash s \sim b[bv=b']$  and we want to know what  $b$  is however substitution is not injective so we can't write this in terms of *wfRCV*. So we define a relation that makes the components of the substitution explicit.

**inductive** *wfRCV-subst*::  $\Theta \Rightarrow \text{rcl-val} \Rightarrow b \Rightarrow (bv*b) \text{ option} \Rightarrow \text{bool}$  **where**

*wfRCV-subst-BBitvecI: wfRCV-subst P (SBitvec bv) B-bitvec sub*  
 $| \text{wfRCV-subst-BIntI: } wfRCV\text{-subst } P \text{ (SNum } n) \text{ B-int sub}$   
 $| \text{wfRCV-subst-BBoolI: } wfRCV\text{-subst } P \text{ (SBool } b) \text{ B-bool sub}$   
 $| \text{wfRCV-subst-BPairI: } \llbracket wfRCV\text{-subst } P \text{ s1 b1 sub ; } wfRCV\text{-subst } P \text{ s2 b2 sub} \rrbracket \Longrightarrow wfRCV\text{-subst } P \text{ (SPair } s1 \text{ s2) (B-pair } b1 \text{ b2) sub}$   
 $| \text{wfRCV-subst-BConsI: } \llbracket AF\text{-typedef } s \text{ dclist} \in \text{set } \Theta ;$   
 $(dc, \{ x : b \mid c \}) \in \text{set dclist} ;$   
 $wfRCV\text{-subst } \Theta \text{ s1 } b \text{ None} \rrbracket \Longrightarrow wfRCV\text{-subst } \Theta \text{ (SCons } s \text{ dc } s1) \text{ (B-id } s) \text{ sub}$   
 $| \text{wfRCV-subst-BConspI: } \llbracket AF\text{-typedef-poly } s \text{ bv dclist} \in \text{set } \Theta ;$   
 $(dc, \{ x : b \mid c \}) \in \text{set dclist} ;$   
 $wfRCV\text{-subst } \Theta \text{ s1 } (b[bv ::= b']_{bb}) \text{ sub} \rrbracket \Longrightarrow wfRCV\text{-subst } \Theta \text{ (SConsp } s \text{ dc } b' \text{ s1) (B-app } s \text{ b}') \text{ sub}$   
 $| \text{wfRCV-subst-BUnitI: } wfRCV\text{-subst } P \text{ SUnit } B\text{-unit sub}$   
 $| \text{wfRCV-subst-BVar1I: } bvar \neq bv \Longrightarrow wfRCV\text{-subst } P \text{ (SUt } n) \text{ (B-var } bv) \text{ (Some (bvar, bin))}$   
 $| \text{wfRCV-subst-BVar2I: } \llbracket bvar = bv ; wfRCV\text{-subst } P \text{ s bin None} \rrbracket \Longrightarrow wfRCV\text{-subst } P \text{ s (B-var } bv)$   
 $\text{(Some (bvar, bin))}$   
 $| \text{wfRCV-subst-BVar3I: } wfRCV\text{-subst } P \text{ (SUt } n) \text{ (B-var } bv) \text{ None}$   
**equivariance** *wfRCV-subst*  
**nominal-inductive** *wfRCV-subst*  $\langle \text{proof} \rangle$

### 6.1.2 Evaluation base-types

**inductive**  $eval-b :: type-valuation \Rightarrow b \Rightarrow rcl-sort \Rightarrow bool$  (  $\langle - \llbracket - \rrbracket \sim - \rangle$  ) **where**

- |  $v \llbracket B\text{-}bool \rrbracket \sim S\text{-}bool$
- |  $v \llbracket B\text{-}int \rrbracket \sim S\text{-}int$
- |  $Some\ s = v\ bv \Longrightarrow v \llbracket B\text{-}var\ bv \rrbracket \sim s$

**equivariance**  $eval-b$

**nominal-inductive**  $eval-b$   $\langle proof \rangle$

### 6.1.3 Wellformed vvaluations

**definition**  $wfI :: \Theta \Rightarrow \Gamma \Rightarrow valuation \Rightarrow bool$  (  $\langle - ; - \vdash - \rangle$  ) **where**

- $\Theta ; \Gamma \vdash i = (\forall (x,b,c) \in toSet\ \Gamma. \exists s. Some\ s = i\ x \wedge \Theta \vdash s : b)$

### 6.1.4 Evaluating Terms

**nominal-function**  $eval-l :: l \Rightarrow rcl-val$  (  $\langle \llbracket - \rrbracket \rangle$  ) **where**

- |  $\llbracket L\text{-}true \rrbracket = SBool\ True$
  - |  $\llbracket L\text{-}false \rrbracket = SBool\ False$
  - |  $\llbracket L\text{-}num\ n \rrbracket = SNum\ n$
  - |  $\llbracket L\text{-}unit \rrbracket = SUnit$
  - |  $\llbracket L\text{-}bitvec\ n \rrbracket = SBitvec\ n$
- $\langle proof \rangle$

**nominal-termination** ( $eqvt$ )  $\langle proof \rangle$

**inductive**  $eval-v :: valuation \Rightarrow v \Rightarrow rcl-val \Rightarrow bool$  (  $\langle - \llbracket - \rrbracket \sim - \rangle$  ) **where**

- $eval\text{-}v\text{-}litI: i \llbracket V\text{-}lit\ l \rrbracket \sim \llbracket l \rrbracket$
- |  $eval\text{-}v\text{-}varI: Some\ sv = i\ x \Longrightarrow i \llbracket V\text{-}var\ x \rrbracket \sim sv$
- |  $eval\text{-}v\text{-}pairI: [ i \llbracket v1 \rrbracket \sim s1 ; i \llbracket v2 \rrbracket \sim s2 ] \Longrightarrow i \llbracket V\text{-}pair\ v1\ v2 \rrbracket \sim SPair\ s1\ s2$
- |  $eval\text{-}v\text{-}consI: i \llbracket v \rrbracket \sim s \Longrightarrow i \llbracket V\text{-}cons\ tyid\ dc\ v \rrbracket \sim SCons\ tyid\ dc\ s$
- |  $eval\text{-}v\text{-}conspI: i \llbracket v \rrbracket \sim s \Longrightarrow i \llbracket V\text{-}consp\ tyid\ dc\ b\ v \rrbracket \sim SConsp\ tyid\ dc\ b\ s$

**equivariance**  $eval-v$

**nominal-inductive**  $eval-v$   $\langle proof \rangle$

**inductive-cases**  $eval-v\text{-}elims$ :

- $i \llbracket V\text{-}lit\ l \rrbracket \sim s$
- $i \llbracket V\text{-}var\ x \rrbracket \sim s$
- $i \llbracket V\text{-}pair\ v1\ v2 \rrbracket \sim s$
- $i \llbracket V\text{-}cons\ tyid\ dc\ v \rrbracket \sim s$
- $i \llbracket V\text{-}consp\ tyid\ dc\ b\ v \rrbracket \sim s$

**inductive**  $eval-e :: valuation \Rightarrow ce \Rightarrow rcl-val \Rightarrow bool$  (  $\langle - \llbracket - \rrbracket \sim - \rangle$  ) **where**

- $eval\text{-}e\text{-}valI: i \llbracket v \rrbracket \sim sv \Longrightarrow i \llbracket CE\text{-}val\ v \rrbracket \sim sv$
- |  $eval\text{-}e\text{-}plusI: [ i \llbracket v1 \rrbracket \sim SNum\ n1 ; i \llbracket v2 \rrbracket \sim SNum\ n2 ] \Longrightarrow i \llbracket (CE\text{-}op\ Plus\ v1\ v2) \rrbracket \sim (SNum\ (n1+n2))$
- |  $eval\text{-}e\text{-}leqI: [ i \llbracket v1 \rrbracket \sim (SNum\ n1) ; i \llbracket v2 \rrbracket \sim (SNum\ n2) ] \Longrightarrow i \llbracket (CE\text{-}op\ LEq\ v1\ v2) \rrbracket \sim (SBool\ (n1 \leq n2))$
- |  $eval\text{-}e\text{-}eqI: [ i \llbracket v1 \rrbracket \sim s1 ; i \llbracket v2 \rrbracket \sim s2 ] \Longrightarrow i \llbracket (CE\text{-}op\ Eq\ v1\ v2) \rrbracket \sim (SBool\ (s1 = s2))$
- |  $eval\text{-}e\text{-}fstI: [ i \llbracket v \rrbracket \sim SPair\ v1\ v2 ] \Longrightarrow i \llbracket (CE\text{-}fst\ v) \rrbracket \sim v1$
- |  $eval\text{-}e\text{-}sndI: [ i \llbracket v \rrbracket \sim SPair\ v1\ v2 ] \Longrightarrow i \llbracket (CE\text{-}snd\ v) \rrbracket \sim v2$
- |  $eval\text{-}e\text{-}concatI: [ i \llbracket v1 \rrbracket \sim (SBitvec\ bv1) ; i \llbracket v2 \rrbracket \sim (SBitvec\ bv2) ] \Longrightarrow i \llbracket (CE\text{-}concat\ v1\ v2) \rrbracket \sim (SBitvec\ (bv1@bv2))$
- |  $eval\text{-}e\text{-}lenI: [ i \llbracket v \rrbracket \sim (SBitvec\ bv) ] \Longrightarrow i \llbracket (CE\text{-}len\ v) \rrbracket \sim (SNum\ (int\ (List.length\ bv)))$

**equivariance** *eval-e*

**nominal-inductive** *eval-e*  $\langle$ proof $\rangle$

**inductive-cases** *eval-e-elim*:

- $i \llbracket (CE\text{-val } v) \rrbracket \sim s$
- $i \llbracket (CE\text{-op Plus } v1 \ v2) \rrbracket \sim s$
- $i \llbracket (CE\text{-op LEq } v1 \ v2) \rrbracket \sim s$
- $i \llbracket (CE\text{-op Eq } v1 \ v2) \rrbracket \sim s$
- $i \llbracket (CE\text{-fst } v) \rrbracket \sim s$
- $i \llbracket (CE\text{-snd } v) \rrbracket \sim s$
- $i \llbracket (CE\text{-concat } v1 \ v2) \rrbracket \sim s$
- $i \llbracket (CE\text{-len } v) \rrbracket \sim s$

**inductive** *eval-c* :: *valuation*  $\Rightarrow$  *c*  $\Rightarrow$  *bool*  $\Rightarrow$  *bool* (  $\langle$  -  $\llbracket$  -  $\rrbracket$   $\sim$  -  $\rangle$  ) **where**

*eval-c-trueI*:  $i \llbracket C\text{-true} \rrbracket \sim \text{True}$

| *eval-c-falseI*:  $i \llbracket C\text{-false} \rrbracket \sim \text{False}$

| *eval-c-conjI*:  $\llbracket i \llbracket c1 \rrbracket \sim b1 ; i \llbracket c2 \rrbracket \sim b2 \rrbracket \Longrightarrow i \llbracket (C\text{-conj } c1 \ c2) \rrbracket \sim (b1 \wedge b2)$

| *eval-c-disjI*:  $\llbracket i \llbracket c1 \rrbracket \sim b1 ; i \llbracket c2 \rrbracket \sim b2 \rrbracket \Longrightarrow i \llbracket (C\text{-disj } c1 \ c2) \rrbracket \sim (b1 \vee b2)$

| *eval-c-impI*:  $\llbracket i \llbracket c1 \rrbracket \sim b1 ; i \llbracket c2 \rrbracket \sim b2 \rrbracket \Longrightarrow i \llbracket (C\text{-imp } c1 \ c2) \rrbracket \sim (b1 \longrightarrow b2)$

| *eval-c-notI*:  $i \llbracket c \rrbracket \sim b \Longrightarrow i \llbracket (C\text{-not } c) \rrbracket \sim (\neg b)$

| *eval-c-eqI*:  $\llbracket i \llbracket e1 \rrbracket \sim sv1 ; i \llbracket e2 \rrbracket \sim sv2 \rrbracket \Longrightarrow i \llbracket (C\text{-eq } e1 \ e2) \rrbracket \sim (sv1 = sv2)$

**equivariance** *eval-c*

**nominal-inductive** *eval-c*  $\langle$ proof $\rangle$

**inductive-cases** *eval-c-elim*:

- $i \llbracket C\text{-true} \rrbracket \sim \text{True}$
- $i \llbracket C\text{-false} \rrbracket \sim \text{False}$
- $i \llbracket (C\text{-conj } c1 \ c2) \rrbracket \sim s$
- $i \llbracket (C\text{-disj } c1 \ c2) \rrbracket \sim s$
- $i \llbracket (C\text{-imp } c1 \ c2) \rrbracket \sim s$
- $i \llbracket (C\text{-not } c) \rrbracket \sim s$
- $i \llbracket (C\text{-eq } e1 \ e2) \rrbracket \sim s$
- $i \llbracket C\text{-true} \rrbracket \sim s$
- $i \llbracket C\text{-false} \rrbracket \sim s$

### 6.1.5 Satisfiability

**inductive** *is-satis* :: *valuation*  $\Rightarrow$  *c*  $\Rightarrow$  *bool* (  $\langle$  -  $\models$  -  $\rangle$  ) **where**

$i \llbracket c \rrbracket \sim \text{True} \Longrightarrow i \models c$

**equivariance** *is-satis*

**nominal-inductive** *is-satis*  $\langle$ proof $\rangle$

**nominal-function** *is-satis-g* :: *valuation*  $\Rightarrow$   $\Gamma$   $\Rightarrow$  *bool* (  $\langle$  -  $\models$  -  $\rangle$  ) **where**

$i \models G\text{Nil} = \text{True}$

|  $i \models ((x, b, c) \#_{\Gamma} G) = (i \models c \wedge i \models G)$

$\langle$ proof $\rangle$

**nominal-termination** (*eqvt*)  $\langle$ proof $\rangle$

## 6.2 Validity

**nominal-function** *valid* ::  $\Theta$   $\Rightarrow$   $\mathcal{B}$   $\Rightarrow$   $\Gamma$   $\Rightarrow$  *c*  $\Rightarrow$  *bool* (  $\langle$  - ; - ; -  $\models$  -  $\rangle$  [50, 50] 50 ) **where**

$P ; B ; G \models c = ( (P ; B ; G \vdash_{wf} c) \wedge (\forall i. (P ; G \vdash i) \wedge i \models G \longrightarrow i \models c) )$

*<proof>*  
**nominal-termination** (*eqvt*) *<proof>*

### 6.3 Lemmas

Lemmas needed for Examples

**lemma** *valid-trueI* [*intro*]:  
  **fixes**  $G::\Gamma$   
  **assumes**  $P ; B \vdash_{wf} G$   
  **shows**  $P ; B ; G \models C\text{-true}$   
*<proof>*  
**end**

# Chapter 7

## Syntax Lemmas

### 7.1 Support, lookup and contexts

**lemma** *supp-v-tau* [*simp*]:

**assumes** *atom z*  $\#$  *v*

**shows**  $\text{supp } (\{ z : b \mid \text{CE-val } (V\text{-var } z) == \text{CE-val } v \}) = \text{supp } v \cup \text{supp } b$

$\langle$ *proof* $\rangle$

**lemma** *supp-v-var-tau* [*simp*]:

**assumes**  $z \neq x$

**shows**  $\text{supp } (\{ z : b \mid \text{CE-val } (V\text{-var } z) == \text{CE-val } (V\text{-var } x) \}) = \{ \text{atom } x \} \cup \text{supp } b$

$\langle$ *proof* $\rangle$

Sometimes we need to work with a version of a binder where the variable is fresh in something else, such as a bigger context. I think these could be generated automatically

**lemma** *obtain-fresh-fun-def*:

**fixes**  $t :: 'b :: \text{fs}$

**shows**  $\exists y :: x. \text{atom } y \# (s, c, \tau, t) \wedge (\text{AF-fundef } f (\text{AF-fun-typ-none } (\text{AF-fun-typ } x \ b \ c \ \tau \ s)) = \text{AF-fundef } f (\text{AF-fun-typ-none } (\text{AF-fun-typ } y \ b \ ((y \leftrightarrow x) \cdot c) \ ((y \leftrightarrow x) \cdot \tau) \ ((y \leftrightarrow x) \cdot s))))$

$\langle$ *proof* $\rangle$

**lemma** *lookup-fun-member*:

**assumes**  $\text{Some } (\text{AF-fundef } f \ ft) = \text{lookup-fun } \Phi \ f$

**shows**  $\text{AF-fundef } f \ ft \in \text{set } \Phi$

$\langle$ *proof* $\rangle$

**lemma** *rig-dom-eq*:

$\text{dom } (G[x \mapsto c]) = \text{dom } G$

$\langle$ *proof* $\rangle$

**lemma** *lookup-in-rig-eq*:

**assumes**  $\text{Some } (b, c) = \text{lookup } \Gamma \ x$

**shows**  $\text{Some } (b, c') = \text{lookup } (\Gamma[x \mapsto c']) \ x$

$\langle$ *proof* $\rangle$

**lemma** *lookup-in-rig-neq*:

**assumes**  $\text{Some } (b, c) = \text{lookup } \Gamma \ y$  **and**  $x \neq y$

**shows**  $\text{Some } (b, c) = \text{lookup } (\Gamma[x \mapsto c']) \ y$

$\langle proof \rangle$

**lemma** *lookup-in-rig*:

**assumes**  $Some (b,c) = lookup \ \Gamma \ y$

**shows**  $\exists c''. Some (b,c'') = lookup (\Gamma[x \mapsto c']) \ y$

$\langle proof \rangle$

**lemma** *lookup-inside[simp]*:

**assumes**  $x \notin fst \ 'toSet \ \Gamma'$

**shows**  $Some (b1,c1) = lookup (\Gamma'@(x,b1,c1)\#_\Gamma \Gamma) \ x$

$\langle proof \rangle$

**lemma** *lookup-inside2*:

**assumes**  $Some (b1,c1) = lookup (\Gamma'@((x,b0,c0)\#_\Gamma \Gamma)) \ y$  **and**  $x \neq y$

**shows**  $Some (b1,c1) = lookup (\Gamma'@((x,b0,c0')\#_\Gamma \Gamma)) \ y$

$\langle proof \rangle$

**fun** *tail*:: 'a list  $\Rightarrow$  'a list **where**

$tail \ [] = []$

|  $tail (x\#xs) = xs$

**lemma** *lookup-options*:

**assumes**  $Some (b,c) = lookup (xt\#_\Gamma G) \ x$

**shows**  $((x,b,c) = xt) \vee (Some (b,c) = lookup G \ x)$

$\langle proof \rangle$

**lemma** *lookup-x*:

**assumes**  $Some (b,c) = lookup G \ x$

**shows**  $x \in fst \ 'toSet \ G$

$\langle proof \rangle$

**lemma** *GCons-eq-appendI*:

**fixes**  $xs1::\Gamma$

**shows**  $[| \ x \ \#_\Gamma \ xs1 = ys; \ xs = xs1 \ @ \ zs \ |] \ ==> \ x \ \#_\Gamma \ xs = ys \ @ \ zs$

$\langle proof \rangle$

**lemma** *split-G*:  $x : toSet \ xs \implies \exists \ ys \ zs. \ xs = ys \ @ \ x \ \#_\Gamma \ zs$

$\langle proof \rangle$

**lemma** *lookup-not-empty*:

**assumes**  $Some \ \tau = lookup \ G \ x$

**shows**  $G \neq GNil$

$\langle proof \rangle$

**lemma** *lookup-in-g*:

**assumes**  $Some (b,c) = lookup \ \Gamma \ x$

**shows**  $(x,b,c) \in toSet \ \Gamma$

$\langle proof \rangle$

**lemma** *lookup-split*:

**fixes**  $\Gamma::\Gamma$

**assumes**  $Some (b,c) = lookup \ \Gamma \ x$

**shows**  $\exists G G'. \Gamma = G'@(x,b,c)\#_{\Gamma} G$   
 ⟨proof⟩

**lemma** *toSet-splitU[simp]*:

$(x',b',c') \in \text{toSet } (\Gamma' @ (x, b, c) \#_{\Gamma} \Gamma) \longleftrightarrow (x',b',c') \in (\text{toSet } \Gamma' \cup \{(x, b, c)\} \cup \text{toSet } \Gamma)$   
 ⟨proof⟩

**lemma** *toSet-splitP[simp]*:

$(\forall (x', b', c') \in \text{toSet } (\Gamma' @ (x, b, c) \#_{\Gamma} \Gamma). P x' b' c') \longleftrightarrow (\forall (x', b', c') \in \text{toSet } \Gamma'. P x' b' c') \wedge P x b c \wedge (\forall (x', b', c') \in \text{toSet } \Gamma. P x' b' c')$  (**is** ?A  $\longleftrightarrow$  ?B)  
 ⟨proof⟩

**lemma** *lookup-restrict*:

**assumes** *Some*  $(b',c') = \text{lookup } (\Gamma'@(x,b,c)\#_{\Gamma}\Gamma) y$  **and**  $x \neq y$   
**shows** *Some*  $(b',c') = \text{lookup } (\Gamma'@\Gamma) y$   
 ⟨proof⟩

**lemma** *supp-list-member*:

**fixes**  $x::'a::fs$  **and**  $l::'a \text{ list}$   
**assumes**  $x \in \text{set } l$   
**shows**  $\text{supp } x \subseteq \text{supp } l$   
 ⟨proof⟩

**lemma** *GNil-append*:

**assumes**  $G\text{Nil} = G1@G2$   
**shows**  $G1 = G\text{Nil} \wedge G2 = G\text{Nil}$   
 ⟨proof⟩

**lemma** *GCons-eq-append-conv*:

**fixes**  $xs::\Gamma$   
**shows**  $x\#_{\Gamma}xs = ys@zs = (ys = G\text{Nil} \wedge x\#_{\Gamma}xs = zs \vee (\exists ys'. x\#_{\Gamma}ys' = ys \wedge xs = ys'@zs))$   
 ⟨proof⟩

**lemma** *dclist-distinct-unique*:

**assumes**  $(dc, \text{const}) \in \text{set } d\text{clist2}$  **and**  $(\text{cons}, \text{const1}) \in \text{set } d\text{clist2}$  **and**  $dc = \text{cons}$  **and** *distinct*  
 (*List.map fst dclist2*)  
**shows**  $(\text{const}) = \text{const1}$   
 ⟨proof⟩

**lemma** *fresh-d-fst-d*:

**assumes**  $\text{atom } u \# \delta$   
**shows**  $u \notin \text{fst } \delta$   
 ⟨proof⟩

**lemma** *bv-not-in-bset-supp*:

**fixes**  $bv::bv$   
**assumes**  $bv \notin B$   
**shows**  $\text{atom } bv \notin \text{supp } B$   
 ⟨proof⟩

**lemma** *u-fresh-d*:

**assumes**  $\text{atom } u \# D$

**shows**  $u \notin \text{fst } \text{setD } D$   
 ⟨proof⟩

## 7.2 Type Definitions

**lemma** *exist-fresh-bv*:

**fixes**  $tm::'a::fs$

**shows**  $\exists bva2\ dclist2. AF\text{-typedef-poly } tyid\ bva\ dclist = AF\text{-typedef-poly } tyid\ bva2\ dclist2 \wedge$   
 $atom\ bva2 \# tm$

⟨proof⟩

**lemma** *obtain-fresh-bv*:

**fixes**  $tm::'a::fs$

**obtains**  $bva2::bv$  **and**  $dclist2$  **where**  $AF\text{-typedef-poly } tyid\ bva\ dclist = AF\text{-typedef-poly } tyid\ bva2$   
 $dclist2 \wedge$

$atom\ bva2 \# tm$

⟨proof⟩

## 7.3 Function Definitions

**lemma** *fun-typ-flip*:

**fixes**  $bv1::bv$  **and**  $c::bv$

**shows**  $(bv1 \leftrightarrow c) \cdot AF\text{-fun-typ } x1\ b1\ c1\ \tau1\ s1 = AF\text{-fun-typ } x1\ ((bv1 \leftrightarrow c) \cdot b1)\ ((bv1 \leftrightarrow c) \cdot c1)$   
 $((bv1 \leftrightarrow c) \cdot \tau1)\ ((bv1 \leftrightarrow c) \cdot s1)$

⟨proof⟩

**lemma** *fun-def-eq*:

**assumes**  $AF\text{-fundef } fa\ (AF\text{-fun-typ-none } (AF\text{-fun-typ } xa\ ba\ ca\ \tau a\ sa)) = AF\text{-fundef } (AF\text{-fun-typ-none } (AF\text{-fun-typ } x\ b\ c\ \tau\ s))$

**shows**  $f = fa$  **and**  $b = ba$  **and**  $[[atom\ xa]]lst. sa = [[atom\ x]]lst. s$  **and**  $[[atom\ xa]]lst. \tau a = [[atom\ x]]lst. \tau$  **and**

$[[atom\ xa]]lst. ca = [[atom\ x]]lst. c$

⟨proof⟩

**lemma** *fun-arg-unique-aux*:

**assumes**  $AF\text{-fun-typ } x1\ b1\ c1\ \tau1'\ s1' = AF\text{-fun-typ } x2\ b2\ c2\ \tau2'\ s2'$

**shows**  $\{ x1 : b1 \mid c1 \} = \{ x2 : b2 \mid c2 \}$

⟨proof⟩

**lemma** *fresh-x-neq*:

**fixes**  $x::x$  **and**  $y::x$

**shows**  $atom\ x \# y = (x \neq y)$

⟨proof⟩

**lemma** *obtain-fresh-z3*:

**fixes**  $tm::'b::fs$

**obtains**  $z::x$  **where**  $\{ x : b \mid c \} = \{ z : b \mid c[x::=V\text{-var } z]_{cv} \} \wedge atom\ z \# tm \wedge atom\ z \# (x,c)$

⟨proof⟩

**lemma** *u-fresh-v*:

**fixes**  $u::u$  **and**  $t::v$

**shows**  $atom\ u \# t$   
 $\langle proof \rangle$

**lemma**  $u\text{-fresh}\text{-ce}$ :  
**fixes**  $u::u$  **and**  $t::ce$   
**shows**  $atom\ u \# t$   
 $\langle proof \rangle$

**lemma**  $u\text{-fresh}\text{-c}$ :  
**fixes**  $u::u$  **and**  $t::c$   
**shows**  $atom\ u \# t$   
 $\langle proof \rangle$

**lemma**  $u\text{-fresh}\text{-g}$ :  
**fixes**  $u::u$  **and**  $t::\Gamma$   
**shows**  $atom\ u \# t$   
 $\langle proof \rangle$

**lemma**  $u\text{-fresh}\text{-t}$ :  
**fixes**  $u::u$  **and**  $t::\tau$   
**shows**  $atom\ u \# t$   
 $\langle proof \rangle$

**lemma**  $b\text{-of}\text{-c}\text{-of}\text{-eq}$ :  
**assumes**  $atom\ z \# \tau$   
**shows**  $\{ z : b\text{-of}\ \tau \mid c\text{-of}\ \tau\ z \} = \tau$   
 $\langle proof \rangle$

**lemma**  $fresh\text{-d}\text{-not}\text{-in}$ :  
**assumes**  $atom\ u2 \# \Delta'$   
**shows**  $u2 \notin fst\ 'setD\ \Delta'$   
 $\langle proof \rangle$

**end**

## Chapter 8

# Wellformedness Lemmas

### 8.1 Prelude

**lemma** *b-of-subst-bb-commute*:

$$(b\text{-of } (\tau[bv::=b]_{\tau b})) = (b\text{-of } \tau)[bv::=b]_{bb}$$

*<proof>*

**lemmas** *wf-intros* = *wfV-wfC-wfG-wfT-wfTs-wfTh-wfB-wfCE-wfTD.intros wfE-wfS-wfCS-wfCSS-wfPhi-wfD-wfFTQ-wfF*

**lemmas** *freshers* = *fresh-prodN b.fresh c.fresh v.fresh ce.fresh fresh-GCons fresh-GNil fresh-at-base*

### 8.2 Strong Elimination

Inversion/elimination for well-formed polymorphic constructors

**lemma** *wf-strong-elim*:

**fixes**  $\Gamma::\Gamma$  **and**  $\Gamma'::\Gamma$  **and**  $v::v$  **and**  $e::e$  **and**  $c::c$  **and**  $\tau::\tau$  **and**  $ts::(\text{string}*\tau)$  *list*  
**and**  $\Delta::\Delta$  **and**  $b::b$  **and**  $ftq::\text{fun-ty-p-q}$  **and**  $ft::\text{fun-ty-p}$  **and**  $ce::ce$  **and**  $td::\text{type-def}$  **and**  $s::s$   
**and**  $tm::'a::fs$

**and**  $cs::\text{branch-s}$  **and**  $css::\text{branch-list}$  **and**  $\Theta::\Theta$

**shows**  $\Theta; \mathcal{B}; \Gamma \vdash_{wf} (V\text{-consp } tyid \text{ dc } b \ v) : b'' \implies (\exists \ bv \ dclist \ x \ b' \ c. b'' = B\text{-app } tyid \ b \ \wedge$

$AF\text{-typedef-poly } tyid \ bv \ dclist \in \text{set } \Theta \ \wedge$

$(dc, \{ \!| \ x : b' \ | \! \} \ c) \in \text{set } dclist \ \wedge$

$\Theta; \mathcal{B} \vdash_{wf} b \ \wedge \text{atom } bv \ \# (\Theta, \mathcal{B}, \Gamma, b, v) \ \wedge \ \Theta; \mathcal{B}; \Gamma \vdash_{wf} v : b'[bv::=b]_{bb} \ \wedge \ \text{atom } bv \ \# \ tm)$

**and**

$\Theta; \mathcal{B}; \Gamma \vdash_{wf} c \implies \text{True}$  **and**

$\Theta; \mathcal{B} \vdash_{wf} \Gamma \implies \text{True}$  **and**

$\Theta; \mathcal{B}; \Gamma \vdash_{wf} \tau \implies \text{True}$  **and**

$\Theta; \mathcal{B}; \Gamma \vdash_{wf} ts \implies \text{True}$  **and**

$\vdash_{wf} \Theta \implies \text{True}$  **and**

$\Theta; \mathcal{B} \vdash_{wf} b \implies \text{True}$  **and**

$\Theta; \mathcal{B}; \Gamma \vdash_{wf} ce : b' \implies \text{True}$  **and**

$\Theta \vdash_{wf} td \implies \text{True}$

*<proof>*

### 8.3 Context Extension

**definition** *wfExt* ::  $\Theta \Rightarrow \mathcal{B} \Rightarrow \Gamma \Rightarrow \Gamma \Rightarrow \text{bool}$  ( $\langle - ; - \vdash_{wf} - \langle - \rangle [50,50,50] 50$ ) **where**

$$wfExt T B G1 G2 = (wfG T B G2 \wedge wfG T B G1 \wedge toSet G1 \subseteq toSet G2)$$

## 8.4 Context

**lemma** *wfG-cons[ms-wb]*:

**fixes**  $\Gamma::\Gamma$

**assumes**  $P; \mathcal{B} \vdash_{wf} (z,b,c) \#_{\Gamma} \Gamma$

**shows**  $P; \mathcal{B} \vdash_{wf} \Gamma \wedge atom z \#_{\Gamma} \Gamma \wedge wfB P \mathcal{B} b$

*<proof>*

**lemma** *wfG-cons2[ms-wb]*:

**fixes**  $\Gamma::\Gamma$

**assumes**  $P; \mathcal{B} \vdash_{wf} zbc \#_{\Gamma} \Gamma$

**shows**  $P; \mathcal{B} \vdash_{wf} \Gamma$

*<proof>*

**lemma** *wf-g-unique*:

**fixes**  $\Gamma::\Gamma$

**assumes**  $\Theta; \mathcal{B} \vdash_{wf} \Gamma$  **and**  $(x,b,c) \in toSet \Gamma$  **and**  $(x,b',c') \in toSet \Gamma$

**shows**  $b=b' \wedge c=c'$

*<proof>*

**lemma** *lookup-if1*:

**fixes**  $\Gamma::\Gamma$

**assumes**  $\Theta; \mathcal{B} \vdash_{wf} \Gamma$  **and**  $Some (b,c) = lookup \Gamma x$

**shows**  $(x,b,c) \in toSet \Gamma \wedge (\forall b' c'. (x,b',c') \in toSet \Gamma \longrightarrow b'=b \wedge c'=c)$

*<proof>*

**lemma** *lookup-if2*:

**assumes**  $wfG P \mathcal{B} \Gamma$  **and**  $(x,b,c) \in toSet \Gamma \wedge (\forall b' c'. (x,b',c') \in toSet \Gamma \longrightarrow b'=b \wedge c'=c)$

**shows**  $Some (b,c) = lookup \Gamma x$

*<proof>*

**lemma** *lookup-iff*:

**fixes**  $\Theta::\Theta$  **and**  $\Gamma::\Gamma$

**assumes**  $\Theta; \mathcal{B} \vdash_{wf} \Gamma$

**shows**  $Some (b,c) = lookup \Gamma x \longleftrightarrow (x,b,c) \in toSet \Gamma \wedge (\forall b' c'. (x,b',c') \in toSet \Gamma \longrightarrow b'=b \wedge c'=c)$

*<proof>*

**lemma** *wfG-lookup-wf*:

**fixes**  $\Theta::\Theta$  **and**  $\Gamma::\Gamma$  **and**  $b::b$  **and**  $\mathcal{B}::\mathcal{B}$

**assumes**  $\Theta; \mathcal{B} \vdash_{wf} \Gamma$  **and**  $Some (b,c) = lookup \Gamma x$

**shows**  $\Theta; \mathcal{B} \vdash_{wf} b$

*<proof>*

**lemma** *wfG-unique*:

**fixes**  $\Gamma::\Gamma$

**assumes**  $wfG B \Theta ((x, b, c) \#_{\Gamma} \Gamma)$  **and**  $(x1,b1,c1) \in toSet ((x, b, c) \#_{\Gamma} \Gamma)$  **and**  $x1=x$

**shows**  $b1 = b \wedge c1 = c$

*<proof>*

**lemma** *wfG-unique-full*:

**fixes**  $\Gamma::\Gamma$   
**assumes**  $wfG \Theta B (\Gamma'@(x, b, c) \#_{\Gamma} \Gamma)$  **and**  $(x1, b1, c1) \in toSet (\Gamma'@(x, b, c) \#_{\Gamma} \Gamma)$  **and**  $x1=x$   
**shows**  $b1 = b \wedge c1 = c$   
 $\langle proof \rangle$

## 8.5 Converting between wb forms

We cannot prove wfB properties here for expressions and statements as need some more facts about  $\Phi$  context which we can prove without this lemma. Trying to cram everything into a single large mutually recursive lemma is not a good idea

**lemma**  $wfX-wfY1$ :

**fixes**  $\Gamma::\Gamma$  **and**  $\Gamma':\Gamma$  **and**  $v::v$  **and**  $e::e$  **and**  $c::c$  **and**  $\tau::\tau$  **and**  $ts::(string*\tau)$  *list* **and**  $\Delta::\Delta$  **and**  $s::s$   
**and**  $b::b$  **and**  $ftq::fun-ty-p-q$  **and**  $ft::fun-ty-p$  **and**  $ce::ce$  **and**  $td::type-def$  **and**  $cs::branch-s$   
**and**  $css::branch-list$

**shows**  $wfV-wf: \Theta; \mathcal{B}; \Gamma \vdash_{wf} v : b \implies \Theta; \mathcal{B} \vdash_{wf} \Gamma \wedge \vdash_{wf} \Theta$  **and**  
 $wfC-wf: \Theta; \mathcal{B}; \Gamma \vdash_{wf} c \implies \Theta; \mathcal{B} \vdash_{wf} \Gamma \wedge \vdash_{wf} \Theta$  **and**  
 $wfG-wf: \Theta; \mathcal{B} \vdash_{wf} \Gamma \implies \vdash_{wf} \Theta$  **and**  
 $wfT-wf: \Theta; \mathcal{B}; \Gamma \vdash_{wf} \tau \implies \Theta; \mathcal{B} \vdash_{wf} \Gamma \wedge \vdash_{wf} \Theta \wedge \Theta; \mathcal{B} \vdash_{wf} b\text{-of } \tau$  **and**  
 $wfTs-wf: \Theta; \mathcal{B}; \Gamma \vdash_{wf} ts \implies \Theta; \mathcal{B} \vdash_{wf} \Gamma \wedge \vdash_{wf} \Theta$  **and**  
 $\vdash_{wf} \Theta \implies True$  **and**  
 $wfB-wf: \Theta; \mathcal{B} \vdash_{wf} b \implies \vdash_{wf} \Theta$  **and**  
 $wfCE-wf: \Theta; \mathcal{B}; \Gamma \vdash_{wf} ce : b \implies \Theta; \mathcal{B} \vdash_{wf} \Gamma \wedge \vdash_{wf} \Theta$  **and**  
 $wfTD-wf: \Theta \vdash_{wf} td \implies \vdash_{wf} \Theta$

$\langle proof \rangle$

**lemma**  $wfX-wfY2$ :

**fixes**  $\Gamma::\Gamma$  **and**  $\Gamma':\Gamma$  **and**  $v::v$  **and**  $e::e$  **and**  $c::c$  **and**  $\tau::\tau$  **and**  $ts::(string*\tau)$  *list* **and**  $\Delta::\Delta$  **and**  $s::s$   
**and**  $b::b$  **and**  $ftq::fun-ty-p-q$  **and**  $ft::fun-ty-p$  **and**  $ce::ce$  **and**  $td::type-def$  **and**  $cs::branch-s$   
**and**  $css::branch-list$

**shows**

$wfE-wf: \Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash_{wf} e : b \implies \Theta; \mathcal{B} \vdash_{wf} \Gamma \wedge \Theta; \mathcal{B}; \Gamma \vdash_{wf} \Delta \wedge \vdash_{wf} \Theta \wedge \Theta \vdash_{wf} \Phi$  **and**  
 $wfS-wf: \Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash_{wf} s : b \implies \Theta; \mathcal{B} \vdash_{wf} \Gamma \wedge \Theta; \mathcal{B}; \Gamma \vdash_{wf} \Delta \wedge \vdash_{wf} \Theta \wedge \Theta \vdash_{wf} \Phi$  **and**  
 $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta ; tid ; dc ; t \vdash_{wf} cs : b \implies \Theta; \mathcal{B} \vdash_{wf} \Gamma \wedge \Theta; \mathcal{B}; \Gamma \vdash_{wf} \Delta \wedge \vdash_{wf} \Theta \wedge \Theta \vdash_{wf} \Phi$  **and**  
 $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta ; tid ; dclist \vdash_{wf} css : b \implies \Theta; \mathcal{B} \vdash_{wf} \Gamma \wedge \Theta; \mathcal{B}; \Gamma \vdash_{wf} \Delta \wedge \vdash_{wf} \Theta \wedge \Theta \vdash_{wf} \Phi$  **and**  
 $wfPhi-wf: \Theta \vdash_{wf} (\Phi::\Phi) \implies \vdash_{wf} \Theta$  **and**  
 $wfD-wf: \Theta; \mathcal{B}; \Gamma \vdash_{wf} \Delta \implies \Theta; \mathcal{B} \vdash_{wf} \Gamma \wedge \vdash_{wf} \Theta$  **and**  
 $wfFTQ-wf: \Theta ; \Phi \vdash_{wf} ftq \implies \Theta \vdash_{wf} \Phi \wedge \vdash_{wf} \Theta$  **and**  
 $wfFT-wf: \Theta ; \Phi ; \mathcal{B} \vdash_{wf} ft \implies \Theta \vdash_{wf} \Phi \wedge \vdash_{wf} \Theta$

$\langle proof \rangle$

**lemmas**  $wfX-wfY = wfX-wfY1 \quad wfX-wfY2$

**lemma**  $setD-ConsD$ :

$ut \in setD (ut' \#_{\Delta} D) = (ut = ut' \vee ut \in setD D)$   
 $\langle proof \rangle$

**lemma**  $wfD-wfT$ :

**fixes**  $\Delta::\Delta$  **and**  $\tau::\tau$   
**assumes**  $\Theta; \mathcal{B}; \Gamma \vdash_{wf} \Delta$

**shows**  $\forall (u, \tau) \in \text{setD } \Delta. \Theta; \mathcal{B}; \Gamma \vdash_{wf} \tau$   
 $\langle \text{proof} \rangle$

**lemma** *subst-b-lookup-d*:

**assumes**  $u \notin \text{fst } ' \text{setD } \Delta$

**shows**  $u \notin \text{fst } ' \text{setD } \Delta[bv ::= b]_{\Delta b}$

$\langle \text{proof} \rangle$

**lemma** *wfG-cons-splitI*:

**fixes**  $\Phi :: \Phi$  **and**  $\Gamma :: \Gamma$

**assumes**  $\Theta; \mathcal{B} \vdash_{wf} \Gamma$  **and**  $\text{atom } x \# \Gamma$  **and**  $wfB \Theta \mathcal{B} b$  **and**

$c \in \{ \text{TRUE}, \text{FALSE} \} \longrightarrow \Theta; \mathcal{B} \vdash_{wf} \Gamma$  **and**

$c \notin \{ \text{TRUE}, \text{FALSE} \} \longrightarrow \Theta ; \mathcal{B} ; (x, b, C\text{-true}) \#_{\Gamma} \Gamma \vdash_{wf} c$

**shows**  $\Theta; \mathcal{B} \vdash_{wf} ((x, b, c) \#_{\Gamma} \Gamma)$

$\langle \text{proof} \rangle$

**lemma** *wfG-consI*:

**fixes**  $\Phi :: \Phi$  **and**  $\Gamma :: \Gamma$  **and**  $c :: c$

**assumes**  $\Theta; \mathcal{B} \vdash_{wf} \Gamma$  **and**  $\text{atom } x \# \Gamma$  **and**  $wfB \Theta \mathcal{B} b$  **and**

$\Theta ; \mathcal{B} ; (x, b, C\text{-true}) \#_{\Gamma} \Gamma \vdash_{wf} c$

**shows**  $\Theta ; \mathcal{B} \vdash_{wf} ((x, b, c) \#_{\Gamma} \Gamma)$

$\langle \text{proof} \rangle$

**lemma** *wfG-elim2*:

**fixes**  $c :: c$

**assumes**  $wfG P \mathcal{B} ((x, b, c) \#_{\Gamma} \Gamma)$

**shows**  $P; \mathcal{B} ; (x, b, \text{TRUE}) \#_{\Gamma} \Gamma \vdash_{wf} c \wedge wfB P \mathcal{B} b$

$\langle \text{proof} \rangle$

**lemma** *wfG-cons-wfC*:

**fixes**  $\Gamma :: \Gamma$  **and**  $c :: c$

**assumes**  $\Theta ; \mathcal{B} \vdash_{wf} (x, b, c) \#_{\Gamma} \Gamma$

**shows**  $\Theta ; \mathcal{B} ; ((x, b, \text{TRUE}) \#_{\Gamma} \Gamma) \vdash_{wf} c$

$\langle \text{proof} \rangle$

**lemma** *wfG-wfB*:

**assumes**  $wfG P \mathcal{B} \Gamma$  **and**  $b \in \text{fst'snd'toSet } \Gamma$

**shows**  $wfB P \mathcal{B} b$

$\langle \text{proof} \rangle$

**lemma** *wfG-cons-TRUE*:

**fixes**  $\Gamma :: \Gamma$  **and**  $b :: b$

**assumes**  $P; \mathcal{B} \vdash_{wf} \Gamma$  **and**  $\text{atom } z \# \Gamma$  **and**  $P; \mathcal{B} \vdash_{wf} b$

**shows**  $P ; \mathcal{B} \vdash_{wf} (z, b, \text{TRUE}) \#_{\Gamma} \Gamma$

$\langle \text{proof} \rangle$

**lemma** *wfG-cons-TRUE2*:

**assumes**  $P; \mathcal{B} \vdash_{wf} (z, b, c) \#_{\Gamma} \Gamma$  **and**  $\text{atom } z \# \Gamma$

**shows**  $P; \mathcal{B} \vdash_{wf} (z, b, \text{TRUE}) \#_{\Gamma} \Gamma$

$\langle \text{proof} \rangle$

**lemma** *wfG-suffix*:

**fixes**  $\Gamma::\Gamma$   
**assumes**  $wfG P \mathcal{B} (\Gamma'\@ \Gamma)$   
**shows**  $wfG P \mathcal{B} \Gamma$   
 $\langle proof \rangle$

**lemma**  $wfV$ - $wfCE$ :  
**fixes**  $v::v$   
**assumes**  $\Theta; \mathcal{B}; \Gamma \vdash_{wf} v : b$   
**shows**  $\Theta; \mathcal{B}; \Gamma \vdash_{wf} CE$ - $val v : b$   
 $\langle proof \rangle$

## 8.6 Support

**lemma**  $wf$ - $supp1$ :  
**fixes**  $\Gamma::\Gamma$  **and**  $\Gamma':\Gamma$  **and**  $v::v$  **and**  $e::e$  **and**  $c::c$  **and**  $\tau::\tau$  **and**  $ts::(string*\tau)$  *list* **and**  $\Delta::\Delta$  **and**  $s::s$  **and**  $b::b$  **and**  $ftq::fun$ - $typ$ - $q$  **and**  $ft::fun$ - $typ$  **and**  $ce::ce$  **and**  $td::type$ - $def$  **and**  $cs::branch$ - $s$  **and**  $css::branch$ - $list$

**shows**  $wfV$ - $supp: \Theta; \mathcal{B}; \Gamma \vdash_{wf} v : b \implies supp v \subseteq atom$ - $dom \Gamma \cup supp \mathcal{B}$  **and**  
 $wfC$ - $supp: \Theta; \mathcal{B}; \Gamma \vdash_{wf} c \implies supp c \subseteq atom$ - $dom \Gamma \cup supp \mathcal{B}$  **and**  
 $wfG$ - $supp: \Theta; \mathcal{B} \vdash_{wf} \Gamma \implies atom$ - $dom \Gamma \subseteq supp \Gamma$  **and**  
 $wfT$ - $supp: \Theta; \mathcal{B}; \Gamma \vdash_{wf} \tau \implies supp \tau \subseteq atom$ - $dom \Gamma \cup supp \mathcal{B}$  **and**  
 $wfTs$ - $supp: \Theta; \mathcal{B}; \Gamma \vdash_{wf} ts \implies supp ts \subseteq atom$ - $dom \Gamma \cup supp \mathcal{B}$  **and**  
 $wfTh$ - $supp: \vdash_{wf} \Theta \implies supp \Theta = \{\}$  **and**  
 $wfB$ - $supp: \Theta; \mathcal{B} \vdash_{wf} b \implies supp b \subseteq supp \mathcal{B}$  **and**  
 $wfCE$ - $supp: \Theta; \mathcal{B}; \Gamma \vdash_{wf} ce : b \implies supp ce \subseteq atom$ - $dom \Gamma \cup supp \mathcal{B}$  **and**  
 $wfTD$ - $supp: \Theta \vdash_{wf} td \implies supp td \subseteq \{\}$

$\langle proof \rangle$

**lemma**  $wf$ - $supp2$ :  
**fixes**  $\Gamma::\Gamma$  **and**  $\Gamma':\Gamma$  **and**  $v::v$  **and**  $e::e$  **and**  $c::c$  **and**  $\tau::\tau$  **and**  
 $ts::(string*\tau)$  *list* **and**  $\Delta::\Delta$  **and**  $s::s$  **and**  $b::b$  **and**  $ftq::fun$ - $typ$ - $q$  **and**  
 $ft::fun$ - $typ$  **and**  $ce::ce$  **and**  $td::type$ - $def$  **and**  $cs::branch$ - $s$  **and**  $css::branch$ - $list$

**shows**  
 $wfE$ - $supp: \Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash_{wf} e : b \implies (supp e \subseteq atom$ - $dom \Gamma \cup supp \mathcal{B} \cup atom$  'fst'  $setD \Delta)$   
**and**  
 $wfS$ - $supp: \Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash_{wf} s : b \implies supp s \subseteq atom$ - $dom \Gamma \cup atom$  'fst'  $setD \Delta \cup supp \mathcal{B}$   
**and**  
 $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta; tid; dc; t \vdash_{wf} cs : b \implies supp cs \subseteq atom$ - $dom \Gamma \cup atom$  'fst'  $setD \Delta \cup supp \mathcal{B}$  **and**  
 $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta; tid; dclist \vdash_{wf} css : b \implies supp css \subseteq atom$ - $dom \Gamma \cup atom$  'fst'  $setD \Delta \cup supp \mathcal{B}$  **and**  
 $wfPhi$ - $supp: \Theta \vdash_{wf} (\Phi::\Phi) \implies supp \Phi = \{\}$  **and**  
 $wfD$ - $supp: \Theta; \mathcal{B}; \Gamma \vdash_{wf} \Delta \implies supp \Delta \subseteq atom$ 'fst'  $(setD \Delta) \cup atom$ - $dom \Gamma \cup supp \mathcal{B}$  **and**  
 $\Theta; \Phi \vdash_{wf} ftq \implies supp ftq = \{\}$  **and**  
 $\Theta; \Phi; \mathcal{B} \vdash_{wf} ft \implies supp ft \subseteq supp \mathcal{B}$

$\langle proof \rangle$

**lemmas**  $wf$ - $supp = wf$ - $supp1$   $wf$ - $supp2$

**lemma**  $wfV$ - $supp$ - $nil$ :  
**fixes**  $v::v$

**assumes**  $P ; \{\|\} ; GNil \vdash_{wf} v : b$   
**shows**  $supp\ v = \{\}$   
 $\langle proof \rangle$

**lemma** *wfT-TRUE-aux*:  
**assumes**  $wfG\ P\ \mathcal{B}\ \Gamma$  **and**  $atom\ z \# (P, \mathcal{B}, \Gamma)$  **and**  $wfB\ P\ \mathcal{B}\ b$   
**shows**  $wfT\ P\ \mathcal{B}\ \Gamma\ (\{\!| z : b \mid TRUE \!\})$   
 $\langle proof \rangle$

**lemma** *wfT-TRUE*:  
**assumes**  $wfG\ P\ \mathcal{B}\ \Gamma$  **and**  $wfB\ P\ \mathcal{B}\ b$   
**shows**  $wfT\ P\ \mathcal{B}\ \Gamma\ (\{\!| z : b \mid TRUE \!\})$   
 $\langle proof \rangle$

**lemma** *phi-flip-eq*:  
**assumes**  $wfPhi\ T\ P$   
**shows**  $(x \leftrightarrow xa) \cdot P = P$   
 $\langle proof \rangle$

**lemma** *wfC-supp-cons*:  
**fixes**  $c'::c$  **and**  $G::\Gamma$   
**assumes**  $P; \mathcal{B} ; (x', b', TRUE) \#_{\Gamma} G \vdash_{wf} c'$   
**shows**  $supp\ c' \subseteq atom\text{-}dom\ G \cup supp\ x' \cup supp\ \mathcal{B}$  **and**  $supp\ c' \subseteq supp\ G \cup supp\ x' \cup supp\ \mathcal{B}$   
 $\langle proof \rangle$

**lemma** *wfG-dom-supp*:  
**fixes**  $x::x$   
**assumes**  $wfG\ P\ \mathcal{B}\ G$   
**shows**  $atom\ x \in atom\text{-}dom\ G \longleftrightarrow atom\ x \in supp\ G$   
 $\langle proof \rangle$

**lemma** *wfG-atoms-supp-eq* :  
**fixes**  $x::x$   
**assumes**  $wfG\ P\ \mathcal{B}\ G$   
**shows**  $atom\ x \in atom\text{-}dom\ G \longleftrightarrow atom\ x \in supp\ G$   
 $\langle proof \rangle$

**lemma** *beta-flip-eq*:  
**fixes**  $x::x$  **and**  $xa::x$  **and**  $\mathcal{B}::\mathcal{B}$   
**shows**  $(x \leftrightarrow xa) \cdot \mathcal{B} = \mathcal{B}$   
 $\langle proof \rangle$

**lemma** *theta-flip-eq2*:  
**assumes**  $\vdash_{wf}\ \Theta$   
**shows**  $(z \leftrightarrow za) \cdot \Theta = \Theta$   
 $\langle proof \rangle$

**lemma** *theta-flip-eq*:  
**assumes**  $wfTh\ \Theta$   
**shows**  $(x \leftrightarrow xa) \cdot \Theta = \Theta$   
 $\langle proof \rangle$

**lemma** *wfT-wfC*:  
**fixes**  $c::c$   
**assumes**  $\Theta; \mathcal{B}; \Gamma \vdash_{wf} \{ z : b \mid c \}$  **and**  $atom\ z \# \Gamma$   
**shows**  $\Theta; \mathcal{B}; (z, b, TRUE) \#_{\Gamma} \Gamma \vdash_{wf} c$   
 $\langle proof \rangle$

**lemma** *u-not-in-dom-g*:  
**fixes**  $u::u$   
**shows**  $atom\ u \notin atom\text{-}dom\ G$   
 $\langle proof \rangle$

**lemma** *bv-not-in-dom-g*:  
**fixes**  $bv::bv$   
**shows**  $atom\ bv \notin atom\text{-}dom\ G$   
 $\langle proof \rangle$

An important lemma that confirms that  $\Gamma$  does not rely on mutable variables

**lemma** *u-not-in-g*:  
**fixes**  $u::u$   
**assumes**  $wfG\ \Theta\ B\ G$   
**shows**  $atom\ u \notin supp\ G$   
 $\langle proof \rangle$

An important lemma that confirms that types only depend on immutable variables

**lemma** *u-not-in-t*:  
**fixes**  $u::u$   
**assumes**  $wfT\ \Theta\ B\ G\ \tau$   
**shows**  $atom\ u \notin supp\ \tau$   
 $\langle proof \rangle$

**lemma** *wfT-supp-c*:  
**fixes**  $\mathcal{B}::\mathcal{B}$  **and**  $z::x$   
**assumes**  $wfT\ P\ \mathcal{B}\ \Gamma\ (\{ z : b \mid c \})$   
**shows**  $supp\ c - \{ atom\ z \} \subseteq atom\text{-}dom\ \Gamma \cup supp\ \mathcal{B}$   
 $\langle proof \rangle$

**lemma** *wfG-wfC[ms-wb]*:  
**assumes**  $wfG\ P\ \mathcal{B}\ ((x, b, c) \#_{\Gamma} \Gamma)$   
**shows**  $wfC\ P\ \mathcal{B}\ ((x, b, TRUE) \#_{\Gamma} \Gamma)\ c$   
 $\langle proof \rangle$

**lemma** *wfT-wf-cons*:  
**assumes**  $wfT\ P\ \mathcal{B}\ \Gamma\ \{ z : b \mid c \}$  **and**  $atom\ z \# \Gamma$   
**shows**  $wfG\ P\ \mathcal{B}\ ((z, b, c) \#_{\Gamma} \Gamma)$   
 $\langle proof \rangle$

**lemma** *wfV-b-fresh*:  
**fixes**  $b::b$  **and**  $v::v$  **and**  $bv::bv$   
**assumes**  $\Theta; \mathcal{B}; \Gamma \vdash_{wf} v : b$  **and**  $bv \notin \mathcal{B}$   
**shows**  $atom\ bv \# v$   
 $\langle proof \rangle$

**lemma** *wfCE-b-fresh*:  
**fixes**  $b::b$  **and**  $ce::ce$  **and**  $bv::bv$   
**assumes**  $\Theta; \mathcal{B}; \Gamma \vdash_{wf} ce: b$  **and**  $bv \notin \mathcal{B}$   
**shows**  $atom\ bv \# ce$   
 $\langle proof \rangle$

## 8.7 Freshness

**lemma** *wfG-fresh-x*:  
**fixes**  $\Gamma::\Gamma$  **and**  $z::x$   
**assumes**  $\Theta; \mathcal{B} \vdash_{wf} \Gamma$  **and**  $atom\ z \# \Gamma$   
**shows**  $atom\ z \# (\Theta, \mathcal{B}, \Gamma)$   
 $\langle proof \rangle$

**lemma** *wfG-wfT*:  
**assumes**  $wfG\ P\ \mathcal{B}\ ((x, b, c[z::=V-var\ x]_{cv}) \#_{\Gamma} G)$  **and**  $atom\ x \# c$   
**shows**  $P; \mathcal{B}; G \vdash_{wf} \{ z : b \mid c \}$   
 $\langle proof \rangle$

**lemma** *wfT-wfT-if*:  
**assumes**  $wfT\ \Theta\ \mathcal{B}\ \Gamma\ (\{ z2 : b \mid CE-val\ v == CE-val\ (V-lit\ L-false)\ IMP\ c[z::=V-var\ z2]_{cv} \})$   
**and**  $atom\ z2 \# (c, \Gamma)$   
**shows**  $wfT\ \Theta\ \mathcal{B}\ \Gamma\ \{ z : b \mid c \}$   
 $\langle proof \rangle$

**lemma** *wfT-fresh-c*:  
**fixes**  $x::x$   
**assumes**  $wfT\ P\ \mathcal{B}\ \Gamma\ \{ z : b \mid c \}$  **and**  $atom\ x \# \Gamma$  **and**  $x \neq z$   
**shows**  $atom\ x \# c$   
 $\langle proof \rangle$

**lemma** *wfG-x-fresh [simp]*:  
**fixes**  $x::x$   
**assumes**  $wfG\ P\ \mathcal{B}\ G$   
**shows**  $atom\ x \notin atom-dom\ G \longleftrightarrow atom\ x \# G$   
 $\langle proof \rangle$

**lemma** *wfD-x-fresh*:  
**fixes**  $x::x$   
**assumes**  $atom\ x \# \Gamma$  **and**  $wfD\ P\ \mathcal{B}\ \Gamma\ \Delta$   
**shows**  $atom\ x \# \Delta$   
 $\langle proof \rangle$

**lemma** *wfG-fresh-x2*:  
**fixes**  $\Gamma::\Gamma$  **and**  $z::x$  **and**  $\Delta::\Delta$  **and**  $\Phi::\Phi$   
**assumes**  $\Theta; \mathcal{B}; \Gamma \vdash_{wf} \Delta$  **and**  $\Theta \vdash_{wf} \Phi$  **and**  $atom\ z \# \Gamma$   
**shows**  $atom\ z \# (\Theta, \Phi, \mathcal{B}, \Gamma, \Delta)$   
 $\langle proof \rangle$

**lemma** *wfV-x-fresh*:  
**fixes**  $v::v$  **and**  $b::b$  **and**  $\Gamma::\Gamma$  **and**  $x::x$   
**assumes**  $\Theta; \mathcal{B}; \Gamma \vdash_{wf} v : b$  **and**  $atom\ x \# \Gamma$

**shows**  $atom\ x \# v$   
 $\langle proof \rangle$

**lemma**  $wfE\text{-}x\text{-fresh}$ :  
**fixes**  $e::e$  **and**  $b::b$  **and**  $\Gamma::\Gamma$  **and**  $\Delta::\Delta$  **and**  $\Phi::\Phi$  **and**  $x::x$   
**assumes**  $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash_{wf} e : b$  **and**  $atom\ x \# \Gamma$   
**shows**  $atom\ x \# e$   
 $\langle proof \rangle$

**lemma**  $wfT\text{-}x\text{-fresh}$ :  
**fixes**  $\tau::\tau$  **and**  $\Gamma::\Gamma$  **and**  $x::x$   
**assumes**  $\Theta; \mathcal{B}; \Gamma \vdash_{wf} \tau$  **and**  $atom\ x \# \Gamma$   
**shows**  $atom\ x \# \tau$   
 $\langle proof \rangle$

**lemma**  $wfS\text{-}x\text{-fresh}$ :  
**fixes**  $s::s$  **and**  $\Delta::\Delta$  **and**  $x::x$   
**assumes**  $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash_{wf} s : b$  **and**  $atom\ x \# \Gamma$   
**shows**  $atom\ x \# s$   
 $\langle proof \rangle$

**lemma**  $wfTh\text{-}fresh$ :  
**fixes**  $x$   
**assumes**  $wfTh\ T$   
**shows**  $atom\ x \# T$   
 $\langle proof \rangle$

**lemmas**  $wfTh\text{-}x\text{-fresh} = wfTh\text{-}fresh$

**lemma**  $wfPhi\text{-}fresh$ :  
**fixes**  $x$   
**assumes**  $wfPhi\ T\ P$   
**shows**  $atom\ x \# P$   
 $\langle proof \rangle$

**lemmas**  $wfPhi\text{-}x\text{-fresh} = wfPhi\text{-}fresh$

**lemmas**  $wb\text{-}x\text{-fresh} = wfTh\text{-}x\text{-fresh}\ wfPhi\text{-}x\text{-fresh}\ wfD\text{-}x\text{-fresh}\ wfT\text{-}x\text{-fresh}\ wfV\text{-}x\text{-fresh}$

**lemma**  $wfG\text{-}inside\text{-}fresh[ms\text{-}fresh]$ :  
**fixes**  $\Gamma::\Gamma$  **and**  $x::x$   
**assumes**  $wfG\ P\ \mathcal{B}\ (\Gamma'@((x,b,c) \#_{\Gamma}\Gamma))$   
**shows**  $atom\ x \notin atom\text{-}dom\ \Gamma'$   
 $\langle proof \rangle$

**lemma**  $wfG\text{-}inside\text{-}x\text{-in}\text{-}atom\text{-}dom$ :  
**fixes**  $c::c$  **and**  $x::x$  **and**  $\Gamma::\Gamma$   
**shows**  $atom\ x \in atom\text{-}dom\ (\Gamma'@ (x, b, c[z::=V\text{-}var\ x]_{cv}) \#_{\Gamma}\Gamma)$   
 $\langle proof \rangle$

**lemma**  $wfG\text{-}inside\text{-}x\text{-neg}$ :  
**fixes**  $c::c$  **and**  $x::x$  **and**  $\Gamma::\Gamma$  **and**  $G::\Gamma$  **and**  $xa::x$   
**assumes**  $G=(\Gamma'@ (x, b, c[z::=V\text{-}var\ x]_{cv}) \#_{\Gamma}\Gamma)$  **and**  $atom\ xa \# G$  **and**  $\Theta; \mathcal{B} \vdash_{wf} G$

**shows**  $xa \neq x$   
 $\langle proof \rangle$

**lemma** *wfG-inside-x-fresh*:

**fixes**  $c::c$  **and**  $x::x$  **and**  $\Gamma::\Gamma$  **and**  $G::\Gamma$  **and**  $xa::x$   
**assumes**  $G=(\Gamma'@ (x, b, c[z::=V-var x]_{cv}) \#_{\Gamma} \Gamma)$  **and**  $atom\ xa \# G$  **and**  $\Theta; \mathcal{B} \vdash_{wf} G$   
**shows**  $atom\ xa \# x$   
 $\langle proof \rangle$

**lemma** *wfT-nil-supp*:

**fixes**  $t::\tau$   
**assumes**  $\Theta; \{\|\} ; GNil \vdash_{wf} t$   
**shows**  $supp\ t = \{\}$   
 $\langle proof \rangle$

## 8.8 Misc

**lemma** *wfG-cons-append*:

**fixes**  $b'::b$   
**assumes**  $\Theta; \mathcal{B} \vdash_{wf} ((x', b', c') \#_{\Gamma} \Gamma') @ (x, b, c) \#_{\Gamma} \Gamma$   
**shows**  $\Theta; \mathcal{B} \vdash_{wf} (\Gamma' @ (x, b, c) \#_{\Gamma} \Gamma) \wedge atom\ x' \# (\Gamma' @ (x, b, c) \#_{\Gamma} \Gamma) \wedge \Theta; \mathcal{B} \vdash_{wf} b' \wedge x' \neq x$   
 $\langle proof \rangle$

**lemma** *flip-u-eq*:

**fixes**  $u::u$  **and**  $u'::u$  **and**  $\Theta::\Theta$  **and**  $\tau::\tau$   
**assumes**  $\Theta; \mathcal{B}; \Gamma \vdash_{wf} \tau$   
**shows**  $(u \leftrightarrow u') \cdot \tau = \tau$  **and**  $(u \leftrightarrow u') \cdot \Gamma = \Gamma$  **and**  $(u \leftrightarrow u') \cdot \Theta = \Theta$  **and**  $(u \leftrightarrow u') \cdot \mathcal{B} = \mathcal{B}$   
 $\langle proof \rangle$

**lemma** *wfT-wf-cons-flip*:

**fixes**  $c::c$  **and**  $x::x$   
**assumes**  $wfT\ P\ \mathcal{B}\ \Gamma\ \{z : b \mid c\}$  **and**  $atom\ x \# (c, \Gamma)$   
**shows**  $wfG\ P\ \mathcal{B}\ ((x, b, c[z::=V-var x]_{cv}) \#_{\Gamma} \Gamma)$   
 $\langle proof \rangle$

## 8.9 Context Strengthening

We can remove an entry for a variable from the context if the variable doesn't appear in the term and the variable is not used later in the context or any other context

**lemma** *fresh-restrict*:

**fixes**  $y::'a::at-base$  **and**  $\Gamma::\Gamma$   
**assumes**  $atom\ y \# (\Gamma' @ (x, b, c) \#_{\Gamma} \Gamma)$   
**shows**  $atom\ y \# (\Gamma' @ \Gamma)$   
 $\langle proof \rangle$

**lemma** *wf-restrict1*:

**fixes**  $\Gamma::\Gamma$  **and**  $\Gamma'::\Gamma$  **and**  $v::v$  **and**  $e::e$  **and**  $c::c$  **and**  $\tau::\tau$  **and**  $ts::(string*\tau)$  *list* **and**  $\Delta::\Delta$  **and**  $s::s$   
**and**  $b::b$  **and**  $ftq::fun-typ-q$  **and**  $ft::fun-typ$  **and**  $ce::ce$  **and**  $td::type-def$   
**and**  $cs::branch-s$  **and**  $css::branch-list$   
**shows**  $\Theta; \mathcal{B}; \Gamma \vdash_{wf} v : b \implies \Gamma = \Gamma_1 @ ((x, b', c') \#_{\Gamma} \Gamma_2) \implies atom\ x \# v \implies atom\ x \# \Gamma_1 \implies \Theta; \mathcal{B}; \Gamma_1 @ \Gamma_2 \vdash_{wf} v : b$  **and**

$\Theta; \mathcal{B}; \Gamma \vdash_{wf} c \implies \Gamma = \Gamma_1 @ ((x, b', c') \#_{\Gamma} \Gamma_2) \implies atom\ x \# c \implies atom\ x \# \Gamma_1 \implies \Theta;$   
 $\mathcal{B}; \Gamma_1 @ \Gamma_2 \vdash_{wf} c$  **and**  
 $\Theta; \mathcal{B} \vdash_{wf} \Gamma \implies \Gamma = \Gamma_1 @ ((x, b', c') \#_{\Gamma} \Gamma_2) \implies atom\ x \# \Gamma_1 \implies \Theta; \mathcal{B} \vdash_{wf} \Gamma_1 @ \Gamma_2$  **and**  
 $\Theta; \mathcal{B}; \Gamma \vdash_{wf} \tau \implies \Gamma = \Gamma_1 @ ((x, b', c') \#_{\Gamma} \Gamma_2) \implies atom\ x \# \tau \implies atom\ x \# \Gamma_1 \implies \Theta;$   
 $\mathcal{B}; \Gamma_1 @ \Gamma_2 \vdash_{wf} \tau$  **and**  
 $\Theta; \mathcal{B}; \Gamma \vdash_{wf} ts \implies True$  **and**  
 $\vdash_{wf} \Theta \implies True$  **and**  
 $\Theta; \mathcal{B} \vdash_{wf} b \implies True$  **and**

$\Theta; \mathcal{B}; \Gamma \vdash_{wf} ce : b \implies \Gamma = \Gamma_1 @ ((x, b', c') \#_{\Gamma} \Gamma_2) \implies atom\ x \# ce \implies atom\ x \# \Gamma_1 \implies \Theta; \mathcal{B};$   
 $\Gamma_1 @ \Gamma_2 \vdash_{wf} ce : b$  **and**  
 $\Theta \vdash_{wf} td \implies True$   
 $\langle proof \rangle$

**lemma** *wf-restrict2*:

**fixes**  $\Gamma::\Gamma$  **and**  $\Gamma':\Gamma$  **and**  $v::v$  **and**  $e::e$  **and**  $c::c$  **and**  $\tau::\tau$  **and**  $ts::(string*\tau)$  *list* **and**  $\Delta::\Delta$  **and**  $s::s$   
**and**  $b::b$  **and**  $ftq::fun\-typ\-q$  **and**  $ft::fun\-typ$  **and**  $ce::ce$  **and**  $td::type\-def$   
**and**  $cs::branch\-s$  **and**  $css::branch\-list$

**shows**  $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash_{wf} e : b \implies \Gamma = \Gamma_1 @ ((x, b', c') \#_{\Gamma} \Gamma_2) \implies atom\ x \# e \implies atom\ x \#$   
 $\Gamma_1 \implies atom\ x \# \Delta \implies \Theta; \Phi; \mathcal{B}; \Gamma_1 @ \Gamma_2; \Delta \vdash_{wf} e : b$  **and**  
 $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash_{wf} s : b \implies True$  **and**  
 $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta; tid; dc; t \vdash_{wf} cs : b \implies True$  **and**  
 $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta; tid; dclist \vdash_{wf} css : b \implies True$  **and**  
 $\Theta \vdash_{wf} (\Phi::\Phi) \implies True$  **and**  
 $\Theta; \mathcal{B}; \Gamma \vdash_{wf} \Delta \implies \Gamma = \Gamma_1 @ ((x, b', c') \#_{\Gamma} \Gamma_2) \implies atom\ x \# \Gamma_1 \implies atom\ x \# \Delta \implies \Theta; \mathcal{B}; \Gamma_1 @ \Gamma_2$   
 $\vdash_{wf} \Delta$  **and**  
 $\Theta; \Phi \vdash_{wf} ftq \implies True$  **and**  
 $\Theta; \Phi; \mathcal{B} \vdash_{wf} ft \implies True$

$\langle proof \rangle$

**lemmas** *wf-restrict=wf-restrict1 wf-restrict2*

**lemma** *wfT-restrict2*:

**fixes**  $\tau::\tau$   
**assumes**  $wfT\ \Theta\ \mathcal{B}\ ((x, b, c) \#_{\Gamma} \Gamma)\ \tau$  **and**  $atom\ x \# \tau$   
**shows**  $\Theta; \mathcal{B}; \Gamma \vdash_{wf} \tau$   
 $\langle proof \rangle$

**lemma** *wfG-intros2*:

**assumes**  $wfC\ P\ \mathcal{B}\ ((x, b, TRUE) \#_{\Gamma} \Gamma)\ c$   
**shows**  $wfG\ P\ \mathcal{B}\ ((x, b, c) \#_{\Gamma} \Gamma)$   
 $\langle proof \rangle$

## 8.10 Type Definitions

**lemma** *wf-theta-weakening1*:

**fixes**  $\Gamma::\Gamma$  **and**  $\Gamma':\Gamma$  **and**  $v::v$  **and**  $e::e$  **and**  $c::c$  **and**  $\tau::\tau$  **and**  $ts::(string*\tau)$  *list* **and**  $\Delta::\Delta$  **and**  $s::s$   
**and**  $b::b$  **and**  $\mathcal{B}::\mathcal{B}$  **and**  $ftq::fun\-typ\-q$  **and**  $ft::fun\-typ$  **and**  $ce::ce$  **and**  $td::type\-def$   
**and**  $cs::branch\-s$  **and**  $css::branch\-list$  **and**  $t::\tau$

**shows**  $\Theta; \mathcal{B}; \Gamma \vdash_{wf} v : b \implies \vdash_{wf} \Theta' \implies set \Theta \subseteq set \Theta' \implies \Theta'; \mathcal{B}; \Gamma \vdash_{wf} v : b$  **and**  
 $\Theta; \mathcal{B}; \Gamma \vdash_{wf} c \implies \vdash_{wf} \Theta' \implies set \Theta \subseteq set \Theta' \implies \Theta'; \mathcal{B}; \Gamma \vdash_{wf} c$  **and**  
 $\Theta; \mathcal{B} \vdash_{wf} \Gamma \implies \vdash_{wf} \Theta' \implies set \Theta \subseteq set \Theta' \implies \Theta'; \mathcal{B} \vdash_{wf} \Gamma$  **and**  
 $\Theta; \mathcal{B}; \Gamma \vdash_{wf} \tau \implies \vdash_{wf} \Theta' \implies set \Theta \subseteq set \Theta' \implies \Theta'; \mathcal{B}; \Gamma \vdash_{wf} \tau$  **and**  
 $\Theta; \mathcal{B}; \Gamma \vdash_{wf} ts \implies \vdash_{wf} \Theta' \implies set \Theta \subseteq set \Theta' \implies \Theta'; \mathcal{B}; \Gamma \vdash_{wf} ts$  **and**  
 $\vdash_{wf} P \implies True$  **and**  
 $\Theta; \mathcal{B} \vdash_{wf} b \implies \vdash_{wf} \Theta' \implies set \Theta \subseteq set \Theta' \implies \Theta'; \mathcal{B} \vdash_{wf} b$  **and**  
 $\Theta; \mathcal{B}; \Gamma \vdash_{wf} ce : b \implies \vdash_{wf} \Theta' \implies set \Theta \subseteq set \Theta' \implies \Theta'; \mathcal{B}; \Gamma \vdash_{wf} ce : b$  **and**  
 $\Theta \vdash_{wf} td \implies \vdash_{wf} \Theta' \implies set \Theta \subseteq set \Theta' \implies \Theta' \vdash_{wf} td$

*<proof>*

**lemma** *wf-theta-weakening2*:

**fixes**  $\Gamma::\Gamma$  **and**  $\Gamma'::\Gamma$  **and**  $v::v$  **and**  $e::e$  **and**  $c::c$  **and**  $\tau::\tau$  **and**  $ts::(string*\tau)$  *list* **and**  $\Delta::\Delta$  **and**  $s::s$   
**and**  $b::b$  **and**  $\mathcal{B}::\mathcal{B}$  **and**  $ftq::fun-typ-q$  **and**  $ft::fun-typ$  **and**  $ce::ce$  **and**  $td::type-def$   
**and**  $cs::branch-s$  **and**  $css::branch-list$  **and**  $t::\tau$

**shows**

$\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash_{wf} e : b \implies \vdash_{wf} \Theta' \implies set \Theta \subseteq set \Theta' \implies \Theta'; \Phi; \mathcal{B}; \Gamma; \Delta \vdash_{wf} e : b$  **and**  
 $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash_{wf} s : b \implies \vdash_{wf} \Theta' \implies set \Theta \subseteq set \Theta' \implies \Theta'; \Phi; \mathcal{B}; \Gamma; \Delta \vdash_{wf} s : b$  **and**  
 $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta; tid; dc; t \vdash_{wf} cs : b \implies \vdash_{wf} \Theta' \implies set \Theta \subseteq set \Theta' \implies \Theta'; \Phi; \mathcal{B}; \Gamma; \Delta;$   
 $tid; dc; t \vdash_{wf} cs : b$  **and**  
 $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta; tid; dclist \vdash_{wf} css : b \implies \vdash_{wf} \Theta' \implies set \Theta \subseteq set \Theta' \implies \Theta'; \Phi; \mathcal{B}; \Gamma; \Delta;$   
 $tid; dclist \vdash_{wf} css : b$  **and**  
 $\Theta \vdash_{wf} (\Phi::\Phi) \implies \vdash_{wf} \Theta' \implies set \Theta \subseteq set \Theta' \implies \Theta' \vdash_{wf} (\Phi::\Phi)$  **and**  
 $\Theta; \mathcal{B}; \Gamma \vdash_{wf} \Delta \implies \vdash_{wf} \Theta' \implies set \Theta \subseteq set \Theta' \implies \Theta'; \mathcal{B}; \Gamma \vdash_{wf} \Delta$  **and**  
 $\Theta; \Phi \vdash_{wf} ftq \implies \vdash_{wf} \Theta' \implies set \Theta \subseteq set \Theta' \implies \Theta'; \Phi \vdash_{wf} ftq$  **and**  
 $\Theta; \Phi; \mathcal{B} \vdash_{wf} ft \implies \vdash_{wf} \Theta' \implies set \Theta \subseteq set \Theta' \implies \Theta'; \Phi; \mathcal{B} \vdash_{wf} ft$

*<proof>*

**lemmas** *wf-theta-weakening* = *wf-theta-weakening1 wf-theta-weakening2*

**lemma** *lookup-wfTD*:

**fixes**  $td::type-def$   
**assumes**  $td \in set \Theta$  **and**  $\vdash_{wf} \Theta$   
**shows**  $\Theta \vdash_{wf} td$

*<proof>*

### 8.10.1 Simple

**lemma** *wfTh-dclist-unique*:

**assumes**  $wfTh \Theta$  **and**  $AF-typedef \ tid \ dclist1 \in set \Theta$  **and**  $AF-typedef \ tid \ dclist2 \in set \Theta$   
**shows**  $dclist1 = dclist2$

*<proof>*

**lemma** *wfTs-ctor-unique*:

**fixes**  $dclist::(string*\tau)$  *list*  
**assumes**  $\Theta; \{\|\}; GNil \vdash_{wf} \ dclist$  **and**  $(c, t1) \in set \ dclist$  **and**  $(c, t2) \in set \ dclist$   
**shows**  $t1 = t2$

*<proof>*

**lemma** *wfTD-ctor-unique*:

**assumes**  $\Theta \vdash_{wf} (AF-typedef \ tid \ dclist)$  **and**  $(c, t1) \in set \ dclist$  **and**  $(c, t2) \in set \ dclist$

**shows**  $t1 = t2$   
 $\langle proof \rangle$

**lemma** *wfTh-ctor-unique*:

**assumes**  $wfTh \Theta$  **and** *AF-typedef*  $tid$   $dclist \in set \Theta$  **and**  $(c, t1) \in set dclist$  **and**  $(c, t2) \in set dclist$

**shows**  $t1 = t2$   
 $\langle proof \rangle$

**lemma** *wfTs-supp-t*:

**fixes**  $dclist::(string*\tau)$  *list*

**assumes**  $(c, t) \in set dclist$  **and**  $\Theta ; B ; GNil \vdash_{wf} dclist$

**shows**  $supp t \subseteq supp B$

$\langle proof \rangle$

**lemma** *wfTh-lookup-supp-empty*:

**fixes**  $t::\tau$

**assumes** *AF-typedef*  $tid$   $dclist \in set \Theta$  **and**  $(c, t) \in set dclist$  **and**  $\vdash_{wf} \Theta$

**shows**  $supp t = \{\}$

$\langle proof \rangle$

**lemma** *wfTh-supp-b*:

**assumes** *AF-typedef*  $tid$   $dclist \in set \Theta$  **and**  $(dc, \{ z : b \mid c \}) \in set dclist$  **and**  $\vdash_{wf} \Theta$

**shows**  $supp b = \{\}$

$\langle proof \rangle$

**lemma** *wfTh-b-eq-iff*:

**fixes**  $bva1::bv$  **and**  $bva2::bv$  **and**  $dc::string$

**assumes**  $(dc, \{ x1 : b1 \mid c1 \}) \in set dclist1$  **and**  $(dc, \{ x2 : b2 \mid c2 \}) \in set dclist2$  **and**

$wfTs P \{ \{ bva1 \} \} GNil dclist1$  **and**  $wfTs P \{ \{ bva2 \} \} GNil dclist2$

$[[atom bva1]]lst.dclist1 = [[atom bva2]]lst.dclist2$

**shows**  $[[atom bva1]]lst. (dc, \{ x1 : b1 \mid c1 \}) = [[atom bva2]]lst. (dc, \{ x2 : b2 \mid c2 \})$

$\langle proof \rangle$

## 8.10.2 Polymorphic

**lemma** *wfTh-wfTs-poly*:

**fixes**  $dclist::(string * \tau)$  *list*

**assumes** *AF-typedef-poly*  $tyid$   $bva$   $dclist \in set P$  **and**  $\vdash_{wf} P$

**shows**  $P ; \{ \{ bva \} \} ; GNil \vdash_{wf} dclist$

$\langle proof \rangle$

**lemma** *wfTh-dclist-poly-unique*:

**assumes**  $wfTh \Theta$  **and** *AF-typedef-poly*  $tid$   $bva$   $dclist1 \in set \Theta$  **and** *AF-typedef-poly*  $tid$   $bva2$   $dclist2 \in set \Theta$

**shows**  $[[atom bva]]lst. dclist1 = [[atom bva2]]lst.dclist2$

$\langle proof \rangle$

**lemma** *wfTh-poly-lookup-supp*:

**fixes**  $t::\tau$

**assumes** *AF-typedef-poly*  $tid$   $bv$   $dclist \in set \Theta$  **and**  $(c, t) \in set dclist$  **and**  $\vdash_{wf} \Theta$

**shows**  $supp t \subseteq \{ atom bv \}$

$\langle proof \rangle$

**lemma** *wfTh-poly-supp-b*:

**assumes** *AF-typedef-poly tid bv dclist*  $\in$  *set*  $\Theta$  **and**  $(dc, \{ z : b \mid c \}) \in$  *set dclist* **and**  $\vdash_{wf} \Theta$

**shows**  $\text{supp } b \subseteq \{ \text{atom } bv \}$

$\langle$ *proof* $\rangle$

**lemma** *subst-g-inside*:

**fixes**  $x::x$  **and**  $c::c$  **and**  $\Gamma::\Gamma$  **and**  $\Gamma'::\Gamma$

**assumes**  $wfG P \mathcal{B} (\Gamma' @ (x, b, c[z::=V\text{-var } x]_{cv}) \#_{\Gamma} \Gamma)$

**shows**  $(\Gamma' @ (x, b, c[z::=V\text{-var } x]_{cv}) \#_{\Gamma} \Gamma)[x::=v]_{\Gamma v} = (\Gamma'[x::=v]_{\Gamma v} @ \Gamma)$

$\langle$ *proof* $\rangle$

**lemma** *wfTh-td-eq*:

**assumes**  $td1 \in$  *set*  $(td2 \# P)$  **and**  $wfTh (td2 \# P)$  **and** *name-of-type*  $td1 =$  *name-of-type*  $td2$

**shows**  $td1 = td2$

$\langle$ *proof* $\rangle$

**lemma** *wfTh-td-unique*:

**assumes**  $td1 \in$  *set*  $P$  **and**  $td2 \in$  *set*  $P$  **and**  $wfTh P$  **and** *name-of-type*  $td1 =$  *name-of-type*  $td2$

**shows**  $td1 = td2$

$\langle$ *proof* $\rangle$

**lemma** *wfTs-distinct*:

**fixes**  $dclist::(\text{string} * \tau)$  *list*

**assumes**  $\Theta ; B ; GNil \vdash_{wf} dclist$

**shows** *distinct*  $(\text{map } fst \text{ } dclist)$

$\langle$ *proof* $\rangle$

**lemma** *wfTh-dclist-distinct*:

**assumes** *AF-typedef s dclist*  $\in$  *set*  $P$  **and**  $wfTh P$

**shows** *distinct*  $(\text{map } fst \text{ } dclist)$

$\langle$ *proof* $\rangle$

**lemma** *wfTh-dc-t-unique2*:

**assumes** *AF-typedef s dclist'*  $\in$  *set*  $P$  **and**  $(dc, tc') \in$  *set dclist'* **and** *AF-typedef s dclist*  $\in$  *set*  $P$  **and**  $wfTh P$  **and**

$(dc, tc) \in$  *set dclist*

**shows**  $tc = tc'$

$\langle$ *proof* $\rangle$

**lemma** *wfTh-dc-t-unique*:

**assumes** *AF-typedef s dclist'*  $\in$  *set*  $P$  **and**  $(dc, \{ x' : b' \mid c' \}) \in$  *set dclist'* **and** *AF-typedef s dclist*  $\in$  *set*  $P$  **and**  $wfTh P$  **and**

$(dc, \{ x : b \mid c \}) \in$  *set dclist*

**shows**  $\{ x' : b' \mid c' \} = \{ x : b \mid c \}$

$\langle$ *proof* $\rangle$

**lemma** *wfTs-wfT*:

**fixes**  $dclist::(\text{string} * \tau)$  *list* **and**  $t::\tau$

**assumes**  $\Theta ; \mathcal{B} ; GNil \vdash_{wf} dclist$  **and**  $(dc, t) \in$  *set dclist*

**shows**  $\Theta ; \mathcal{B} ; GNil \vdash_{wf} t$

$\langle$ *proof* $\rangle$

**lemma** *wfTh-wfT*:

**fixes**  $t::\tau$

**assumes** *wfTh P* **and** *AF-typedef tid dclist*  $\in$  *set P* **and**  $(dc, t) \in$  *set dclist*

**shows**  $P ; \{\|\} ; GNil \vdash_{wf} t$

*<proof>*

**lemma** *td-lookup-eq-iff*:

**fixes**  $dc :: \text{string}$  **and**  $bva1::bv$  **and**  $bva2::bv$

**assumes**  $[[atom\ bva1]]lst. dclist1 = [[atom\ bva2]]lst. dclist2$  **and**  $(dc, \{ x : b \mid c \}) \in$  *set dclist1*

**shows**  $\exists x2\ b2\ c2. (dc, \{ x2 : b2 \mid c2 \}) \in$  *set dclist2*

*<proof>*

**lemma** *lst-t-b-eq-iff*:

**fixes**  $bva1::bv$  **and**  $bva2::bv$

**assumes**  $[[atom\ bva1]]lst. \{ x1 : b1 \mid c1 \} = [[atom\ bva2]]lst. \{ x2 : b2 \mid c2 \}$

**shows**  $[[atom\ bva1]]lst. b1 = [[atom\ bva2]]lst. b2$

*<proof>*

**lemma** *wfTh-typedef-poly-b-eq-iff*:

**assumes** *AF-typedef-poly tyid bva1 dclist1*  $\in$  *set P* **and**  $(dc, \{ x1 : b1 \mid c1 \}) \in$  *set dclist1*

**and** *AF-typedef-poly tyid bva2 dclist2*  $\in$  *set P* **and**  $(dc, \{ x2 : b2 \mid c2 \}) \in$  *set dclist2* **and**  $\vdash_{wf} P$

**shows**  $b1[bva1::=b]_{bb} = b2[bva2::=b]_{bb}$

*<proof>*

## 8.11 Equivariance Lemmas

**lemma** *x-not-in-u-set[simp]*:

**fixes**  $x::x$  **and**  $us::u\ fset$

**shows**  $atom\ x \notin \text{supp}\ us$

*<proof>*

**lemma** *wfS-flip-eq*:

**fixes**  $s1::s$  **and**  $x1::x$  **and**  $s2::s$  **and**  $x2::x$  **and**  $\Delta::\Delta$

**assumes**  $[[atom\ x1]]lst. s1 = [[atom\ x2]]lst. s2$  **and**  $[[atom\ x1]]lst. t1 = [[atom\ x2]]lst. t2$  **and**  $[[atom\ x1]]lst. c1 = [[atom\ x2]]lst. c2$  **and**  $atom\ x2 \# \Gamma$  **and**

$\Theta ; \mathcal{B} ; \Gamma \vdash_{wf} \Delta$  **and**

$\Theta ; \Phi ; \mathcal{B} ; (x1, b, c1) \#_{\Gamma} \Gamma ; \Delta \vdash_{wf} s1 : b\text{-of}\ t1$

**shows**  $\Theta ; \Phi ; \mathcal{B} ; (x2, b, c2) \#_{\Gamma} \Gamma ; \Delta \vdash_{wf} s2 : b\text{-of}\ t2$

*<proof>*

## 8.12 Lookup

**lemma** *wf-not-in-prefix*:

**assumes**  $\Theta ; B \vdash_{wf} (\Gamma' @ (x, b1, c1) \#_{\Gamma} \Gamma)$

**shows**  $x \notin \text{fst}\ 'toSet\ \Gamma'$

*<proof>*

**lemma** *lookup-inside-wf[simp]*:

**assumes**  $\Theta ; B \vdash_{wf} (\Gamma' @ (x, b1, c1) \#_{\Gamma} \Gamma)$

**shows**  $Some\ (b1, c1) = \text{lookup}\ (\Gamma' @ (x, b1, c1) \#_{\Gamma} \Gamma)\ x$

*<proof>*

**lemma** *lookup-weakening*:

**fixes**  $\Theta::\Theta$  **and**  $\Gamma::\Gamma$  **and**  $\Gamma'::\Gamma$

**assumes** *Some*  $(b,c) = \text{lookup } \Gamma \ x$  **and**  $\text{toSet } \Gamma \subseteq \text{toSet } \Gamma'$  **and**  $\Theta; \mathcal{B} \vdash_{wf} \Gamma'$  **and**  $\Theta; \mathcal{B} \vdash_{wf} \Gamma$

**shows** *Some*  $(b,c) = \text{lookup } \Gamma' \ x$

*<proof>*

**lemma** *wfPhi-lookup-fun-unique*:

**fixes**  $\Phi::\Phi$

**assumes**  $\Theta \vdash_{wf} \Phi$  **and**  $AF\text{-fundef } f \text{ fd} \in \text{set } \Phi$

**shows** *Some*  $(AF\text{-fundef } f \text{ fd}) = \text{lookup-fun } \Phi \ f$

*<proof>*

**lemma** *lookup-fun-weakening*:

**fixes**  $\Phi'::\Phi$

**assumes** *Some*  $\text{fd} = \text{lookup-fun } \Phi \ f$  **and**  $\text{set } \Phi \subseteq \text{set } \Phi'$  **and**  $\Theta \vdash_{wf} \Phi'$

**shows** *Some*  $\text{fd} = \text{lookup-fun } \Phi' \ f$

*<proof>*

**lemma** *fundef-poly-fresh-bv*:

**assumes** *atom*  $bv2 \ \# \ (bv1, b1, c1, \tau1, s1)$

**shows**  $*$  :  $(AF\text{-fun-typ-some } bv2 \ (AF\text{-fun-typ } x1 \ ((bv1 \leftrightarrow bv2) \cdot b1) \ ((bv1 \leftrightarrow bv2) \cdot c1) \ ((bv1 \leftrightarrow bv2) \cdot \tau1) \ ((bv1 \leftrightarrow bv2) \cdot s1))) = (AF\text{-fun-typ-some } bv1 \ (AF\text{-fun-typ } x1 \ b1 \ c1 \ \tau1 \ s1))$

(**is**  $(AF\text{-fun-typ-some } ?bv \ ?fun\text{-typ} = AF\text{-fun-typ-some } ?bva \ ?fun\text{-typ}a)$ )

*<proof>*

It is possible to collapse some of the easy to prove inductive cases into a single proof at the qed line but this makes it fragile under change. For example, changing the lemma statement might make one of the previously trivial cases non-trivial and so the collapsing needs to be unpacked. Is there a way to find which case has failed in the qed line?

**lemma** *wb-b-weakening1*:

**fixes**  $\Gamma::\Gamma$  **and**  $\Gamma'::\Gamma$  **and**  $v::v$  **and**  $e::e$  **and**  $c::c$  **and**  $\tau::\tau$  **and**  $ts::(\text{string}*\tau)$  *list* **and**  $\Delta::\Delta$  **and**  $s::s$  **and**  $\mathcal{B}::\mathcal{B}$  **and**  $ftq::\text{fun-typ-q}$  **and**  $ft::\text{fun-typ}$  **and**  $ce::ce$  **and**  $td::\text{type-def}$

**and**  $cs::\text{branch-s}$  **and**  $css::\text{branch-list}$

**shows**  $\Theta; \mathcal{B}; \Gamma \vdash_{wf} v : b \implies \mathcal{B} \mid\subseteq \mathcal{B}' \implies \Theta; \mathcal{B}'; \Gamma \vdash_{wf} v : b$  **and**

$\Theta; \mathcal{B}; \Gamma \vdash_{wf} c \implies \mathcal{B} \mid\subseteq \mathcal{B}' \implies \Theta; \mathcal{B}'; \Gamma \vdash_{wf} c$  **and**

$\Theta; \mathcal{B} \vdash_{wf} \Gamma \implies \mathcal{B} \mid\subseteq \mathcal{B}' \implies \Theta; \mathcal{B}' \vdash_{wf} \Gamma$  **and**

$\Theta; \mathcal{B}; \Gamma \vdash_{wf} \tau \implies \mathcal{B} \mid\subseteq \mathcal{B}' \implies \Theta; \mathcal{B}'; \Gamma \vdash_{wf} \tau$  **and**

$\Theta; \mathcal{B}; \Gamma \vdash_{wf} ts \implies \mathcal{B} \mid\subseteq \mathcal{B}' \implies \Theta; \mathcal{B}'; \Gamma \vdash_{wf} ts$  **and**

$\vdash_{wf} P \implies \text{True}$  **and**

$wfB \ \Theta \ \mathcal{B} \ b \implies \mathcal{B} \mid\subseteq \mathcal{B}' \implies wfB \ \Theta \ \mathcal{B}' \ b$  **and**

$\Theta; \mathcal{B}; \Gamma \vdash_{wf} ce : b \implies \mathcal{B} \mid\subseteq \mathcal{B}' \implies \Theta; \mathcal{B}'; \Gamma \vdash_{wf} ce : b$  **and**

$\Theta \vdash_{wf} td \implies \text{True}$

*<proof>*

**lemma** *wb-b-weakening2*:

**fixes**  $\Gamma::\Gamma$  **and**  $\Gamma'::\Gamma$  **and**  $v::v$  **and**  $e::e$  **and**  $c::c$  **and**  $\tau::\tau$  **and**  $ts::(\text{string}*\tau)$  *list* **and**  $\Delta::\Delta$  **and**  $s::s$  **and**  $\mathcal{B}::\mathcal{B}$  **and**  $ftq::\text{fun-typ-q}$  **and**  $ft::\text{fun-typ}$  **and**  $ce::ce$  **and**  $td::\text{type-def}$

**and**  $cs::\text{branch-s}$  **and**  $css::\text{branch-list}$

**shows**

$\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash_{wf} e : b \implies \mathcal{B} \sqsubseteq \mathcal{B}' \implies \Theta; \Phi; \mathcal{B}'; \Gamma; \Delta \vdash_{wf} e : b$  **and**  
 $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash_{wf} s : b \implies \mathcal{B} \sqsubseteq \mathcal{B}' \implies \Theta; \Phi; \mathcal{B}'; \Gamma; \Delta \vdash_{wf} s : b$  **and**  
 $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta; tid; dc; t \vdash_{wf} cs : b \implies \mathcal{B} \sqsubseteq \mathcal{B}' \implies \Theta; \Phi; \mathcal{B}'; \Gamma; \Delta; tid; dc; t$   
 $\vdash_{wf} cs : b$  **and**  
 $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta; tid; dclist \vdash_{wf} css : b \implies \mathcal{B} \sqsubseteq \mathcal{B}' \implies \Theta; \Phi; \mathcal{B}'; \Gamma; \Delta; tid; dclist$   
 $\vdash_{wf} css : b$  **and**  
 $\Theta \vdash_{wf} (\Phi::\Phi) \implies True$  **and**  
 $\Theta; \mathcal{B}; \Gamma \vdash_{wf} \Delta \implies \mathcal{B} \sqsubseteq \mathcal{B}' \implies \Theta; \mathcal{B}'; \Gamma \vdash_{wf} \Delta$  **and**  
 $\Theta; \Phi \vdash_{wf} ftq \implies True$  **and**  
 $\Theta; \Phi; \mathcal{B} \vdash_{wf} ft \implies \mathcal{B} \sqsubseteq \mathcal{B}' \implies \Theta; \Phi; \mathcal{B}' \vdash_{wf} ft$   
 $\langle proof \rangle$

**lemmas** *wb-b-weakening* = *wb-b-weakening1* *wb-b-weakening2*

**lemma** *wfG-b-weakening*:

**fixes**  $\Gamma::\Gamma$   
**assumes**  $\mathcal{B} \sqsubseteq \mathcal{B}'$  **and**  $\Theta; \mathcal{B} \vdash_{wf} \Gamma$   
**shows**  $\Theta; \mathcal{B}' \vdash_{wf} \Gamma$   
 $\langle proof \rangle$

**lemma** *wfT-b-weakening*:

**fixes**  $\Gamma::\Gamma$  **and**  $\Theta::\Theta$  **and**  $\tau::\tau$   
**assumes**  $\mathcal{B} \sqsubseteq \mathcal{B}'$  **and**  $\Theta; \mathcal{B}; \Gamma \vdash_{wf} \tau$   
**shows**  $\Theta; \mathcal{B}'; \Gamma \vdash_{wf} \tau$   
 $\langle proof \rangle$

**lemma** *wfB-subst-wfB*:

**fixes**  $\tau::\tau$  **and**  $b'::b$  **and**  $b::b$   
**assumes**  $\Theta; \{|bv|\} \vdash_{wf} b$  **and**  $\Theta; \mathcal{B} \vdash_{wf} b'$   
**shows**  $\Theta; \mathcal{B} \vdash_{wf} b[bv::=b']_{bb}$   
 $\langle proof \rangle$

**lemma** *wfT-subst-wfB*:

**fixes**  $\tau::\tau$  **and**  $b'::b$   
**assumes**  $\Theta; \{|bv|\}; (x, b, c) \#_{\Gamma} GNil \vdash_{wf} \tau$  **and**  $\Theta; \mathcal{B} \vdash_{wf} b'$   
**shows**  $\Theta; \mathcal{B} \vdash_{wf} (b\text{-of } \tau)[bv::=b']_{bb}$   
 $\langle proof \rangle$

**lemma** *wfG-cons-unique*:

**assumes**  $(x1, b1, c1) \in toSet ((x, b, c) \#_{\Gamma} \Gamma)$  **and**  $wfG \Theta \mathcal{B} ((x, b, c) \#_{\Gamma} \Gamma)$  **and**  $x = x1$   
**shows**  $b1 = b \wedge c1 = c$   
 $\langle proof \rangle$

**lemma** *wfG-member-unique*:

**assumes**  $(x1, b1, c1) \in toSet (\Gamma' @ ((x, b, c) \#_{\Gamma} \Gamma))$  **and**  $wfG \Theta \mathcal{B} (\Gamma' @ ((x, b, c) \#_{\Gamma} \Gamma))$  **and**  $x = x1$   
**shows**  $b1 = b \wedge c1 = c$   
 $\langle proof \rangle$

## 8.13 Function Definitions

**lemma** *wb-phi-weakening*:

**fixes**  $\Gamma::\Gamma$  **and**  $\Gamma':\Gamma$  **and**  $v::v$  **and**  $e::e$  **and**  $c::c$  **and**  $\tau::\tau$  **and**  $ts::(\text{string}*\tau)$  *list* **and**  $\Delta::\Delta$  **and**  $s::s$   
**and**  $\mathcal{B}::\mathcal{B}$  **and**  $ftq::\text{fun-typ-q}$  **and**  $ft::\text{fun-typ}$  **and**  $ce::ce$  **and**  $td::\text{type-def}$   
**and**  $cs::\text{branch-s}$  **and**  $css::\text{branch-list}$  **and**  $\Phi::\Phi$

**shows**

$\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash_{wf} e : b \implies \Theta \vdash_{wf} \Phi' \implies \text{set } \Phi \subseteq \text{set } \Phi' \implies \Theta; \Phi'; \mathcal{B}; \Gamma; \Delta \vdash_{wf} e : b$   
**and**

$\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash_{wf} s : b \implies \Theta \vdash_{wf} \Phi' \implies \text{set } \Phi \subseteq \text{set } \Phi' \implies \Theta; \Phi'; \mathcal{B}; \Gamma; \Delta \vdash_{wf} s : b$   
**and**

$\Theta; \Phi; \mathcal{B}; \Gamma; \Delta; tid; dc; t \vdash_{wf} cs : b \implies \Theta \vdash_{wf} \Phi' \implies \text{set } \Phi \subseteq \text{set } \Phi' \implies \Theta; \Phi'; \mathcal{B}; \Gamma; \Delta; tid; dc; t \vdash_{wf} cs : b$  **and**

$\Theta; \Phi; \mathcal{B}; \Gamma; \Delta; tid; dclist \vdash_{wf} css : b \implies \Theta \vdash_{wf} \Phi' \implies \text{set } \Phi \subseteq \text{set } \Phi' \implies \Theta; \Phi'; \mathcal{B}; \Gamma; \Delta; tid; dclist \vdash_{wf} css : b$  **and**

$\Theta \vdash_{wf} (\Phi::\Phi) \implies \text{True}$  **and**

$\Theta; \mathcal{B}; \Gamma \vdash_{wf} \Delta \implies \text{True}$  **and**

$\Theta; \Phi \vdash_{wf} ftq \implies \Theta \vdash_{wf} \Phi' \implies \text{set } \Phi \subseteq \text{set } \Phi' \implies \Theta; \Phi' \vdash_{wf} ftq$  **and**

$\Theta; \Phi; \mathcal{B} \vdash_{wf} ft \implies \Theta \vdash_{wf} \Phi' \implies \text{set } \Phi \subseteq \text{set } \Phi' \implies \Theta; \Phi'; \mathcal{B} \vdash_{wf} ft$

*<proof>*

**lemma** *wfT-fun-return-t:*

**fixes**  $\tau a'::\tau$  **and**  $\tau'::\tau$

**assumes**  $\Theta; \mathcal{B}; (xa, b, ca) \#_{\Gamma} GNil \vdash_{wf} \tau a'$  **and**  $(AF\text{-fun-typ } x b c \tau' s') = (AF\text{-fun-typ } xa b ca \tau a' sa')$

**shows**  $\Theta; \mathcal{B}; (x, b, c) \#_{\Gamma} GNil \vdash_{wf} \tau'$

*<proof>*

**lemma** *wfFT-wf-aux:*

**fixes**  $\tau::\tau$  **and**  $\Theta::\Theta$  **and**  $\Phi::\Phi$  **and**  $ft :: \text{fun-typ-q}$  **and**  $s::s$  **and**  $\Delta::\Delta$

**assumes**  $\Theta; \Phi; B \vdash_{wf} (AF\text{-fun-typ } x b c \tau s)$

**shows**  $\Theta; B; (x, b, c) \#_{\Gamma} GNil \vdash_{wf} \tau \wedge \Theta \vdash_{wf} \Phi \wedge \text{supp } s \subseteq \{ \text{atom } x \} \cup \text{supp } B$

*<proof>*

**lemma** *wfFT-simple-wf:*

**fixes**  $\tau::\tau$  **and**  $\Theta::\Theta$  **and**  $\Phi::\Phi$  **and**  $ft :: \text{fun-typ-q}$  **and**  $s::s$  **and**  $\Delta::\Delta$

**assumes**  $\Theta; \Phi \vdash_{wf} (AF\text{-fun-typ-none } (AF\text{-fun-typ } x b c \tau s))$

**shows**  $\Theta; \{|\}\}; (x, b, c) \#_{\Gamma} GNil \vdash_{wf} \tau \wedge \Theta \vdash_{wf} \Phi \wedge \text{supp } s \subseteq \{ \text{atom } x \}$

*<proof>*

**lemma** *wfFT-poly-wf:*

**fixes**  $\tau::\tau$  **and**  $\Theta::\Theta$  **and**  $\Phi::\Phi$  **and**  $ftq :: \text{fun-typ-q}$  **and**  $s::s$  **and**  $\Delta::\Delta$

**assumes**  $\Theta; \Phi \vdash_{wf} (AF\text{-fun-typ-some } bv (AF\text{-fun-typ } x b c \tau s))$

**shows**  $\Theta; \{|bv|\}; (x, b, c) \#_{\Gamma} GNil \vdash_{wf} \tau \wedge \Theta \vdash_{wf} \Phi \wedge \Theta; \Phi; \{|bv|\} \vdash_{wf} (AF\text{-fun-typ } x b c \tau s)$

*<proof>*

**lemma** *wfFT-poly-wfT:*

**fixes**  $\tau::\tau$  **and**  $\Theta::\Theta$  **and**  $\Phi::\Phi$  **and**  $ft :: \text{fun-typ-q}$

**assumes**  $\Theta; \Phi \vdash_{wf} (AF\text{-fun-typ-some } bv (AF\text{-fun-typ } x b c \tau s))$

**shows**  $\Theta; \{|bv|\}; (x, b, c) \#_{\Gamma} GNil \vdash_{wf} \tau$

*<proof>*

**lemma** *b-of-supp:*

$\text{supp } (b\text{-of } t) \subseteq \text{supp } t$

$\langle \text{proof} \rangle$

**lemma** *wfPhi-f-simple-wf*:

**fixes**  $\tau::\tau$  **and**  $\Theta::\Theta$  **and**  $\Phi::\Phi$  **and**  $ft :: \text{fun-ty}p\text{-}q$  **and**  $s::s$  **and**  $\Phi'::\Phi$   
**assumes**  $AF\text{-fund}ef\ f \ (AF\text{-fun-ty}p\text{-none} \ (AF\text{-fun-ty}p \ x \ b \ c \ \tau \ s)) \in \text{set } \Phi$  **and**  $\Theta \vdash_{wf} \Phi$  **and**  $\text{set } \Phi \subseteq \text{set } \Phi'$  **and**  $\Theta \vdash_{wf} \Phi'$   
**shows**  $\Theta ; \{\|\}\ ; (x,b,c) \#_{\Gamma} GNil \vdash_{wf} \tau \wedge \Theta \vdash_{wf} \Phi \wedge \text{supp } s \subseteq \{ \text{atom } x \}$   
 $\langle \text{proof} \rangle$

**lemma** *wfPhi-f-simple-wfT*:

**fixes**  $\tau::\tau$  **and**  $\Theta::\Theta$  **and**  $\Phi::\Phi$  **and**  $ft :: \text{fun-ty}p\text{-}q$   
**assumes**  $\text{Some} \ (AF\text{-fund}ef\ f \ (AF\text{-fun-ty}p\text{-none} \ (AF\text{-fun-ty}p \ x \ b \ c \ \tau \ s))) = \text{lookup-fun } \Phi \ f$  **and**  $\Theta \vdash_{wf} \Phi$   
**shows**  $\Theta ; \{\|\}\ ; (x,b,c) \#_{\Gamma} GNil \vdash_{wf} \tau$   
 $\langle \text{proof} \rangle$

**lemma** *wfPhi-f-simple-supp-b*:

**fixes**  $\tau::\tau$  **and**  $\Theta::\Theta$  **and**  $\Phi::\Phi$  **and**  $ft :: \text{fun-ty}p\text{-}q$   
**assumes**  $\text{Some} \ (AF\text{-fund}ef\ f \ (AF\text{-fun-ty}p\text{-none} \ (AF\text{-fun-ty}p \ x \ b \ c \ \tau \ s))) = \text{lookup-fun } \Phi \ f$  **and**  $\Theta \vdash_{wf} \Phi$   
**shows**  $\text{supp } b = \{\}$   
 $\langle \text{proof} \rangle$

**lemma** *wfPhi-f-simple-supp-t*:

**fixes**  $\tau::\tau$  **and**  $\Theta::\Theta$  **and**  $\Phi::\Phi$  **and**  $ft :: \text{fun-ty}p\text{-}q$   
**assumes**  $\text{Some} \ (AF\text{-fund}ef\ f \ (AF\text{-fun-ty}p\text{-none} \ (AF\text{-fun-ty}p \ x \ b \ c \ \tau \ s))) = \text{lookup-fun } \Phi \ f$  **and**  $\Theta \vdash_{wf} \Phi$   
**shows**  $\text{supp } \tau \subseteq \{ \text{atom } x \}$   
 $\langle \text{proof} \rangle$

**lemma** *wfPhi-f-simple-supp-c*:

**fixes**  $\tau::\tau$  **and**  $\Theta::\Theta$  **and**  $\Phi::\Phi$  **and**  $ft :: \text{fun-ty}p\text{-}q$   
**assumes**  $\text{Some} \ (AF\text{-fund}ef\ f \ (AF\text{-fun-ty}p\text{-none} \ (AF\text{-fun-ty}p \ x \ b \ c \ \tau \ s))) = \text{lookup-fun } \Phi \ f$  **and**  $\Theta \vdash_{wf} \Phi$   
**shows**  $\text{supp } c \subseteq \{ \text{atom } x \}$   
 $\langle \text{proof} \rangle$

**lemma** *wfPhi-f-simple-supp-s*:

**fixes**  $\tau::\tau$  **and**  $\Theta::\Theta$  **and**  $\Phi::\Phi$  **and**  $ft :: \text{fun-ty}p\text{-}q$   
**assumes**  $\text{Some} \ (AF\text{-fund}ef\ f \ (AF\text{-fun-ty}p\text{-none} \ (AF\text{-fun-ty}p \ x \ b \ c \ \tau \ s))) = \text{lookup-fun } \Phi \ f$  **and**  $\Theta \vdash_{wf} \Phi$   
**shows**  $\text{supp } s \subseteq \{ \text{atom } x \}$   
 $\langle \text{proof} \rangle$

**lemma** *wfPhi-f-poly-wf*:

**fixes**  $\tau::\tau$  **and**  $\Theta::\Theta$  **and**  $\Phi::\Phi$  **and**  $ft :: \text{fun-ty}p\text{-}q$  **and**  $s::s$  **and**  $\Phi'::\Phi$   
**assumes**  $AF\text{-fund}ef\ f \ (AF\text{-fun-ty}p\text{-some } bv \ (AF\text{-fun-ty}p \ x \ b \ c \ \tau \ s)) \in \text{set } \Phi$  **and**  $\Theta \vdash_{wf} \Phi$  **and**  $\text{set } \Phi \subseteq \text{set } \Phi'$  **and**  $\Theta \vdash_{wf} \Phi'$   
**shows**  $\Theta ; \{bv\}\ ; (x,b,c) \#_{\Gamma} GNil \vdash_{wf} \tau \wedge \Theta \vdash_{wf} \Phi' \wedge \Theta ; \Phi' ; \{bv\} \vdash_{wf} \ (AF\text{-fun-ty}p \ x \ b \ c \ \tau \ s)$   
 $\langle \text{proof} \rangle$

**lemma** *wfPhi-f-poly-wfT*:

**fixes**  $\tau::\tau$  **and**  $\Theta::\Theta$  **and**  $\Phi::\Phi$  **and**  $ft :: fun\text{-}typ\text{-}q$   
**assumes**  $Some (AF\text{-}fundef f (AF\text{-}fun\text{-}typ\text{-}some bv (AF\text{-}fun\text{-}typ x b c \tau s))) = lookup\text{-}fun \Phi f$  **and**  $\Theta$   
 $\vdash_{wf} \Phi$   
**shows**  $\Theta ; \{ | bv | \} ; (x,b,c) \#_{\Gamma} GNil \vdash_{wf} \tau$   
 $\langle proof \rangle$

**lemma** *wfPhi-f-poly-supp-b*:  
**fixes**  $\tau::\tau$  **and**  $\Theta::\Theta$  **and**  $\Phi::\Phi$  **and**  $ft :: fun\text{-}typ\text{-}q$   
**assumes**  $Some (AF\text{-}fundef f (AF\text{-}fun\text{-}typ\text{-}some bv (AF\text{-}fun\text{-}typ x b c \tau s))) = lookup\text{-}fun \Phi f$  **and**  $\Theta$   
 $\vdash_{wf} \Phi$   
**shows**  $supp b \subseteq supp bv$   
 $\langle proof \rangle$

**lemma** *wfPhi-f-poly-supp-t*:  
**fixes**  $\tau::\tau$  **and**  $\Theta::\Theta$  **and**  $\Phi::\Phi$  **and**  $ft :: fun\text{-}typ\text{-}q$   
**assumes**  $Some (AF\text{-}fundef f (AF\text{-}fun\text{-}typ\text{-}some bv (AF\text{-}fun\text{-}typ x b c \tau s))) = lookup\text{-}fun \Phi f$  **and**  $\Theta$   
 $\vdash_{wf} \Phi$   
**shows**  $supp \tau \subseteq \{ atom x , atom bv \}$   
 $\langle proof \rangle$

**lemma** *wfPhi-f-poly-supp-b-of-t*:  
**fixes**  $\tau::\tau$  **and**  $\Theta::\Theta$  **and**  $\Phi::\Phi$  **and**  $ft :: fun\text{-}typ\text{-}q$   
**assumes**  $Some (AF\text{-}fundef f (AF\text{-}fun\text{-}typ\text{-}some bv (AF\text{-}fun\text{-}typ x b c \tau s))) = lookup\text{-}fun \Phi f$  **and**  $\Theta$   
 $\vdash_{wf} \Phi$   
**shows**  $supp (b\text{-}of \tau) \subseteq \{ atom bv \}$   
 $\langle proof \rangle$

**lemma** *wfPhi-f-poly-supp-c*:  
**fixes**  $\tau::\tau$  **and**  $\Theta::\Theta$  **and**  $\Phi::\Phi$  **and**  $ft :: fun\text{-}typ\text{-}q$   
**assumes**  $Some (AF\text{-}fundef f (AF\text{-}fun\text{-}typ\text{-}some bv (AF\text{-}fun\text{-}typ x b c \tau s))) = lookup\text{-}fun \Phi f$  **and**  $\Theta$   
 $\vdash_{wf} \Phi$   
**shows**  $supp c \subseteq \{ atom x , atom bv \}$   
 $\langle proof \rangle$

**lemma** *wfPhi-f-poly-supp-s*:  
**fixes**  $\tau::\tau$  **and**  $\Theta::\Theta$  **and**  $\Phi::\Phi$  **and**  $ft :: fun\text{-}typ\text{-}q$   
**assumes**  $Some (AF\text{-}fundef f (AF\text{-}fun\text{-}typ\text{-}some bv (AF\text{-}fun\text{-}typ x b c \tau s))) = lookup\text{-}fun \Phi f$  **and**  $\Theta$   
 $\vdash_{wf} \Phi$   
**shows**  $supp s \subseteq \{ atom x , atom bv \}$   
 $\langle proof \rangle$

**lemmas**  $wfPhi\text{-}f\text{-}supp = wfPhi\text{-}f\text{-}poly\text{-}supp\text{-}b \ wfPhi\text{-}f\text{-}simple\text{-}supp\text{-}b \ wfPhi\text{-}f\text{-}poly\text{-}supp\text{-}c$   
 $wfPhi\text{-}f\text{-}simple\text{-}supp\text{-}t \ wfPhi\text{-}f\text{-}poly\text{-}supp\text{-}t \ wfPhi\text{-}f\text{-}simple\text{-}supp\text{-}t \ wfPhi\text{-}f\text{-}poly\text{-}wfT \ wfPhi\text{-}f\text{-}simple\text{-}wfT$   
 $wfPhi\text{-}f\text{-}poly\text{-}supp\text{-}s \ wfPhi\text{-}f\text{-}simple\text{-}supp\text{-}s$

**lemma** *fun-typ-eq-ret-unique*:  
**assumes**  $(AF\text{-}fun\text{-}typ x1 b1 c1 \tau1' s1') = (AF\text{-}fun\text{-}typ x2 b2 c2 \tau2' s2')$   
**shows**  $\tau1 [x1 ::= v]_{\tau v} = \tau2 [x2 ::= v]_{\tau v}$   
 $\langle proof \rangle$

**lemma** *fun-typ-eq-body-unique*:

**fixes**  $v::v$  **and**  $x1::x$  **and**  $x2::x$  **and**  $s1'::s$  **and**  $s2'::s$   
**assumes**  $(AF\text{-fun-ty}p\ x1\ b1\ c1\ \tau1'\ s1') = (AF\text{-fun-ty}p\ x2\ b2\ c2\ \tau2'\ s2')$   
**shows**  $s1'[x1::=v]_{sv} = s2'[x2::=v]_{sv}$   
 $\langle proof \rangle$

**lemma** *fun-ret-unique*:

**assumes**  $Some\ (AF\text{-fundef}\ f\ (AF\text{-fun-ty}p\text{-none}\ (AF\text{-fun-ty}p\ x1\ b1\ c1\ \tau1'\ s1')) = lookup\text{-fun}\ \Phi\ f$  **and**  
 $Some\ (AF\text{-fundef}\ f\ (AF\text{-fun-ty}p\text{-none}\ (AF\text{-fun-ty}p\ x2\ b2\ c2\ \tau2'\ s2')) = lookup\text{-fun}\ \Phi\ f$   
**shows**  $\tau1'[x1::=v]_{\tau v} = \tau2'[x2::=v]_{\tau v}$   
 $\langle proof \rangle$

**lemma** *fun-poly-arg-unique*:

**fixes**  $bv1::bv$  **and**  $bv2::bv$  **and**  $b::b$  **and**  $\tau1::\tau$  **and**  $\tau2::\tau$   
**assumes**  $[[atom\ bv1]]lst.\ (AF\text{-fun-ty}p\ x1\ b1\ c1\ \tau1\ s1) = [[atom\ bv2]]lst.\ (AF\text{-fun-ty}p\ x2\ b2\ c2\ \tau2\ s2)$   
**(is**  $[[atom\ ?x]]lst.\ ?a = [[atom\ ?y]]lst.\ ?b$   
**shows**  $\{ x1 : b1[bv1::=b]_{bb} \mid c1[bv1::=b]_{cb} \} = \{ x2 : b2[bv2::=b]_{bb} \mid c2[bv2::=b]_{cb} \}$   
 $\langle proof \rangle$

**lemma** *fun-poly-ret-unique*:

**assumes**  $Some\ (AF\text{-fundef}\ f\ (AF\text{-fun-ty}p\text{-some}\ bv1\ (AF\text{-fun-ty}p\ x1\ b1\ c1\ \tau1'\ s1')) = lookup\text{-fun}\ \Phi\ f$  **and**  
 $Some\ (AF\text{-fundef}\ f\ (AF\text{-fun-ty}p\text{-some}\ bv2\ (AF\text{-fun-ty}p\ x2\ b2\ c2\ \tau2'\ s2')) = lookup\text{-fun}\ \Phi\ f$   
**shows**  $\tau1'[bv1::=b]_{\tau b}[x1::=v]_{\tau v} = \tau2'[bv2::=b]_{\tau b}[x2::=v]_{\tau v}$   
 $\langle proof \rangle$

**lemma** *fun-poly-body-unique*:

**assumes**  $Some\ (AF\text{-fundef}\ f\ (AF\text{-fun-ty}p\text{-some}\ bv1\ (AF\text{-fun-ty}p\ x1\ b1\ c1\ \tau1'\ s1')) = lookup\text{-fun}\ \Phi\ f$  **and**  
 $Some\ (AF\text{-fundef}\ f\ (AF\text{-fun-ty}p\text{-some}\ bv2\ (AF\text{-fun-ty}p\ x2\ b2\ c2\ \tau2'\ s2')) = lookup\text{-fun}\ \Phi\ f$   
**shows**  $s1'[bv1::=b]_{sb}[x1::=v]_{sv} = s2'[bv2::=b]_{sb}[x2::=v]_{sv}$   
 $\langle proof \rangle$

**lemma** *funtyp-eq-iff-equalities*:

**fixes**  $s'::s$  **and**  $s::s$   
**assumes**  $[[atom\ x]]lst.\ ((c', \tau'), s') = [[atom\ x]]lst.\ ((c, \tau), s)$   
**shows**  $\{ x' : b \mid c' \} = \{ x : b \mid c \} \wedge s'[x'::=v]_{sv} = s[x::=v]_{sv} \wedge \tau'[x'::=v]_{\tau v} = \tau[x::=v]_{\tau v}$   
 $\langle proof \rangle$

## 8.14 Weakening

**lemma** *wfX-wfB1*:

**fixes**  $\Gamma::\Gamma$  **and**  $\Gamma'::\Gamma$  **and**  $v::v$  **and**  $e::e$  **and**  $c::c$  **and**  $\tau::\tau$  **and**  $ts::(string*\tau)$  *list* **and**  $\Delta::\Delta$  **and**  $s::s$   
**and**  $b::b$  **and**  $\mathcal{B}::\mathcal{B}$  **and**  $\Phi::\Phi$  **and**  $ftq::fun\text{-typ}\text{-}q$  **and**  $ft::fun\text{-typ}$  **and**  $ce::ce$  **and**  $td::type\text{-}def$   
**and**  $cs::branch\text{-}s$  **and**  $css::branch\text{-}list$   
**shows**  $wfV\text{-}wfB: \Theta; \mathcal{B}; \Gamma \vdash_{wf} v : b \implies \Theta; \mathcal{B} \vdash_{wf} b$  **and**  
 $\Theta; \mathcal{B}; \Gamma \vdash_{wf} c \implies True$  **and**  
 $\Theta; \mathcal{B} \vdash_{wf} \Gamma \implies True$  **and**  
 $wfT\text{-}wfB: \Theta; \mathcal{B}; \Gamma \vdash_{wf} \tau \implies \Theta; \mathcal{B} \vdash_{wf} b\text{-of}\ \tau$  **and**  
 $\Theta; \mathcal{B}; \Gamma \vdash_{wf} ts \implies True$  **and**  
 $\vdash_{wf} \Theta \implies True$  **and**  
 $\Theta; \mathcal{B} \vdash_{wf} b \implies True$  **and**  
 $wfCE\text{-}wfB: \Theta; \mathcal{B}; \Gamma \vdash_{wf} ce : b \implies \Theta; \mathcal{B} \vdash_{wf} b$  **and**  
 $\Theta \vdash_{wf} td \implies True$

$\langle proof \rangle$

**lemma** *wfX-wfB2*:

**fixes**  $\Gamma::\Gamma$  and  $\Gamma'::\Gamma$  and  $v::v$  and  $e::e$  and  $c::c$  and  $\tau::\tau$  and  $ts::(\text{string}*\tau)$  list and  $\Delta::\Delta$  and  $s::s$   
**and**  $b::b$  and  $\mathcal{B}::\mathcal{B}$  and  $\Phi::\Phi$  and  $ftq::\text{fun-typ-q}$  and  $ft::\text{fun-typ}$  and  $ce::ce$  and  $td::\text{type-def}$   
**and**  $cs::\text{branch-s}$  and  $css::\text{branch-list}$

**shows**

$wfE\text{-}wfB$ :  $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash_{wf} e : b \implies \Theta; \mathcal{B} \vdash_{wf} b$  **and**  
 $wfS\text{-}wfB$ :  $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash_{wf} s : b \implies \Theta; \mathcal{B} \vdash_{wf} b$  **and**  
 $wfCS\text{-}wfB$ :  $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta; tid; dc; t \vdash_{wf} cs : b \implies \Theta; \mathcal{B} \vdash_{wf} b$  **and**  
 $wfCSS\text{-}wfB$ :  $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta; tid; dclist \vdash_{wf} css : b \implies \Theta; \mathcal{B} \vdash_{wf} b$  **and**  
 $\Theta \vdash_{wf} \Phi \implies \text{True}$  **and**  
 $\Theta; \mathcal{B}; \Gamma \vdash_{wf} \Delta \implies \text{True}$  **and**  
 $\Theta; \Phi \vdash_{wf} ftq \implies \text{True}$  **and**  
 $\Theta; \Phi; \mathcal{B} \vdash_{wf} ft \implies \mathcal{B} \sqsubseteq \mathcal{B}' \implies \Theta; \Phi; \mathcal{B}' \vdash_{wf} ft$

*<proof>*

**lemmas**  $wfX\text{-}wfB = wfX\text{-}wfB1$   $wfX\text{-}wfB2$

**lemma** *wf-weakening1*:

**fixes**  $\Gamma::\Gamma$  and  $\Gamma'::\Gamma$  and  $v::v$  and  $e::e$  and  $c::c$  and  $\tau::\tau$  and  $ts::(\text{string}*\tau)$  list and  $\Delta::\Delta$  and  $s::s$   
**and**  $\mathcal{B}::\mathcal{B}$  and  $ftq::\text{fun-typ-q}$  and  $ft::\text{fun-typ}$  and  $ce::ce$  and  $td::\text{type-def}$   
**and**  $cs::\text{branch-s}$  and  $css::\text{branch-list}$

**shows** *wfV-weakening*:  $\Theta; \mathcal{B}; \Gamma \vdash_{wf} v : b \implies \Theta; \mathcal{B} \vdash_{wf} \Gamma' \implies \text{toSet } \Gamma \subseteq \text{toSet } \Gamma' \implies \Theta; \mathcal{B}; \Gamma' \vdash_{wf} v : b$  **and**

*wfC-weakening*:  $\Theta; \mathcal{B}; \Gamma \vdash_{wf} c \implies \Theta; \mathcal{B} \vdash_{wf} \Gamma' \implies \text{toSet } \Gamma \subseteq \text{toSet } \Gamma' \implies \Theta; \mathcal{B}; \Gamma' \vdash_{wf} c$

**and**

$\Theta; \mathcal{B} \vdash_{wf} \Gamma \implies \text{True}$  **and**

*wfT-weakening*:  $\Theta; \mathcal{B}; \Gamma \vdash_{wf} \tau \implies \Theta; \mathcal{B} \vdash_{wf} \Gamma' \implies \text{toSet } \Gamma \subseteq \text{toSet } \Gamma' \implies \Theta; \mathcal{B}; \Gamma' \vdash_{wf} \tau$

**and**

$\Theta; \mathcal{B}; \Gamma \vdash_{wf} ts \implies \text{True}$  **and**

$\vdash_{wf} P \implies \text{True}$  **and**

*wfB-weakening*:  $wfB \Theta \mathcal{B} b \implies \mathcal{B} \sqsubseteq \mathcal{B}' \implies wfB \Theta \mathcal{B}' b$  **and**

*wfCE-weakening*:  $\Theta; \mathcal{B}; \Gamma \vdash_{wf} ce : b \implies \Theta; \mathcal{B} \vdash_{wf} \Gamma' \implies \text{toSet } \Gamma \subseteq \text{toSet } \Gamma' \implies \Theta; \mathcal{B}; \Gamma' \vdash_{wf} ce : b$  **and**

$\Theta \vdash_{wf} td \implies \text{True}$

*<proof>*

**lemma** *wf-weakening2*:

**fixes**  $\Gamma::\Gamma$  and  $\Gamma'::\Gamma$  and  $v::v$  and  $e::e$  and  $c::c$  and  $\tau::\tau$  and  $ts::(\text{string}*\tau)$  list and  $\Delta::\Delta$  and  $s::s$   
**and**  $\mathcal{B}::\mathcal{B}$  and  $ftq::\text{fun-typ-q}$  and  $ft::\text{fun-typ}$  and  $ce::ce$  and  $td::\text{type-def}$   
**and**  $cs::\text{branch-s}$  and  $css::\text{branch-list}$

**shows**

*wfE-weakening*:  $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash_{wf} e : b \implies \Theta; \mathcal{B} \vdash_{wf} \Gamma' \implies \text{toSet } \Gamma \subseteq \text{toSet } \Gamma' \implies \Theta; \Phi; \mathcal{B}; \Gamma'; \Delta \vdash_{wf} e : b$  **and**

*wfS-weakening*:  $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash_{wf} s : b \implies \Theta; \mathcal{B} \vdash_{wf} \Gamma' \implies \text{toSet } \Gamma \subseteq \text{toSet } \Gamma' \implies \Theta; \Phi; \mathcal{B}; \Gamma'; \Delta \vdash_{wf} s : b$  **and**

$\Theta; \Phi; \mathcal{B}; \Gamma; \Delta; tid; dc; t \vdash_{wf} cs : b \implies \Theta; \mathcal{B} \vdash_{wf} \Gamma' \implies \text{toSet } \Gamma \subseteq \text{toSet } \Gamma' \implies \Theta; \Phi; \mathcal{B}; \Gamma'; \Delta; tid; dc; t \vdash_{wf} cs : b$  **and**

$\Theta; \Phi; \mathcal{B}; \Gamma; \Delta; tid; dclist \vdash_{wf} css : b \implies \Theta; \mathcal{B} \vdash_{wf} \Gamma' \implies \text{toSet } \Gamma \subseteq \text{toSet } \Gamma' \implies \Theta; \Phi; \mathcal{B}; \Gamma'; \Delta; tid; dclist \vdash_{wf} css : b$  **and**

$\Theta \vdash_{wf} (\Phi::\Phi) \implies \text{True}$  **and**

*wfD-weakening*:  $\Theta; \mathcal{B}; \Gamma \vdash_{wf} \Delta \implies \Theta; \mathcal{B} \vdash_{wf} \Gamma' \implies toSet \Gamma \subseteq toSet \Gamma' \implies \Theta; \mathcal{B}; \Gamma' \vdash_{wf} \Delta$   
**and**

$\Theta; \Phi \vdash_{wf} ftq \implies True$  **and**  
 $\Theta; \Phi; \mathcal{B} \vdash_{wf} ft \implies True$

*<proof>*

**lemmas** *wf-weakening* = *wf-weakening1 wf-weakening2*

**lemma** *wfV-weakening-cons*:

**fixes**  $\Gamma::\Gamma$  **and**  $\Gamma':\Gamma$  **and**  $v::v$  **and**  $c::c$   
**assumes**  $\Theta; \mathcal{B}; \Gamma \vdash_{wf} v : b$  **and** *atom*  $y \# \Gamma$  **and**  $\Theta; \mathcal{B}; ((y, b', TRUE) \#_{\Gamma} \Gamma) \vdash_{wf} c$   
**shows**  $\Theta; \mathcal{B}; (y, b', c) \#_{\Gamma} \Gamma \vdash_{wf} v : b$

*<proof>*

**lemma** *wfG-cons-weakening*:

**fixes**  $\Gamma':\Gamma$   
**assumes**  $\Theta; \mathcal{B} \vdash_{wf} ((x, b, c) \#_{\Gamma} \Gamma)$  **and**  $\Theta; \mathcal{B} \vdash_{wf} \Gamma'$  **and** *toSet*  $\Gamma \subseteq toSet \Gamma'$  **and** *atom*  $x \# \Gamma'$   
**shows**  $\Theta; \mathcal{B} \vdash_{wf} ((x, b, c) \#_{\Gamma} \Gamma')$

*<proof>*

**lemma** *wfT-weakening-aux*:

**fixes**  $\Gamma::\Gamma$  **and**  $\Gamma':\Gamma$  **and**  $c::c$   
**assumes**  $\Theta; \mathcal{B}; \Gamma \vdash_{wf} \{z : b \mid c\}$  **and**  $\Theta; \mathcal{B} \vdash_{wf} \Gamma'$  **and** *toSet*  $\Gamma \subseteq toSet \Gamma'$  **and** *atom*  $z \# \Gamma'$   
**shows**  $\Theta; \mathcal{B}; \Gamma' \vdash_{wf} \{z : b \mid c\}$

*<proof>*

**lemma** *wfT-weakening-all*:

**fixes**  $\Gamma::\Gamma$  **and**  $\Gamma':\Gamma$  **and**  $\tau::\tau$   
**assumes**  $\Theta; \mathcal{B}; \Gamma \vdash_{wf} \tau$  **and**  $\Theta; \mathcal{B}' \vdash_{wf} \Gamma'$  **and** *toSet*  $\Gamma \subseteq toSet \Gamma'$  **and**  $\mathcal{B} \mid\subseteq \mathcal{B}'$   
**shows**  $\Theta; \mathcal{B}'; \Gamma' \vdash_{wf} \tau$

*<proof>*

**lemma** *wfT-weakening-nil*:

**fixes**  $\Gamma::\Gamma$  **and**  $\Gamma':\Gamma$  **and**  $\tau::\tau$   
**assumes**  $\Theta; \{\|\}; GNil \vdash_{wf} \tau$  **and**  $\Theta; \mathcal{B}' \vdash_{wf} \Gamma'$   
**shows**  $\Theta; \mathcal{B}'; \Gamma' \vdash_{wf} \tau$

*<proof>*

**lemma** *wfTh-wfT2*:

**fixes**  $x::x$  **and**  $v::v$  **and**  $\tau::\tau$  **and**  $G::\Gamma$   
**assumes** *wfTh*  $\Theta$  **and** *AF-typedef*  $s dclist \in set \Theta$  **and**  
 $(dc, \tau) \in set dclist$  **and**  $\Theta; \mathcal{B} \vdash_{wf} G$   
**shows** *supp*  $\tau = \{ \}$  **and**  $\tau[x::=v]_{\tau v} = \tau$  **and** *wfT*  $\Theta \mathcal{B} G \tau$

*<proof>*

**lemma** *wf-d-weakening*:

**fixes**  $\Gamma::\Gamma$  **and**  $\Gamma':\Gamma$  **and**  $v::v$  **and**  $e::e$  **and**  $c::c$  **and**  $\tau::\tau$  **and**  $ts::(string*\tau)$  *list* **and**  $\Delta::\Delta$  **and**  $s::s$   
**and**  $\mathcal{B}::\mathcal{B}$  **and** *ftq::fun-typ-q* **and** *ft::fun-typ* **and**  $ce::ce$  **and** *td::type-def*  
**and**  $cs::branch-s$  **and**  $css::branch-list$

**shows**

$\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash_{wf} e : b \implies \Theta; \mathcal{B}; \Gamma \vdash_{wf} \Delta' \implies setD \Delta \subseteq setD \Delta' \implies \Theta; \Phi; \mathcal{B}; \Gamma; \Delta' \vdash_{wf} e : b$  **and**

$\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash_{wf} s : b \implies \Theta; \mathcal{B}; \Gamma \vdash_{wf} \Delta' \implies \text{setD } \Delta \subseteq \text{setD } \Delta' \implies \Theta; \Phi; \mathcal{B}; \Gamma; \Delta' \vdash_{wf} s : b$  **and**  
 $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta; tid; dc; t \vdash_{wf} cs : b \implies \Theta; \mathcal{B}; \Gamma \vdash_{wf} \Delta' \implies \text{setD } \Delta \subseteq \text{setD } \Delta' \implies \Theta; \Phi; \mathcal{B}; \Gamma; \Delta'; tid; dc; t \vdash_{wf} cs : b$  **and**  
 $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta; tid; dclist \vdash_{wf} css : b \implies \Theta; \mathcal{B}; \Gamma \vdash_{wf} \Delta' \implies \text{setD } \Delta \subseteq \text{setD } \Delta' \implies \Theta; \Phi; \mathcal{B}; \Gamma; \Delta'; tid; dclist \vdash_{wf} css : b$  **and**  
 $\Theta \vdash_{wf} (\Phi::\Phi) \implies \text{True}$  **and**  
 $\Theta; \mathcal{B}; \Gamma \vdash_{wf} \Delta \implies \text{True}$  **and**  
 $\Theta; \Phi \vdash_{wf} ftq \implies \text{True}$  **and**  
 $\Theta; \Phi; \mathcal{B} \vdash_{wf} ft \implies \text{True}$   
 ⟨proof⟩

## 8.15 Useful well-formedness instances

Well-formedness for particular constructs that we will need later

**lemma** *wfC-e-eq*:

**fixes**  $ce::ce$  **and**  $\Gamma::\Gamma$   
**assumes**  $\Theta; \mathcal{B}; \Gamma \vdash_{wf} ce : b$  **and**  $\text{atom } x \# \Gamma$   
**shows**  $\Theta; \mathcal{B}; ((x, b, \text{TRUE}) \#_{\Gamma} \Gamma) \vdash_{wf} (\text{CE-val } (V\text{-var } x) == ce)$   
 ⟨proof⟩

**lemma** *wfC-e-eq2*:

**fixes**  $e1::ce$  **and**  $e2::ce$   
**assumes**  $\Theta; \mathcal{B}; \Gamma \vdash_{wf} e1 : b$  **and**  $\Theta; \mathcal{B}; \Gamma \vdash_{wf} e2 : b$  **and**  $\vdash_{wf} \Theta$  **and**  $\text{atom } x \# \Gamma$   
**shows**  $\Theta; \mathcal{B}; (x, b, (\text{CE-val } (V\text{-var } x)) == e1) \#_{\Gamma} \Gamma \vdash_{wf} (\text{CE-val } (V\text{-var } x)) == e2$   
 ⟨proof⟩

**lemma** *wfT-wfT-if-rev*:

**assumes**  $wfV P \mathcal{B} \Gamma v$  (*base-for-lit l*) **and**  $wfT P \mathcal{B} \Gamma t$  **and**  $\langle \text{atom } z1 \# \Gamma \rangle$   
**shows**  $wfT P \mathcal{B} \Gamma (\{ z1 : b\text{-of } t \mid \text{CE-val } v == \text{CE-val } (V\text{-lit } l) \text{ IMP } (c\text{-of } t z1) \})$   
 ⟨proof⟩

**lemma** *wfT-eq-imp*:

**fixes**  $zz::x$  **and**  $ll::l$  **and**  $\tau'::\tau$   
**assumes** *base-for-lit ll = B-bool* **and**  $\Theta; \{\|\}; GNil \vdash_{wf} \tau'$  **and**  
 $\Theta; \{\|\} \vdash_{wf} (x, b\text{-of } \{ z' : B\text{-bool} \mid \text{TRUE} \}, c\text{-of } \{ z' : B\text{-bool} \mid \text{TRUE} \} x) \#_{\Gamma} GNil$  **and**  
 $\text{atom } zz \# x$   
**shows**  $\Theta; \{\|\}; (x, b\text{-of } \{ z' : B\text{-bool} \mid \text{TRUE} \}, c\text{-of } \{ z' : B\text{-bool} \mid \text{TRUE} \} x) \#_{\Gamma} GNil \vdash_{wf} \{ zz : b\text{-of } \tau' \mid [ [ x ]^v ]^{ce} == [ [ ll ]^v ]^{ce} \text{ IMP } c\text{-of } \tau' zz \}$   
 ⟨proof⟩

**lemma** *wfC-v-eq*:

**fixes**  $ce::ce$  **and**  $\Gamma::\Gamma$  **and**  $v::v$   
**assumes**  $\Theta; \mathcal{B}; \Gamma \vdash_{wf} v : b$  **and**  $\text{atom } x \# \Gamma$   
**shows**  $\Theta; \mathcal{B}; ((x, b, \text{TRUE}) \#_{\Gamma} \Gamma) \vdash_{wf} (\text{CE-val } (V\text{-var } x) == \text{CE-val } v)$   
 ⟨proof⟩

**lemma** *wfT-e-eq*:

**fixes**  $ce::ce$   
**assumes**  $\Theta; \mathcal{B}; \Gamma \vdash_{wf} ce : b$  **and**  $\text{atom } z \# \Gamma$   
**shows**  $\Theta; \mathcal{B}; \Gamma \vdash_{wf} \{ z : b \mid \text{CE-val } (V\text{-var } z) == ce \}$

$\langle proof \rangle$

**lemma** *wfT-v-eq*:

**assumes**  $wfB \Theta \mathcal{B} b$  **and**  $wfV \Theta \mathcal{B} \Gamma v b$  **and**  $atom z \# \Gamma$   
**shows**  $wfT \Theta \mathcal{B} \Gamma \{ z : b \mid C\text{-eq} (CE\text{-val} (V\text{-var} z)) (CE\text{-val} v) \}$

$\langle proof \rangle$

**lemma** *wfC-wfG*:

**fixes**  $\Gamma::\Gamma$  **and**  $c::c$  **and**  $b::b$   
**assumes**  $\Theta ; B ; \Gamma \vdash_{wf} c$  **and**  $\Theta ; B \vdash_{wf} b$  **and**  $atom x \# \Gamma$   
**shows**  $\Theta ; B \vdash_{wf} (x,b,c)\#_{\Gamma} \Gamma$

$\langle proof \rangle$

## 8.16 Replacing the constraint on a variable in a context

**lemma** *wfG-cons-fresh2*:

**fixes**  $\Gamma':\Gamma$   
**assumes**  $wfG P \mathcal{B} ((x',b',c') \#_{\Gamma} \Gamma' @ (x, b, c) \#_{\Gamma} \Gamma)$   
**shows**  $x' \neq x$

$\langle proof \rangle$

**lemma** *replace-in-g-inside*:

**fixes**  $\Gamma::\Gamma$   
**assumes**  $wfG P \mathcal{B} (\Gamma' @ ((x,b0,c0') \#_{\Gamma} \Gamma))$   
**shows**  $replace\text{-in}\text{-}g (\Gamma' @ ((x,b0,c0') \#_{\Gamma} \Gamma)) x c0 = (\Gamma' @ ((x,b0,c0) \#_{\Gamma} \Gamma))$

$\langle proof \rangle$

**lemma** *wfG-supp-rig-eq*:

**fixes**  $\Gamma::\Gamma$   
**assumes**  $wfG P \mathcal{B} (\Gamma'' @ (x, b0, c0) \#_{\Gamma} \Gamma)$  **and**  $wfG P \mathcal{B} (\Gamma'' @ (x, b0, c0') \#_{\Gamma} \Gamma)$   
**shows**  $supp (\Gamma'' @ (x, b0, c0') \#_{\Gamma} \Gamma) \cup supp \mathcal{B} = supp (\Gamma'' @ (x, b0, c0) \#_{\Gamma} \Gamma) \cup supp \mathcal{B}$

$\langle proof \rangle$

**lemma** *fresh-replace-inside[ms-fresh]*:

**fixes**  $y::x$  **and**  $\Gamma::\Gamma$   
**assumes**  $wfG P \mathcal{B} (\Gamma'' @ (x, b, c) \#_{\Gamma} \Gamma)$  **and**  $wfG P \mathcal{B} (\Gamma'' @ (x, b, c') \#_{\Gamma} \Gamma)$   
**shows**  $atom y \# (\Gamma'' @ (x, b, c) \#_{\Gamma} \Gamma) = atom y \# (\Gamma'' @ (x, b, c') \#_{\Gamma} \Gamma)$

$\langle proof \rangle$

**lemma** *wf-replace-inside1*:

**fixes**  $\Gamma::\Gamma$  **and**  $\Phi::\Phi$  **and**  $\Theta::\Theta$  **and**  $\Gamma':\Gamma$  **and**  $v::v$  **and**  $e::e$  **and**  $c::c$  **and**  $c'::c$  **and**  $c'::c$  **and**  $\tau::\tau$   
**and**  $ts::(string*\tau)$  **list** **and**  $\Delta::\Delta$  **and**  $b'::b$  **and**  $b::b$  **and**  $s::s$   
**and**  $ftq::fun\text{-}typ\text{-}q$  **and**  $ft::fun\text{-}typ$  **and**  $ce::ce$  **and**  $td::type\text{-}def$  **and**  $cs::branch\text{-}s$  **and**  $css::branch\text{-}list$

**shows**  $wfV\text{-}replace\text{-}inside: \Theta ; \mathcal{B} ; G \vdash_{wf} v : b' \implies G = (\Gamma' @ (x, b, c') \#_{\Gamma} \Gamma) \implies \Theta ; \mathcal{B} ; ((x,b,TRUE) \#_{\Gamma} \Gamma) \vdash_{wf} c \implies \Theta ; \mathcal{B} ; (\Gamma' @ (x, b, c) \#_{\Gamma} \Gamma) \vdash_{wf} v : b'$  **and**

$wfC\text{-}replace\text{-}inside: \Theta ; \mathcal{B} ; G \vdash_{wf} c'' \implies G = (\Gamma' @ (x, b, c') \#_{\Gamma} \Gamma) \implies \Theta ; \mathcal{B} ; ((x,b,TRUE) \#_{\Gamma} \Gamma) \vdash_{wf} c \implies \Theta ; \mathcal{B} ; (\Gamma' @ (x, b, c) \#_{\Gamma} \Gamma) \vdash_{wf} c''$  **and**

$wfG\text{-}replace\text{-}inside: \Theta ; \mathcal{B} \vdash_{wf} G \implies G = (\Gamma' @ (x, b, c') \#_{\Gamma} \Gamma) \implies \Theta ; \mathcal{B} ; ((x,b,TRUE) \#_{\Gamma} \Gamma) \vdash_{wf} c \implies \Theta ; \mathcal{B} \vdash_{wf} (\Gamma' @ (x, b, c) \#_{\Gamma} \Gamma)$  **and**

$wfT\text{-}replace\text{-}inside: \Theta ; \mathcal{B} ; G \vdash_{wf} \tau \implies G = (\Gamma' @ (x, b, c') \#_{\Gamma} \Gamma) \implies \Theta ; \mathcal{B} ; ((x,b,TRUE) \#_{\Gamma} \Gamma) \vdash_{wf} c \implies \Theta ; \mathcal{B} ; (\Gamma' @ (x, b, c) \#_{\Gamma} \Gamma) \vdash_{wf} \tau$  **and**

$\Theta; \mathcal{B}; \Gamma \vdash_{wf} ts \implies \text{True}$  **and**  
 $\vdash_{wf} P \implies \text{True}$  **and**  
 $\Theta; \mathcal{B} \vdash_{wf} b \implies \text{True}$  **and**  
*wfCE-replace-inside*:  $\Theta; \mathcal{B}; G \vdash_{wf} ce : b' \implies G = (\Gamma' @ (x, b, c') \#_{\Gamma} \Gamma) \implies \Theta; \mathcal{B};$   
 $((x, b, \text{TRUE}) \#_{\Gamma} \Gamma) \vdash_{wf} c \implies \Theta; \mathcal{B}; (\Gamma' @ (x, b, c) \#_{\Gamma} \Gamma) \vdash_{wf} ce : b'$  **and**  
 $\Theta \vdash_{wf} td \implies \text{True}$   
 <proof>

**lemma** *wf-replace-inside2*:

**fixes**  $\Gamma::\Gamma$  **and**  $\Phi::\Phi$  **and**  $\Theta::\Theta$  **and**  $\Gamma':\Gamma$  **and**  $v::v$  **and**  $e::e$  **and**  $c::c$  **and**  $c'':c$  **and**  $c':c$  **and**  $\tau::\tau$   
**and**  $ts::(\text{string}*\tau)$  **list** **and**  $\Delta::\Delta$  **and**  $b':b$  **and**  $b::b$  **and**  $s::s$   
**and**  $ftq::\text{fun-ty-p-q}$  **and**  $ft::\text{fun-ty-p}$  **and**  $ce::ce$  **and**  $td::\text{type-def}$  **and**  $cs::\text{branch-s}$  **and**  $css::\text{branch-list}$   
**shows**

$\Theta; \Phi; \mathcal{B}; G; D \vdash_{wf} e : b' \implies G = (\Gamma' @ (x, b, c') \#_{\Gamma} \Gamma) \implies \Theta; \mathcal{B}; ((x, b, \text{TRUE}) \#_{\Gamma} \Gamma)$   
 $\vdash_{wf} c \implies \Theta; \Phi; \mathcal{B}; (\Gamma' @ (x, b, c) \#_{\Gamma} \Gamma); D \vdash_{wf} e : b'$  **and**  
 $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash_{wf} s : b \implies \text{True}$  **and**  
 $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta; tid; dc; t \vdash_{wf} cs : b \implies \text{True}$  **and**  
 $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta; tid; dclist \vdash_{wf} css : b \implies \text{True}$  **and**  
 $\Theta \vdash_{wf} \Phi \implies \text{True}$  **and**  
 $\Theta; \mathcal{B}; G \vdash_{wf} \Delta \implies G = (\Gamma' @ (x, b, c') \#_{\Gamma} \Gamma) \implies \Theta; \mathcal{B}; ((x, b, \text{TRUE}) \#_{\Gamma} \Gamma) \vdash_{wf} c \implies \Theta;$   
 $\mathcal{B}; (\Gamma' @ (x, b, c) \#_{\Gamma} \Gamma) \vdash_{wf} \Delta$  **and**  
 $\Theta; \Phi \vdash_{wf} ftq \implies \text{True}$  **and**  
 $\Theta; \Phi; \mathcal{B} \vdash_{wf} ft \implies \text{True}$   
 <proof>

**lemmas** *wf-replace-inside* = *wf-replace-inside1* *wf-replace-inside2*

**lemma** *wfC-replace-cons*:

**assumes** *wfG*  $P \mathcal{B} ((x, b, c1) \#_{\Gamma} \Gamma)$  **and** *wfC*  $P \mathcal{B} ((x, b, \text{TRUE}) \#_{\Gamma} \Gamma) c2$   
**shows** *wfC*  $P \mathcal{B} ((x, b, c1) \#_{\Gamma} \Gamma) c2$   
 <proof>

**lemma** *wfC-refl*:

**assumes** *wfG*  $\Theta \mathcal{B} ((x, b', c') \#_{\Gamma} \Gamma)$   
**shows** *wfC*  $\Theta \mathcal{B} ((x, b', c') \#_{\Gamma} \Gamma) c'$   
 <proof>

**lemma** *wfG-wfC-inside*:

**assumes**  $(x, b, c) \in \text{toSet } G$  **and** *wfG*  $\Theta B G$   
**shows** *wfC*  $\Theta B G c$   
 <proof>

**lemma** *wfT-wf-cons3*:

**assumes**  $\Theta; \mathcal{B}; \Gamma \vdash_{wf} \{ z : b \mid c \}$  **and** *atom*  $y \# (c, \Gamma)$   
**shows**  $\Theta; \mathcal{B} \vdash_{wf} (y, b, c[z ::= V\text{-var } y]_{cv}) \#_{\Gamma} \Gamma$   
 <proof>

**lemma** *wfT-wfC-cons*:

**assumes** *wfT*  $P \mathcal{B} \Gamma \{ z1 : b \mid c1 \}$  **and** *wfT*  $P \mathcal{B} \Gamma \{ z2 : b \mid c2 \}$  **and** *atom*  $x \# (c1, c2, \Gamma)$   
**shows** *wfC*  $P \mathcal{B} ((x, b, c1[z1 ::= V\text{-var } x]_v) \#_{\Gamma} \Gamma) (c2[z2 ::= V\text{-var } x]_v)$  **(is** *wfC*  $P \mathcal{B} ?G ?c$ **)**  
 <proof>

**lemma** *wfT-wfC2*:

**fixes**  $c::c$  **and**  $x::x$   
**assumes**  $\Theta; \mathcal{B}; \Gamma \vdash_{wf} \{ z : b \mid c \}$  **and**  $atom\ x \# \Gamma$   
**shows**  $\Theta; \mathcal{B}; (x, b, TRUE) \#_{\Gamma} \Gamma \vdash_{wf} c[z::=[x]^v]_v$

*<proof>*

**lemma** *wfT-wfG*:

**fixes**  $x::x$  **and**  $\Gamma::\Gamma$  **and**  $z::x$  **and**  $c::c$  **and**  $b::b$   
**assumes**  $\Theta; \mathcal{B}; \Gamma \vdash_{wf} \{ z : b \mid c \}$  **and**  $atom\ x \# \Gamma$   
**shows**  $\Theta; \mathcal{B} \vdash_{wf} (x, b, c[z::=[x]^v]_v) \#_{\Gamma} \Gamma$

*<proof>*

**lemma** *wfG-replace-inside2*:

**fixes**  $\Gamma::\Gamma$   
**assumes**  $wfG\ P\ \mathcal{B}\ (\Gamma' @ (x, b, c') \#_{\Gamma} \Gamma)$  **and**  $wfG\ P\ \mathcal{B}\ ((x, b, c) \#_{\Gamma} \Gamma)$   
**shows**  $wfG\ P\ \mathcal{B}\ (\Gamma' @ (x, b, c) \#_{\Gamma} \Gamma)$

*<proof>*

**lemma** *wfG-replace-inside-full*:

**fixes**  $\Gamma::\Gamma$   
**assumes**  $wfG\ P\ \mathcal{B}\ (\Gamma' @ (x, b, c') \#_{\Gamma} \Gamma)$  **and**  $wfG\ P\ \mathcal{B}\ (\Gamma' @ ((x, b, c) \#_{\Gamma} \Gamma))$   
**shows**  $wfG\ P\ \mathcal{B}\ (\Gamma' @ (x, b, c) \#_{\Gamma} \Gamma)$

*<proof>*

**lemma** *wfT-replace-inside2*:

**assumes**  $wfT\ \Theta\ \mathcal{B}\ (\Gamma' @ (x, b, c') \#_{\Gamma} \Gamma)\ t$  **and**  $wfG\ \Theta\ \mathcal{B}\ (\Gamma' @ ((x, b, c) \#_{\Gamma} \Gamma))$   
**shows**  $wfT\ \Theta\ \mathcal{B}\ (\Gamma' @ (x, b, c) \#_{\Gamma} \Gamma)\ t$

*<proof>*

**lemma** *wfD-unique*:

**assumes**  $wfD\ P\ \mathcal{B}\ \Gamma\ \Delta$  **and**  $(u, \tau') \in setD\ \Delta$  **and**  $(u, \tau) \in setD\ \Delta$   
**shows**  $\tau' = \tau$

*<proof>*

**lemma** *replace-in-g-forget*:

**fixes**  $x::x$   
**assumes**  $wfG\ P\ B\ G$   
**shows**  $atom\ x \notin atom\text{-}dom\ G \implies (G[x \mapsto c]) = G$  **and**  
 $atom\ x \# G \implies (G[x \mapsto c]) = G$

*<proof>*

**lemma** *replace-in-g-fresh-single*:

**fixes**  $G::\Gamma$  **and**  $x::x$   
**assumes**  $\langle \Theta; \mathcal{B} \vdash_{wf} G[x' \mapsto c'] \rangle$  **and**  $atom\ x \# G$  **and**  $\langle \Theta; \mathcal{B} \vdash_{wf} G \rangle$   
**shows**  $atom\ x \# G[x' \mapsto c']$

*<proof>*

## 8.17 Preservation of well-formedness under substitution

**lemma** *wfC-cons-switch*:

**fixes**  $c::c$  **and**  $c'::c$   
**assumes**  $\Theta; \mathcal{B}; (x, b, c) \#_{\Gamma} \Gamma \vdash_{wf} c'$

**shows**  $\Theta; \mathcal{B}; (x, b, c') \#_{\Gamma} \Gamma \vdash_{wf} c$   
 $\langle proof \rangle$

**lemma** *subst-g-inside-simple*:

**fixes**  $\Gamma_1::\Gamma$  **and**  $\Gamma_2::\Gamma$

**assumes**  $wfG P \mathcal{B} (\Gamma_1 @ ((x, b, c) \#_{\Gamma} \Gamma_2))$

**shows**  $(\Gamma_1 @ ((x, b, c) \#_{\Gamma} \Gamma_2)) [x ::= v]_{\Gamma_v} = \Gamma_1 [x ::= v]_{\Gamma_v} @ \Gamma_2$

$\langle proof \rangle$

**lemma** *subst-c-TRUE-FALSE*:

**fixes**  $c::c$

**assumes**  $c \notin \{TRUE, FALSE\}$

**shows**  $c [x ::= v]_{cv} \notin \{TRUE, FALSE\}$

$\langle proof \rangle$

**lemma** *lookup-subst*:

**assumes** *Some*  $(b, c) = \text{lookup } \Gamma \ x$  **and**  $x \neq x'$

**shows**  $\exists c'. \text{Some } (b, c') = \text{lookup } \Gamma [x' ::= v]_{\Gamma_v} \ x$

$\langle proof \rangle$

**lemma** *lookup-subst2*:

**assumes** *Some*  $(b, c) = \text{lookup } (\Gamma' @ ((x', b_1, c0 [z0 ::= [x']_{cv}] \#_{\Gamma} \Gamma)) \ x$  **and**  $x \neq x'$  **and**

$\Theta; \mathcal{B} \vdash_{wf} (\Gamma' @ ((x', b_1, c0 [z0 ::= [x']_{cv}] \#_{\Gamma} \Gamma))$

**shows**  $\exists c'. \text{Some } (b, c') = \text{lookup } (\Gamma' [x' ::= v]_{\Gamma_v} @ \Gamma) \ x$

$\langle proof \rangle$

**lemma** *wf-subst1*:

**fixes**  $\Gamma::\Gamma$  **and**  $\Gamma':\Gamma$  **and**  $v::v$  **and**  $e::e$  **and**  $c::c$  **and**  $\tau::\tau$  **and**  $ts::(\text{string}*\tau)$  *list* **and**  $\Delta::\Delta$  **and**  $b::b$   
**and**  $ftq::\text{fun-ty-p-q}$  **and**  $ft::\text{fun-ty-p}$  **and**  $ce::ce$  **and**  $td::\text{type-def}$

**shows**  $wfV\text{-subst}: \Theta; \mathcal{B}; \Gamma \vdash_{wf} v : b \implies \Gamma = \Gamma_1 @ ((x, b', c') \#_{\Gamma} \Gamma_2) \implies \Theta; \mathcal{B}; \Gamma_2 \vdash_{wf} v' : b' \implies$   
 $\Theta; \mathcal{B}; \Gamma [x ::= v]_{\Gamma_v} \vdash_{wf} v [x ::= v]_{vv} : b$  **and**

$wfC\text{-subst}: \Theta; \mathcal{B}; \Gamma \vdash_{wf} c \implies \Gamma = \Gamma_1 @ ((x, b', c') \#_{\Gamma} \Gamma_2) \implies \Theta; \mathcal{B}; \Gamma_2 \vdash_{wf} v' : b' \implies \Theta;$

$\mathcal{B}; \Gamma [x ::= v]_{\Gamma_v} \vdash_{wf} c [x ::= v]_{cv}$  **and**

$wfG\text{-subst}: \Theta; \mathcal{B} \vdash_{wf} \Gamma \implies \Gamma = \Gamma_1 @ ((x, b', c') \#_{\Gamma} \Gamma_2) \implies \Theta; \mathcal{B}; \Gamma_2 \vdash_{wf} v' : b' \implies$

$\Theta; \mathcal{B} \vdash_{wf} \Gamma [x ::= v]_{\Gamma_v}$  **and**

$wfT\text{-subst}: \Theta; \mathcal{B}; \Gamma \vdash_{wf} \tau \implies \Gamma = \Gamma_1 @ ((x, b', c') \#_{\Gamma} \Gamma_2) \implies \Theta; \mathcal{B}; \Gamma_2 \vdash_{wf} v' : b' \implies$

$\Theta; \mathcal{B}; \Gamma [x ::= v]_{\Gamma_v} \vdash_{wf} \tau [x ::= v]_{\tau v}$  **and**

$\Theta; \mathcal{B}; \Gamma \vdash_{wf} ts \implies \text{True}$  **and**

$\vdash_{wf} \Theta \implies \text{True}$  **and**

$\Theta; \mathcal{B} \vdash_{wf} b \implies \text{True}$  **and**

$wfCE\text{-subst}: \Theta; \mathcal{B}; \Gamma \vdash_{wf} ce : b \implies \Gamma = \Gamma_1 @ ((x, b', c') \#_{\Gamma} \Gamma_2) \implies \Theta; \mathcal{B}; \Gamma_2 \vdash_{wf} v' : b' \implies$   
 $\Theta; \mathcal{B}; \Gamma [x ::= v]_{\Gamma_v} \vdash_{wf} ce [x ::= v]_{cev} : b$  **and**

$\Theta \vdash_{wf} td \implies \text{True}$

$\langle proof \rangle$

**lemma** *wf-subst2*:

**fixes**  $\Gamma::\Gamma$  **and**  $\Gamma':\Gamma$  **and**  $v::v$  **and**  $e::e$  **and**  $c::c$  **and**  $\tau::\tau$  **and**  $ts::(\text{string}*\tau)$  *list* **and**  $\Delta::\Delta$  **and**  $b::b$   
**and**  $ftq::\text{fun-ty-p-q}$  **and**  $ft::\text{fun-ty-p}$  **and**  $ce::ce$  **and**  $td::\text{type-def}$

**shows**  $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash_{wf} e : b \implies \Gamma = \Gamma_1 @ ((x, b', c') \#_{\Gamma} \Gamma_2) \implies \Theta; \mathcal{B}; \Gamma_2 \vdash_{wf} v' : b' \implies \Theta;$   
 $\Phi; \mathcal{B}; \Gamma [x ::= v]_{\Gamma_v}; \Delta [x ::= v]_{\Delta_v} \vdash_{wf} e [x ::= v]_{ev} : b$  **and**

$\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash_{wf} s : b \implies \Gamma = \Gamma_1 @ ((x, b', c') \#_{\Gamma} \Gamma_2) \implies \Theta; \mathcal{B}; \Gamma_2 \vdash_{wf} v' : b' \implies \Theta; \Phi;$

$\mathcal{B}; \Gamma [x ::= v]_{\Gamma_v}; \Delta [x ::= v]_{\Delta_v} \vdash_{wf} s [x ::= v]_{sv} : b$  **and**

$\Theta; \Phi; \mathcal{B}; \Gamma; \Delta; tid; dc; t \vdash_{wf} cs : b \implies \Gamma = \Gamma_1 @ ((x, b', c') \#_{\Gamma} \Gamma_2) \implies \Theta; \mathcal{B}; \Gamma_2 \vdash_{wf} v' : b'$   
 $\implies \Theta; \Phi; \mathcal{B}; \Gamma[x::=v']_{\Gamma v}; \Delta[x::=v']_{\Delta v}; tid; dc; t \vdash_{wf} subst\text{-}branchv\ cs\ x\ v' : b$  **and**  
 $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta; tid; dclist \vdash_{wf} css : b \implies \Gamma = \Gamma_1 @ ((x, b', c') \#_{\Gamma} \Gamma_2) \implies \Theta; \mathcal{B}; \Gamma_2 \vdash_{wf} v' : b'$   
 $\implies \Theta; \Phi; \mathcal{B}; \Gamma[x::=v']_{\Gamma v}; \Delta[x::=v']_{\Delta v}; tid; dclist \vdash_{wf} subst\text{-}branchlv\ css\ x\ v' : b$  **and**  
 $\Theta \vdash_{wf} (\Phi::\Phi) \implies True$  **and**  
 $\Theta; \mathcal{B}; \Gamma \vdash_{wf} \Delta \implies \Gamma = \Gamma_1 @ ((x, b', c') \#_{\Gamma} \Gamma_2) \implies \Theta; \mathcal{B}; \Gamma_2 \vdash_{wf} v' : b' \implies \Theta; \mathcal{B}; \Gamma[x::=v']_{\Gamma v}$   
 $\vdash_{wf} \Delta[x::=v']_{\Delta v}$  **and**  
 $\Theta; \Phi \vdash_{wf} ftq \implies True$  **and**  
 $\Theta; \Phi; \mathcal{B} \vdash_{wf} ft \implies True$   
*<proof>*

**lemmas** *wf-subst = wf-subst1 wf-subst2*

**lemma** *wfG-subst-wfV:*

**assumes**  $\Theta; \mathcal{B} \vdash_{wf} \Gamma' @ (x, b, c0[z0::=V\text{-var}\ x]_{cv}) \#_{\Gamma} \Gamma$  **and**  $wfV\ \Theta\ \mathcal{B}\ \Gamma\ v\ b$   
**shows**  $\Theta; \mathcal{B} \vdash_{wf} \Gamma'[x::=v]_{\Gamma v} @ \Gamma$   
*<proof>*

**lemma** *wfG-member-subst:*

**assumes**  $(x1, b1, c1) \in toSet\ (\Gamma' @ \Gamma)$  **and**  $wfG\ \Theta\ \mathcal{B}\ (\Gamma' @ ((x, b, c) \#_{\Gamma} \Gamma))$  **and**  $x \neq x1$   
**shows**  $\exists c1'. (x1, b1, c1') \in toSet\ ((\Gamma'[x::=v]_{\Gamma v}) @ \Gamma)$   
*<proof>*

**lemma** *wfG-member-subst2:*

**assumes**  $(x1, b1, c1) \in toSet\ (\Gamma' @ ((x, b, c) \#_{\Gamma} \Gamma))$  **and**  $wfG\ \Theta\ \mathcal{B}\ (\Gamma' @ ((x, b, c) \#_{\Gamma} \Gamma))$  **and**  $x \neq x1$   
**shows**  $\exists c1'. (x1, b1, c1') \in toSet\ ((\Gamma'[x::=v]_{\Gamma v}) @ \Gamma)$   
*<proof>*

**lemma** *wbc-subst:*

**fixes**  $\Gamma::\Gamma$  **and**  $\Gamma':\Gamma$  **and**  $v::v$   
**assumes**  $wfC\ \Theta\ \mathcal{B}\ (\Gamma' @ ((x, b, c') \#_{\Gamma} \Gamma))\ c$  **and**  $\Theta; \mathcal{B}; \Gamma \vdash_{wf} v : b$   
**shows**  $\Theta; \mathcal{B}; ((\Gamma'[x::=v]_{\Gamma v}) @ \Gamma) \vdash_{wf} c[x::=v]_{cv}$   
*<proof>*

**lemma** *wfG-inside-fresh-suffix:*

**assumes**  $wfG\ P\ B\ (\Gamma' @ (x, b, c) \#_{\Gamma} \Gamma)$   
**shows**  $atom\ x\ \#_{\Gamma}$   
*<proof>*

**lemmas** *wf-b-subst-lemmas = subst-eb.simps wf-intros*

*forget-subst subst-b-b-def subst-b-v-def subst-b-ce-def fresh-e-opp-all subst-bb.simps wfV-b-fresh ms-fresh-all(6)*

**lemma** *wf-b-subst1:*

**fixes**  $\Gamma::\Gamma$  **and**  $\Gamma':\Gamma$  **and**  $v::v$  **and**  $e::e$  **and**  $c::c$  **and**  $\tau::\tau$  **and**  $ts::(string*\tau)$  *list* **and**  $\Delta::\Delta$  **and**  $b::b$   
**and**  $ftq::fun\text{-}typ\text{-}q$  **and**  $ft::fun\text{-}typ$  **and**  $s::s$  **and**  $b'::b$  **and**  $ce::ce$  **and**  $td::type\text{-}def$   
**and**  $cs::branch\text{-}s$  **and**  $css::branch\text{-}list$   
**shows**  $\Theta; B'; \Gamma \vdash_{wf} v : b' \implies \{|bv|\} = B' \implies \Theta; B \vdash_{wf} b \implies \Theta; B; \Gamma[bv::=b]_{\Gamma b} \vdash_{wf}$   
 $v[bv::=b]_{vb} : b'[bv::=b]_{bb}$  **and**  
 $\Theta; B'; \Gamma \vdash_{wf} c \implies \{|bv|\} = B' \implies \Theta; B \vdash_{wf} b \implies \Theta; B; \Gamma[bv::=b]_{\Gamma b} \vdash_{wf}$   
 $c[bv::=b]_{cb}$  **and**  
 $\Theta; B' \vdash_{wf} \Gamma \implies \{|bv|\} = B' \implies \Theta; B \vdash_{wf} b \implies \Theta; B \vdash_{wf} \Gamma[bv::=b]_{\Gamma b}$  **and**  
 $\Theta; B'; \Gamma \vdash_{wf} \tau \implies \{|bv|\} = B' \implies \Theta; B \vdash_{wf} b \implies \Theta; B; \Gamma[bv::=b]_{\Gamma b} \vdash_{wf}$

$\tau[bv::=b]_{\tau b}$  **and**  
 $\Theta; \mathcal{B}; \Gamma \vdash_{wf} ts \implies True$  **and**  
 $\vdash_{wf} \Theta \implies True$  **and**  
 $\Theta; B' \vdash_{wf} b' \implies \{|bv|\} = B' \implies \Theta; B \vdash_{wf} b \implies \Theta; B \vdash_{wf} b'[bv::=b]_{bb}$  **and**  
 $\Theta; B'; \Gamma \vdash_{wf} ce : b' \implies \{|bv|\} = B' \implies \Theta; B \vdash_{wf} b \implies \Theta; B; \Gamma[bv::=b]_{\Gamma b} \vdash_{wf}$   
 $ce[bv::=b]_{ceb} : b'[bv::=b]_{bb}$  **and**  
 $\Theta \vdash_{wf} td \implies True$   
 <proof>

**lemma** *wf-b-subst2*:

**fixes**  $\Gamma::\Gamma$  **and**  $\Gamma'::\Gamma$  **and**  $v::v$  **and**  $e::e$  **and**  $c::c$  **and**  $\tau::\tau$  **and**  $ts::(string*\tau)$  *list* **and**  $\Delta::\Delta$  **and**  $b::b$   
**and**  $ftq::fun\text{-}typ\text{-}q$  **and**  $ft::fun\text{-}typ$  **and**  $s::s$  **and**  $b'::b$  **and**  $ce::ce$  **and**  $td::type\text{-}def$   
**and**  $cs::branch\text{-}s$  **and**  $css::branch\text{-}list$   
**shows**  $\Theta; \Phi; B'; \Gamma; \Delta \vdash_{wf} e : b' \implies \{|bv|\} = B' \implies \Theta; B \vdash_{wf} b \implies \Theta; \Phi; B;$   
 $\Gamma[bv::=b]_{\Gamma b}; \Delta[bv::=b]_{\Delta b} \vdash_{wf} e[bv::=b]_{eb} : b'[bv::=b]_{bb}$  **and**  
 $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash_{wf} s : b \implies True$  **and**  
 $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta; tid; dc; t \vdash_{wf} cs : b \implies True$  **and**  
 $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta; tid; dclist \vdash_{wf} css : b \implies True$  **and**  
 $\Theta \vdash_{wf} (\Phi::\Phi) \implies True$  **and**  
 $\Theta; B'; \Gamma \vdash_{wf} \Delta \implies \{|bv|\} = B' \implies \Theta; B \vdash_{wf} b \implies \Theta; B; \Gamma[bv::=b]_{\Gamma b} \vdash_{wf} \Delta[bv::=b]_{\Delta b}$   
**and**  
 $\Theta; \Phi \vdash_{wf} ftq \implies True$  **and**  
 $\Theta; \Phi; \mathcal{B} \vdash_{wf} ft \implies True$   
 <proof>

**lemmas** *wf-b-subst = wf-b-subst1 wf-b-subst2*

**lemma** *wfT-subst-wfT*:

**fixes**  $\tau::\tau$  **and**  $b'::b$  **and**  $bv::bv$   
**assumes**  $\Theta; \{|bv|\}; (x, b, c) \#_{\Gamma} GNil \vdash_{wf} \tau$  **and**  $\Theta; B \vdash_{wf} b'$   
**shows**  $\Theta; B; (x, b[bv::=b]_{bb}, c[bv::=b]_{cb}) \#_{\Gamma} GNil \vdash_{wf} (\tau[bv::=b]_{\tau b})$   
 <proof>

**lemma** *wf-trans*:

**fixes**  $\Gamma::\Gamma$  **and**  $\Gamma'::\Gamma$  **and**  $v::v$  **and**  $e::e$  **and**  $c::c$  **and**  $\tau::\tau$  **and**  $ts::(string*\tau)$  *list* **and**  $\Delta::\Delta$  **and**  $b::b$   
**and**  $ftq::fun\text{-}typ\text{-}q$  **and**  $ft::fun\text{-}typ$  **and**  $ce::ce$  **and**  $td::type\text{-}def$  **and**  $s::s$   
**and**  $cs::branch\text{-}s$  **and**  $css::branch\text{-}list$  **and**  $\Theta::\Theta$   
**shows**  $\Theta; \mathcal{B}; \Gamma \vdash_{wf} v : b' \implies \Gamma = (x, b, c2) \#_{\Gamma} G \implies \Theta; \mathcal{B}; (x, b, c1) \#_{\Gamma} G \vdash_{wf} c2$   
 $\implies \Theta; \mathcal{B}; (x, b, c1) \#_{\Gamma} G \vdash_{wf} v : b'$  **and**  
 $\Theta; \mathcal{B}; \Gamma \vdash_{wf} c \implies \Gamma = (x, b, c2) \#_{\Gamma} G \implies \Theta; \mathcal{B}; (x, b, c1) \#_{\Gamma} G \vdash_{wf} c2 \implies$   
 $\Theta; \mathcal{B}; (x, b, c1) \#_{\Gamma} G \vdash_{wf} c$  **and**  
 $\Theta; \mathcal{B} \vdash_{wf} \Gamma \implies True$  **and**  
 $\Theta; \mathcal{B}; \Gamma \vdash_{wf} \tau \implies True$  **and**  
 $\Theta; \mathcal{B}; \Gamma \vdash_{wf} ts \implies True$  **and**  
 $\vdash_{wf} \Theta \implies True$  **and**  
 $\Theta; \mathcal{B} \vdash_{wf} b \implies True$  **and**  
 $\Theta; \mathcal{B}; \Gamma \vdash_{wf} ce : b' \implies \Gamma = (x, b, c2) \#_{\Gamma} G \implies \Theta; \mathcal{B}; (x, b, c1) \#_{\Gamma} G \vdash_{wf} c2 \implies \Theta;$   
 $\mathcal{B}; (x, b, c1) \#_{\Gamma} G \vdash_{wf} ce : b'$  **and**  
 $\Theta \vdash_{wf} td \implies True$   
 <proof>

end

# Chapter 9

## Type System

The MiniSail type system. We define subtyping judgement first and then typing judgement for the term forms

### 9.1 Subtyping

Subtyping is defined on top of refinement constraint logic (RCL). A subtyping check is converted into an RCL validity check.

**inductive** *subtype* ::  $\Theta \Rightarrow \mathcal{B} \Rightarrow \Gamma \Rightarrow \tau \Rightarrow \tau \Rightarrow \text{bool}$  ( $\langle \cdot ; \cdot ; \cdot \vdash \cdot \lesssim \cdot \rangle$  [50, 50, 50] 50) **where**

*subtype-baseI*:  $\llbracket$   
*atom*  $x \# (\Theta, \mathcal{B}, \Gamma, z, c, z', c')$  ;  
 $\Theta; \mathcal{B}; \Gamma \vdash_{wf} \{ z : b \mid c \}$  ;  
 $\Theta; \mathcal{B}; \Gamma \vdash_{wf} \{ z' : b \mid c' \}$  ;  
 $\Theta; \mathcal{B}; (x, b, c[z ::= [x]^v]_v) \#_{\Gamma} \Gamma \models c'[z' ::= [x]^v]_v$   
 $\rrbracket \Rightarrow$   
 $\Theta; \mathcal{B}; \Gamma \vdash \{ z : b \mid c \} \lesssim \{ z' : b \mid c' \}$

**equivariance** *subtype*

**nominal-inductive** *subtype*

**avoids** *subtype-baseI*:  $x$

$\langle \text{proof} \rangle$

**inductive-cases** *subtype-elim*:

$\Theta; \mathcal{B}; \Gamma \vdash \{ z : b \mid c \} \lesssim \{ z' : b \mid c' \}$

$\Theta; \mathcal{B}; \Gamma \vdash \tau_1 \lesssim \tau_2$

### 9.2 Literals

The type synthesised has the constraint that  $z$  equates to the literal

**inductive** *infer-l* ::  $l \Rightarrow \tau \Rightarrow \text{bool}$  ( $\langle \vdash \cdot \Rightarrow \cdot \rangle$  [50] 50) **where**

*infer-trueI*:  $\vdash L\text{-true} \Rightarrow \{ z : B\text{-bool} \mid [[z]^v]^{ce} == [[L\text{-true}]^v]^{ce} \}$   
*infer-falseI*:  $\vdash L\text{-false} \Rightarrow \{ z : B\text{-bool} \mid [[z]^v]^{ce} == [[L\text{-false}]^v]^{ce} \}$   
*infer-numI*:  $\vdash L\text{-num } n \Rightarrow \{ z : B\text{-int} \mid [[z]^v]^{ce} == [[L\text{-num } n]^v]^{ce} \}$   
*infer-unitI*:  $\vdash L\text{-unit} \Rightarrow \{ z : B\text{-unit} \mid [[z]^v]^{ce} == [[L\text{-unit}]^v]^{ce} \}$   
*infer-bitvecI*:  $\vdash L\text{-bitvec } bv \Rightarrow \{ z : B\text{-bitvec} \mid [[z]^v]^{ce} == [[L\text{-bitvec } bv]^v]^{ce} \}$

**nominal-inductive** *infer-l*  $\langle proof \rangle$   
**equivariance** *infer-l*

**inductive-cases** *infer-l-elim*[*elim!*]:

$\vdash L\text{-true} \Rightarrow \tau$   
 $\vdash L\text{-false} \Rightarrow \tau$   
 $\vdash L\text{-num } n \Rightarrow \tau$   
 $\vdash L\text{-unit} \Rightarrow \tau$   
 $\vdash L\text{-bitvec } x \Rightarrow \tau$   
 $\vdash l \Rightarrow \tau$

**lemma** *infer-l-form2*[*simp*]:

**shows**  $\exists z. \vdash l \Rightarrow (\{ z : \text{base-for-lit } l \mid [[z]^v]^{ce} == [[l]^v]^{ce} \})$   
 $\langle proof \rangle$

## 9.3 Values

**inductive** *infer-v* ::  $\Theta \Rightarrow \mathcal{B} \Rightarrow \Gamma \Rightarrow v \Rightarrow \tau \Rightarrow \text{bool}$  ( $\langle - ; - ; - \vdash - \Rightarrow - \rangle$  [50, 50, 50] 50) **where**

*infer-v-varI*:  $\llbracket$

$\Theta; \mathcal{B} \vdash_{wf} \Gamma ;$   
 $\text{Some } (b,c) = \text{lookup } \Gamma \ x ;$   
 $\text{atom } z \# x ; \text{atom } z \# (\Theta, \mathcal{B}, \Gamma)$

$\rrbracket \Rightarrow$

$\Theta; \mathcal{B}; \Gamma \vdash [x]^v \Rightarrow \{ z : b \mid [[z]^v]^{ce} == [[x]^v]^{ce} \}$

| *infer-v-litI*:  $\llbracket$

$\Theta; \mathcal{B} \vdash_{wf} \Gamma ;$   
 $\vdash l \Rightarrow \tau$

$\rrbracket \Rightarrow$

$\Theta; \mathcal{B}; \Gamma \vdash [l]^v \Rightarrow \tau$

| *infer-v-pairI*:  $\llbracket$

$\text{atom } z \# (v1, v2); \text{atom } z \# (\Theta, \mathcal{B}, \Gamma) ;$   
 $\Theta; \mathcal{B}; \Gamma \vdash (v1::v) \Rightarrow t1 ;$   
 $\Theta; \mathcal{B}; \Gamma \vdash (v2::v) \Rightarrow t2$

$\rrbracket \Rightarrow$

$\Theta; \mathcal{B}; \Gamma \vdash V\text{-pair } v1 \ v2 \Rightarrow (\{ z : B\text{-pair } (b\text{-of } t1) (b\text{-of } t2) \mid [[z]^v]^{ce} == [[v1, v2]^v]^{ce} \})$

| *infer-v-consI*:  $\llbracket$

$AF\text{-typedef } s \ \text{dclist} \in \text{set } \Theta ;$   
 $(dc, tc) \in \text{set } \text{dclist} ;$   
 $\Theta; \mathcal{B}; \Gamma \vdash v \Rightarrow tv ;$   
 $\Theta; \mathcal{B}; \Gamma \vdash tv \lesssim tc ;$   
 $\text{atom } z \# v ; \text{atom } z \# (\Theta, \mathcal{B}, \Gamma)$

$\rrbracket \Rightarrow$

$\Theta; \mathcal{B}; \Gamma \vdash V\text{-cons } s \ dc \ v \Rightarrow (\{ z : B\text{-id } s \mid [[z]^v]^{ce} == [V\text{-cons } s \ dc \ v]^v]^{ce} \})$

| *infer-v-conspI*:  $\llbracket$

$AF\text{-typedef-poly } s \ bv \ \text{dclist} \in \text{set } \Theta ;$   
 $(dc, tc) \in \text{set } \text{dclist} ;$

$$\begin{array}{l}
\Theta; \mathcal{B}; \Gamma \vdash v \Rightarrow tv; \\
\Theta; \mathcal{B}; \Gamma \vdash tv \lesssim tc[bv::=b]_{\tau b}; \\
atom\ z \# (\Theta, \mathcal{B}, \Gamma, v, b); \\
atom\ bv \# (\Theta, \mathcal{B}, \Gamma, v, b); \\
\Theta; \mathcal{B} \vdash_{wf} b \\
\boxed{\Longrightarrow} \\
\Theta; \mathcal{B}; \Gamma \vdash V-consp\ s\ dc\ b\ v \Rightarrow (\{ z : B-app\ s\ b \mid [[z]^v]^{ce} == (CE-val\ (V-consp\ s\ dc\ b\ v)) \})
\end{array}$$

**equivariance** *infer-v*

**nominal-inductive** *infer-v*

**avoids** *infer-v-conspI*: *bv* **and** *z* | *infer-v-varI*: *z* | *infer-v-pairI*: *z* | *infer-v-consI*: *z*  
*<proof>*

**inductive-cases** *infer-v-elim*[*elim!*]:

$$\begin{array}{l}
\Theta; \mathcal{B}; \Gamma \vdash V-var\ x \Rightarrow \tau \\
\Theta; \mathcal{B}; \Gamma \vdash V-lit\ l \Rightarrow \tau \\
\Theta; \mathcal{B}; \Gamma \vdash V-pair\ v1\ v2 \Rightarrow \tau \\
\Theta; \mathcal{B}; \Gamma \vdash V-cons\ s\ dc\ v \Rightarrow \tau \\
\Theta; \mathcal{B}; \Gamma \vdash V-pair\ v1\ v2 \Rightarrow (\{ z : b \mid c \}) \\
\Theta; \mathcal{B}; \Gamma \vdash V-pair\ v1\ v2 \Rightarrow (\{ z : [b1, b2]^b \mid [[z]^v]^{ce} == [[v1, v2]^v]^{ce} \}) \\
\Theta; \mathcal{B}; \Gamma \vdash V-consp\ s\ dc\ b\ v \Rightarrow \tau
\end{array}$$

**inductive** *check-v* ::  $\Theta \Rightarrow \mathcal{B} \Rightarrow \Gamma \Rightarrow v \Rightarrow \tau \Rightarrow bool$  ( $\langle \cdot; \cdot; \cdot \vdash \cdot \Leftarrow \cdot \rangle [50, 50, 50] 50$ ) **where**

*check-v-subtypeI*:  $\boxed{\Theta; \mathcal{B}; \Gamma \vdash \tau1 \lesssim \tau2; \Theta; \mathcal{B}; \Gamma \vdash v \Rightarrow \tau1} \Longrightarrow \Theta; \mathcal{B}; \Gamma \vdash v \Leftarrow \tau2$

**equivariance** *check-v*

**nominal-inductive** *check-v* *<proof>*

**inductive-cases** *check-v-elim*[*elim!*]:

$$\Theta; \mathcal{B}; \Gamma \vdash v \Leftarrow \tau$$

## 9.4 Expressions

Type synthesis for expressions

**inductive** *infer-e* ::  $\Theta \Rightarrow \Phi \Rightarrow \mathcal{B} \Rightarrow \Gamma \Rightarrow \Delta \Rightarrow e \Rightarrow \tau \Rightarrow bool$  ( $\langle \cdot; \cdot; \cdot; \cdot; \cdot \vdash \cdot \Rightarrow \cdot \rangle [50, 50, 50, 50] 50$ ) **where**

*infer-e-valI*:  $\boxed{\begin{array}{l} (\Theta; \mathcal{B}; \Gamma \vdash_{wf} \Delta); \\ (\Theta \vdash_{wf} (\Phi::\Phi)); \\ (\Theta; \mathcal{B}; \Gamma \vdash v \Rightarrow \tau) \end{array}} \Longrightarrow \Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash (AE-val\ v) \Rightarrow \tau$

| *infer-e-plusI*:  $\boxed{\begin{array}{l} \Theta; \mathcal{B}; \Gamma \vdash_{wf} \Delta; \\ \Theta \vdash_{wf} (\Phi::\Phi); \\ \Theta; \mathcal{B}; \Gamma \vdash v1 \Rightarrow \{ z1 : B-int \mid c1 \}; \\ \Theta; \mathcal{B}; \Gamma \vdash v2 \Rightarrow \{ z2 : B-int \mid c2 \}; \\ atom\ z3 \# (AE-op\ Plus\ v1\ v2); atom\ z3 \# \Gamma \end{array}} \Longrightarrow \Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash AE-op\ Plus\ v1\ v2 \Rightarrow \{ z3 : B-int \mid [[z3]^v]^{ce} == (CE-op\ Plus\ [v1]^{ce}\ [v2]^{ce}) \}$

| *infer-e-leqI*:  $\boxed{\phantom{\Theta; \mathcal{B}; \Gamma \vdash_{wf} \Delta; \Theta \vdash_{wf} (\Phi::\Phi); \Theta; \mathcal{B}; \Gamma \vdash v1 \Rightarrow \{ z1 : B-int \mid c1 \}; \Theta; \mathcal{B}; \Gamma \vdash v2 \Rightarrow \{ z2 : B-int \mid c2 \}; atom\ z3 \# (AE-op\ Plus\ v1\ v2); atom\ z3 \# \Gamma}}$

$\Theta; \mathcal{B}; \Gamma \vdash_{wf} \Delta;$   
 $\Theta \vdash_{wf} (\Phi::\Phi);$   
 $\Theta; \mathcal{B}; \Gamma \vdash v1 \Rightarrow \{ \{ z1 : B-int \mid c1 \} \};$   
 $\Theta; \mathcal{B}; \Gamma \vdash v2 \Rightarrow \{ \{ z2 : B-int \mid c2 \} \};$   
 $atom\ z3 \# (AE-op\ LEq\ v1\ v2); atom\ z3 \# \Gamma$   
 $\] \Longrightarrow$   
 $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash AE-op\ LEq\ v1\ v2 \Rightarrow \{ \{ z3 : B-bool \mid [[z3]^v]^{ce} == (CE-op\ LEq\ [v1]^{ce}\ [v2]^{ce}) \} \}$

$| infer-e-eqI: \[$   
 $\Theta; \mathcal{B}; \Gamma \vdash_{wf} \Delta;$   
 $\Theta \vdash_{wf} (\Phi::\Phi);$   
 $\Theta; \mathcal{B}; \Gamma \vdash v1 \Rightarrow \{ \{ z1 : b \mid c1 \} \};$   
 $\Theta; \mathcal{B}; \Gamma \vdash v2 \Rightarrow \{ \{ z2 : b \mid c2 \} \};$   
 $atom\ z3 \# (AE-op\ Eq\ v1\ v2); atom\ z3 \# \Gamma;$   
 $b \in \{ B-bool, B-int, B-unit \}$   
 $\] \Longrightarrow$   
 $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash AE-op\ Eq\ v1\ v2 \Rightarrow \{ \{ z3 : B-bool \mid [[z3]^v]^{ce} == (CE-op\ Eq\ [v1]^{ce}\ [v2]^{ce}) \} \}$

$| infer-e-appI: \[$   
 $\Theta; \mathcal{B}; \Gamma \vdash_{wf} \Delta;$   
 $\Theta \vdash_{wf} (\Phi::\Phi);$   
 $Some\ (AF-fundef\ f\ (AF-fun-typ-none\ (AF-fun-typ\ x\ b\ c\ \tau'\ s')) = lookup-fun\ \Phi\ f);$   
 $\Theta; \mathcal{B}; \Gamma \vdash v \Leftarrow \{ \{ x : b \mid c \} \};$   
 $atom\ x \# (\Theta, \Phi, \mathcal{B}, \Gamma, \Delta, v, \tau);$   
 $\tau'[x::=v]_v = \tau$   
 $\] \Longrightarrow$   
 $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash AE-app\ f\ v \Rightarrow \tau$

$| infer-e-appPI: \[$   
 $\Theta; \mathcal{B}; \Gamma \vdash_{wf} \Delta;$   
 $\Theta \vdash_{wf} (\Phi::\Phi);$   
 $\Theta; \mathcal{B} \vdash_{wf} b';$   
 $Some\ (AF-fundef\ f\ (AF-fun-typ-some\ bv\ (AF-fun-typ\ x\ b\ c\ \tau'\ s')) = lookup-fun\ \Phi\ f);$   
 $\Theta; \mathcal{B}; \Gamma \vdash v \Leftarrow \{ \{ x : b[bv::=b']_b \mid c[bv::=b']_b \} \}; atom\ x \# \Gamma;$   
 $(\tau'[bv::=b']_b[x::=v]_v) = \tau;$   
 $atom\ bv \# (\Theta, \Phi, \mathcal{B}, \Gamma, \Delta, b', v, \tau)$   
 $\] \Longrightarrow$   
 $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash AE-appP\ f\ b'\ v \Rightarrow \tau$

$| infer-e-fstI: \[$   
 $\Theta; \mathcal{B}; \Gamma \vdash_{wf} \Delta;$   
 $\Theta \vdash_{wf} (\Phi::\Phi);$   
 $\Theta; \mathcal{B}; \Gamma \vdash v \Rightarrow \{ \{ z' : [b1, b2]^b \mid c \} \};$   
 $atom\ z \# AE-fst\ v; atom\ z \# \Gamma \] \Longrightarrow$   
 $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash AE-fst\ v \Rightarrow \{ \{ z : b1 \mid [[z]^v]^{ce} == ((CE-fst\ [v]^{ce})) \} \}$

$| infer-e-sndI: \[$   
 $\Theta; \mathcal{B}; \Gamma \vdash_{wf} \Delta;$   
 $\Theta \vdash_{wf} (\Phi::\Phi);$   
 $\Theta; \mathcal{B}; \Gamma \vdash v \Rightarrow \{ \{ z' : [b1, b2]^b \mid c \} \};$   
 $atom\ z \# AE-snd\ v; atom\ z \# \Gamma \] \Longrightarrow$   
 $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash AE-snd\ v \Rightarrow \{ \{ z : b2 \mid [[z]^v]^{ce} == ((CE-snd\ [v]^{ce})) \} \}$

| *infer-e-lenI*:  $\llbracket$   
 $\Theta; \mathcal{B}; \Gamma \vdash_{wf} \Delta$  ;  
 $\Theta \vdash_{wf} (\Phi::\Phi)$  ;  
 $\Theta; \mathcal{B}; \Gamma \vdash v \Rightarrow \{ \{ z' : B\text{-bitvec} \mid c \} \}$  ;  
 $atom\ z \# AE\text{-len}\ v ; atom\ z \# \Gamma \rrbracket \Longrightarrow$   
 $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash AE\text{-len}\ v \Rightarrow \{ \{ z : B\text{-int} \mid [[z]^v]^{ce} == ((CE\text{-len}\ [v]^{ce})) \} \}$

| *infer-e-mvarI*:  $\llbracket$   
 $\Theta; \mathcal{B} \vdash_{wf} \Gamma$  ;  
 $\Theta \vdash_{wf} (\Phi::\Phi)$  ;  
 $\Theta; \mathcal{B}; \Gamma \vdash_{wf} \Delta$  ;  
 $(u, \tau) \in setD\ \Delta \rrbracket \Longrightarrow$   
 $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash AE\text{-mvar}\ u \Rightarrow \tau$

| *infer-e-concatI*:  $\llbracket$   
 $\Theta; \mathcal{B}; \Gamma \vdash_{wf} \Delta$  ;  
 $\Theta \vdash_{wf} (\Phi::\Phi)$  ;  
 $\Theta; \mathcal{B}; \Gamma \vdash v1 \Rightarrow \{ \{ z1 : B\text{-bitvec} \mid c1 \} \}$  ;  
 $\Theta; \mathcal{B}; \Gamma \vdash v2 \Rightarrow \{ \{ z2 : B\text{-bitvec} \mid c2 \} \}$  ;  
 $atom\ z3 \# (AE\text{-concat}\ v1\ v2); atom\ z3 \# \Gamma \rrbracket \Longrightarrow$   
 $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash AE\text{-concat}\ v1\ v2 \Rightarrow \{ \{ z3 : B\text{-bitvec} \mid [[z3]^v]^{ce} == (CE\text{-concat}\ [v1]^{ce}\ [v2]^{ce}) \} \}$

| *infer-e-splitI*:  $\llbracket$   
 $\Theta; \mathcal{B}; \Gamma \vdash_{wf} \Delta$  ;  
 $\Theta \vdash_{wf} (\Phi::\Phi)$  ;  
 $\Theta; \mathcal{B}; \Gamma \vdash v1 \Rightarrow \{ \{ z1 : B\text{-bitvec} \mid c1 \} \}$  ;  
 $\Theta; \mathcal{B}; \Gamma \vdash v2 \Leftarrow \{ \{ z2 : B\text{-int} \mid (CE\text{-op}\ LEq\ (CE\text{-val}\ (V\text{-lit}\ (L\text{-num}\ 0)))\ (CE\text{-val}\ (V\text{-var}\ z2)))$   
 $== (CE\text{-val}\ (V\text{-lit}\ L\text{-true}))\ AND$   
 $(CE\text{-op}\ LEq\ (CE\text{-val}\ (V\text{-var}\ z2))\ (CE\text{-len}\ (CE\text{-val}\ (v1)))) == (CE\text{-val}$   
 $(V\text{-lit}\ L\text{-true})) \}$  ;  
 $atom\ z1 \# (AE\text{-split}\ v1\ v2); atom\ z1 \# \Gamma$  ;  
 $atom\ z2 \# (AE\text{-split}\ v1\ v2); atom\ z2 \# \Gamma$  ;  
 $atom\ z3 \# (AE\text{-split}\ v1\ v2); atom\ z3 \# \Gamma$   
 $\rrbracket \Longrightarrow$   
 $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash (AE\text{-split}\ v1\ v2) \Rightarrow \{ \{ z3 : B\text{-pair}\ B\text{-bitvec}\ B\text{-bitvec} \mid$   
 $((CE\text{-val}\ v1) == (CE\text{-concat}\ (CE\text{-fst}\ (CE\text{-val}\ (V\text{-var}\ z3)))\ (CE\text{-snd}\ (CE\text{-val}\ (V\text{-var}$   
 $z3))))))$   
 $AND\ (((CE\text{-len}\ (CE\text{-fst}\ (CE\text{-val}\ (V\text{-var}\ z3)))) == (CE\text{-val}\ (v2))) \}$

**equivariance** *infer-e*

**nominal-inductive** *infer-e*

**avoids** *infer-e-appI*:  $x \mid infer\text{-e}\text{-appPI}$ :  $bv \mid infer\text{-e}\text{-splitI}$ :  $z3$  and  $z1$  and  $z2$   
 $\langle proof \rangle$

**inductive-cases** *infer-e-elim*[*elim*!]:

$\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash (AE\text{-op}\ Plus\ v1\ v2) \Rightarrow \{ \{ z3 : B\text{-int} \mid [[z3]^v]^{ce} == (CE\text{-op}\ Plus\ [v1]^{ce}\ [v2]^{ce}) \} \}$   
 $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash (AE\text{-op}\ LEq\ v1\ v2) \Rightarrow \{ \{ z3 : B\text{-bool} \mid [[z3]^v]^{ce} == (CE\text{-op}\ LEq\ [v1]^{ce}\ [v2]^{ce}) \} \}$   
 $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash (AE\text{-op}\ Plus\ v1\ v2) \Rightarrow \{ \{ z3 : B\text{-int} \mid c \} \}$   
 $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash (AE\text{-op}\ Plus\ v1\ v2) \Rightarrow \{ \{ z3 : b \mid c \} \}$   
 $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash (AE\text{-op}\ LEq\ v1\ v2) \Rightarrow \{ \{ z3 : b \mid c \} \}$   
 $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash (AE\text{-app}\ f\ v) \Rightarrow \tau$

$\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash (AE\text{-val } v) \Rightarrow \tau$   
 $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash (AE\text{-fst } v) \Rightarrow \tau$   
 $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash (AE\text{-snd } v) \Rightarrow \tau$   
 $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash (AE\text{-mvar } u) \Rightarrow \tau$   
 $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash (AE\text{-op Plus } v1 \ v2) \Rightarrow \tau$   
 $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash (AE\text{-op LEq } v1 \ v2) \Rightarrow \tau$   
 $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash (AE\text{-op LEq } v1 \ v2) \Rightarrow \{ \! \{ z\mathcal{?} : B\text{-bool} \mid c \} \! \}$   
 $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash (AE\text{-app } f \ v) \Rightarrow \tau[x::=v]_{\tau v}$   
 $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash (AE\text{-op opp } v1 \ v2) \Rightarrow \tau$   
 $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash (AE\text{-len } v) \Rightarrow \tau$   
 $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash (AE\text{-len } v) \Rightarrow \{ \! \{ z : B\text{-int} \mid [[z]^v]^{ce} == ((CE\text{-len } [v]^{ce})) \} \! \}$   
 $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash AE\text{-concat } v1 \ v2 \Rightarrow \tau$   
 $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash AE\text{-concat } v1 \ v2 \Rightarrow \{ \! \{ z : b \mid c \} \! \}$   
 $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash AE\text{-concat } v1 \ v2 \Rightarrow \{ \! \{ z : B\text{-bitvec} \mid [[z]^v]^{ce} == (CE\text{-concat } [v1]^{ce} \ [v2]^{ce}) \} \! \}$   
 $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash (AE\text{-appP } f \ b \ v) \Rightarrow \tau$   
 $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash AE\text{-split } v1 \ v2 \Rightarrow \tau$   
 $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash (AE\text{-op Eq } v1 \ v2) \Rightarrow \{ \! \{ z\mathcal{?} : b \mid c \} \! \}$   
 $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash (AE\text{-op Eq } v1 \ v2) \Rightarrow \{ \! \{ z\mathcal{?} : B\text{-bool} \mid c \} \! \}$   
 $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash (AE\text{-op Eq } v1 \ v2) \Rightarrow \tau$   
**nominal-termination** (*eqvt*)  $\langle proof \rangle$

## 9.5 Statements

**inductive** *check-s* ::  $\Theta \Rightarrow \Phi \Rightarrow \mathcal{B} \Rightarrow \Gamma \Rightarrow \Delta \Rightarrow s \Rightarrow \tau \Rightarrow \text{bool} \ (\langle - ; - ; - ; - ; - \vdash - \Leftarrow \rightarrow [50, 50, 50, 50, 50] \ 50)$  **and**

*check-branch-s* ::  $\Theta \Rightarrow \Phi \Rightarrow \mathcal{B} \Rightarrow \Gamma \Rightarrow \Delta \Rightarrow \text{tyid} \Rightarrow \text{string} \Rightarrow \tau \Rightarrow v \Rightarrow \text{branch-s} \Rightarrow \tau \Rightarrow \text{bool} \ (\langle - ; - ; - ; - ; - ; - ; - ; - \vdash - \Leftarrow \rightarrow [50, 50, 50, 50, 50] \ 50)$  **and**

*check-branch-list* ::  $\Theta \Rightarrow \Phi \Rightarrow \mathcal{B} \Rightarrow \Gamma \Rightarrow \Delta \Rightarrow \text{tyid} \Rightarrow (\text{string} * \tau) \text{ list} \Rightarrow v \Rightarrow \text{branch-list} \Rightarrow \tau \Rightarrow \text{bool} \ (\langle - ; - ; - ; - ; - ; - ; - ; - \vdash - \Leftarrow \rightarrow [50, 50, 50, 50, 50] \ 50)$  **where**

*check-valI*:  $\llbracket$   
 $\Theta; \mathcal{B}; \Gamma \vdash_{wf} \Delta$ ;  
 $\Theta \vdash_{wf} \Phi$ ;  
 $\Theta; \mathcal{B}; \Gamma \vdash v \Rightarrow \tau'$ ;  
 $\Theta; \mathcal{B}; \Gamma \vdash \tau' \lesssim \tau \rrbracket \implies$   
 $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash (AS\text{-val } v) \Leftarrow \tau$

| *check-letI*:  $\llbracket$   
 $\text{atom } x \# (\Theta, \Phi, \mathcal{B}, \Gamma, \Delta, e, \tau)$ ;  
 $\text{atom } z \# (x, \Theta, \Phi, \mathcal{B}, \Gamma, \Delta, e, \tau, s)$ ;  
 $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash e \Rightarrow \{ \! \{ z : b \mid c \} \! \}$ ;  
 $\Theta; \Phi; \mathcal{B}; ((x, b, c[z::=V\text{-var } x]_v) \#_{\Gamma} \Gamma); \Delta \vdash s \Leftarrow \tau$   
 $\rrbracket \implies$   
 $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash (AS\text{-let } x \ e \ s) \Leftarrow \tau$

| *check-assertI*:  $\llbracket$   
 $\text{atom } x \# (\Theta, \Phi, \mathcal{B}, \Gamma, \Delta, c, \tau, s)$ ;  
 $\Theta; \Phi; \mathcal{B}; ((x, B\text{-bool}, c) \#_{\Gamma} \Gamma); \Delta \vdash s \Leftarrow \tau$ ;  
 $\Theta; \mathcal{B}; \Gamma \models c$ ;  
 $\Theta; \mathcal{B}; \Gamma \vdash_{wf} \Delta$   
 $\rrbracket \implies$   
 $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash (AS\text{-assert } c \ s) \Leftarrow \tau$

$| \text{check-branch-s-branchI}: \llbracket$   
 $\Theta; \mathcal{B}; \Gamma \vdash_{wf} \Delta;$   
 $\vdash_{wf} \Theta;$   
 $\Theta; \mathcal{B}; \Gamma \vdash_{wf} \tau;$   
 $\Theta; \{\|\}; GNil \vdash_{wf} \text{const};$   
 $\text{atom } x \# (\Theta, \Phi, \mathcal{B}, \Gamma, \Delta, \text{tid}, \text{cons}, \text{const}, v, \tau);$   
 $\Theta; \Phi; \mathcal{B}; ((x, b\text{-of } \text{const}, ([v]^{ce} == [V\text{-cons } \text{tid } \text{cons } [x]^v]^{ce}) \text{ AND } (c\text{-of } \text{const } x)) \#_{\Gamma} \Gamma); \Delta \vdash s \Leftarrow$   
 $\tau$   
 $\rrbracket \Rightarrow$   
 $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta; \text{tid}; \text{cons}; \text{const}; v \vdash (AS\text{-branch } \text{cons } x \ s) \Leftarrow \tau$

$| \text{check-branch-list-consI}: \llbracket$   
 $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta; \text{tid}; \text{cons}; \text{const}; v \vdash \text{cs} \Leftarrow \tau;$   
 $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta; \text{tid}; \text{dclist}; v \vdash \text{css} \Leftarrow \tau$   
 $\rrbracket \Rightarrow$   
 $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta; \text{tid}; (\text{cons}, \text{const}) \# \text{dclist}; v \vdash AS\text{-cons } \text{cs } \text{css} \Leftarrow \tau$

$| \text{check-branch-list-finalI}: \llbracket$   
 $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta; \text{tid}; \text{cons}; \text{const}; v \vdash \text{cs} \Leftarrow \tau$   
 $\rrbracket \Rightarrow$   
 $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta; \text{tid}; [(\text{cons}, \text{const})]; v \vdash AS\text{-final } \text{cs} \Leftarrow \tau$

$| \text{check-ifI}: \llbracket$   
 $\text{atom } z \# (\Theta, \Phi, \mathcal{B}, \Gamma, \Delta, v, s1, s2, \tau);$   
 $(\Theta; \mathcal{B}; \Gamma \vdash v \Leftarrow (\{z : B\text{-bool} \mid TRUE \})) ;$   
 $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash s1 \Leftarrow (\{z : b\text{-of } \tau \mid ([v]^{ce} == [[L\text{-true}]^v]^{ce}) \text{ IMP } (c\text{-of } \tau \ z) \});$   
 $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash s2 \Leftarrow (\{z : b\text{-of } \tau \mid ([v]^{ce} == [[L\text{-false}]^v]^{ce}) \text{ IMP } (c\text{-of } \tau \ z) \});$   
 $\rrbracket \Rightarrow$   
 $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash IF \ v \ THEN \ s1 \ ELSE \ s2 \Leftarrow \tau$

$| \text{check-let2I}: \llbracket$   
 $\text{atom } x \# (\Theta, \Phi, \mathcal{B}, G, \Delta, t, s1, \tau);$   
 $\Theta; \Phi; \mathcal{B}; G; \Delta \vdash s1 \Leftarrow t;$   
 $\Theta; \Phi; \mathcal{B}; ((x, b\text{-of } t, c\text{-of } t \ x) \#_{\Gamma} G); \Delta \vdash s2 \Leftarrow \tau$   
 $\rrbracket \Rightarrow$   
 $\Theta; \Phi; \mathcal{B}; G; \Delta \vdash (LET \ x : t = s1 \ IN \ s2) \Leftarrow \tau$

$| \text{check-varI}: \llbracket$   
 $\text{atom } u \# (\Theta, \Phi, \mathcal{B}, \Gamma, \Delta, \tau', v, \tau);$   
 $\Theta; \mathcal{B}; \Gamma \vdash v \Leftarrow \tau';$   
 $\Theta; \Phi; \mathcal{B}; \Gamma; ((u, \tau') \#_{\Delta} \Delta) \vdash s \Leftarrow \tau$   
 $\rrbracket \Rightarrow$   
 $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash (VAR \ u : \tau' = v \ IN \ s) \Leftarrow \tau$

$| \text{check-assignI}: \llbracket$   
 $\Theta \vdash_{wf} \Phi;$   
 $\Theta; \mathcal{B}; \Gamma \vdash_{wf} \Delta;$   
 $(u, \tau) \in \text{setD } \Delta;$   
 $\Theta; \mathcal{B}; \Gamma \vdash v \Leftarrow \tau;$   
 $\Theta; \mathcal{B}; \Gamma \vdash (\{z : B\text{-unit} \mid TRUE \}) \lesssim \tau'$   
 $\rrbracket \Rightarrow$   
 $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash (u ::= v) \Leftarrow \tau'$

| *check-whileI*:  $\llbracket$   
 $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash s1 \Leftarrow \{ z : B\text{-bool} \mid TRUE \};$   
 $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash s2 \Leftarrow \{ z : B\text{-unit} \mid TRUE \};$   
 $\Theta; \mathcal{B}; \Gamma \vdash (\{ z : B\text{-unit} \mid TRUE \}) \lesssim \tau'$   
 $\rrbracket \Rightarrow$   
 $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash WHILE\ s1\ DO\ \{ s2 \} \Leftarrow \tau'$

| *check-seqI*:  $\llbracket$   
 $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash s1 \Leftarrow \{ z : B\text{-unit} \mid TRUE \};$   
 $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash s2 \Leftarrow \tau$   
 $\rrbracket \Rightarrow$   
 $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash s1 ;; s2 \Leftarrow \tau$

| *check-caseI*:  $\llbracket$   
 $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta; tid ; dclist ; v \vdash cs \Leftarrow \tau ;$   
 $(AF\text{-typedef}\ tid\ dclist) \in set\ \Theta ;$   
 $\Theta; \mathcal{B}; \Gamma \vdash v \Leftarrow \{ z : B\text{-id}\ tid \mid TRUE \};$   
 $\vdash_{wf}\ \Theta$   
 $\rrbracket \Rightarrow$   
 $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash AS\text{-match}\ v\ cs \Leftarrow \tau$

**equivariance** *check-s*

We only need avoidance for cases where a variable is added to a context

**nominal-inductive** *check-s*

**avoids** *check-letI*:  $x$  **and**  $z$  | *check-branch-s-branchI*:  $x$  | *check-let2I*:  $x$  | *check-varI*:  $u$  | *check-iffI*:  $z$   
*check-assertI*:  $x$   
 $\langle proof \rangle$

**inductive-cases** *check-s-elim*[*elim!*]:

$\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash AS\text{-val}\ v \Leftarrow \tau$   
 $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash AS\text{-let}\ x\ e\ s \Leftarrow \tau$   
 $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash AS\text{-if}\ v\ s1\ s2 \Leftarrow \tau$   
 $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash AS\text{-let2}\ x\ t\ s1\ s2 \Leftarrow \tau$   
 $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash AS\text{-while}\ s1\ s2 \Leftarrow \tau$   
 $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash AS\text{-var}\ u\ t\ v\ s \Leftarrow \tau$   
 $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash AS\text{-seq}\ s1\ s2 \Leftarrow \tau$   
 $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash AS\text{-assign}\ u\ v \Leftarrow \tau$   
 $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash AS\text{-match}\ v\ cs \Leftarrow \tau$   
 $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash AS\text{-assert}\ c\ s \Leftarrow \tau$

**inductive-cases** *check-branch-s-elim*[*elim!*]:

$\Theta; \Phi; \mathcal{B}; \Gamma; \Delta; tid ; dclist ; v \vdash (AS\text{-final}\ cs) \Leftarrow \tau$   
 $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta; tid ; dclist ; v \vdash (AS\text{-cons}\ cs\ css) \Leftarrow \tau$   
 $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta; tid ; cons ; const ; v \vdash (AS\text{-branch}\ dc\ x\ s) \Leftarrow \tau$

## 9.6 Programs

Type check function bodies

**inductive** *check-funtyp* ::  $\Theta \Rightarrow \Phi \Rightarrow \mathcal{B} \Rightarrow fun\text{-typ} \Rightarrow bool\ (\langle - ; - ; - \vdash - \rangle)$  **where**

*check-funtypI*:  $\llbracket$   
*atom*  $x \# (\Theta, \Phi, B, b)$ ;  
 $\Theta; \Phi; B; ((x, b, c) \#_{\Gamma} GNil); \llbracket_{\Delta} \vdash s \Leftarrow \tau$   
 $\rrbracket \Rightarrow$   
 $\Theta; \Phi; B \vdash (AF\text{-fun-ty}p\ x\ b\ c\ \tau\ s)$

**equivariance** *check-funtyp*

**nominal-inductive** *check-funtyp*

**avoids** *check-funtypI*:  $x$

$\langle proof \rangle$

**inductive** *check-funtypq* ::  $\Theta \Rightarrow \Phi \Rightarrow \text{fun-ty}p\text{-}q \Rightarrow \text{bool} (\langle - ; - \vdash - \rangle)$  **where**

*check-fundefq-simpleI*:  $\llbracket$   
 $\Theta; \Phi; \{\llbracket\}\} \vdash (AF\text{-fun-ty}p\ x\ b\ c\ t\ s)$

$\rrbracket \Rightarrow$   
 $\Theta; \Phi \vdash ((AF\text{-fun-ty}p\text{-}none\ (AF\text{-fun-ty}p\ x\ b\ c\ t\ s))$

*check-funtypq-polyI*:  $\llbracket$   
*atom*  $bv \# (\Theta, \Phi, (AF\text{-fun-ty}p\ x\ b\ c\ t\ s))$ ;  
 $\Theta; \Phi; \{|bv|\} \vdash (AF\text{-fun-ty}p\ x\ b\ c\ t\ s)$

$\rrbracket \Rightarrow$   
 $\Theta; \Phi \vdash (AF\text{-fun-ty}p\text{-}some\ bv\ (AF\text{-fun-ty}p\ x\ b\ c\ t\ s))$

**equivariance** *check-funtypq*

**nominal-inductive** *check-funtypq*

**avoids** *check-funtypq-polyI*:  $bv$

$\langle proof \rangle$

**inductive** *check-fundef* ::  $\Theta \Rightarrow \Phi \Rightarrow \text{fun-def} \Rightarrow \text{bool} (\langle - ; - \vdash - \rangle)$  **where**

*check-fundefI*:  $\llbracket$   
 $\Theta; \Phi \vdash ft$

$\rrbracket \Rightarrow$   
 $\Theta; \Phi \vdash (AF\text{-fundef}\ f\ ft)$

**equivariance** *check-fundef*

**nominal-inductive** *check-fundef*  $\langle proof \rangle$

Temporarily remove this simproc as it produces untidy eliminations

**declare**[[ *simproc del*: *alpha-lst*]]

**inductive-cases** *check-funtyp-elim*[[*elim!*]]:

*check-funtyp*  $\Theta\ \Phi\ B\ ft$

**inductive-cases** *check-funtypq-elim*[[*elim!*]]:

*check-funtypq*  $\Theta\ \Phi\ (AF\text{-fun-ty}p\text{-}none\ (AF\text{-fun-ty}p\ x\ b\ c\ \tau\ s))$   
*check-funtypq*  $\Theta\ \Phi\ (AF\text{-fun-ty}p\text{-}some\ bv\ (AF\text{-fun-ty}p\ x\ b\ c\ \tau\ s))$

**inductive-cases** *check-fundef-elim*[[*elim!*]]:

*check-fundef*  $\Theta\ \Phi\ (AF\text{-fundef}\ f\ ftq)$

**declare**[[ *simproc add*: *alpha-lst*]]

**nominal-function**  $\Delta\text{-of} :: \text{var-def list} \Rightarrow \Delta$  **where**

$\Delta\text{-of} [] = \text{DNil}$

|  $\Delta\text{-of} ((\text{AV-def } u \ t \ v)\#vs) = (u,t) \#_{\Delta} (\Delta\text{-of } vs)$

$\langle \text{proof} \rangle$

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

**inductive** *check-prog* ::  $p \Rightarrow \tau \Rightarrow \text{bool} (\ \vdash \ - \Leftarrow \ - \ )$  **where**

$\llbracket$   
   $\Theta; \Phi; \{\|\}; \text{GNil}; \Delta\text{-of } \mathcal{G} \vdash s \Leftarrow \tau$   
 $\rrbracket \Rightarrow \vdash (\text{AP-prog } \Theta \ \Phi \ \mathcal{G} \ s) \Leftarrow \tau$

**inductive-cases** *check-prog-elim*[*elim!*]:

$\vdash (\text{AP-prog } \Theta \ \Phi \ \mathcal{G} \ s) \Leftarrow \tau$

**end**

# Chapter 10

## Operational Semantics

Here we define the operational semantics in terms of a small-step reduction relation.

### 10.1 Reduction Rules

The store for mutable variables

**type-synonym**  $\delta = (u*v) \text{ list}$

**nominal-function**  $\text{update-d} :: \delta \Rightarrow u \Rightarrow v \Rightarrow \delta$  **where**

$\text{update-d } [] \text{ - -} = []$   
|  $\text{update-d } ((u',v')\#\delta) \ u \ v = (\text{if } u = u' \text{ then } ((u,v)\#\delta) \ \text{else } ((u',v')\# (\text{update-d } \delta \ u \ v)))$   
 $\langle \text{proof} \rangle$

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

Relates constructor to the branch in the case and binding variable and statement

**inductive**  $\text{find-branch} :: \text{dc} \Rightarrow \text{branch-list} \Rightarrow \text{branch-s} \Rightarrow \text{bool}$  **where**

$\text{find-branch-finalI: } \text{dc}' = \text{dc} \implies \text{find-branch } \text{dc}' \ (\text{AS-final } (\text{AS-branch } \text{dc} \ x \ s)) \ (\text{AS-branch } \text{dc} \ x \ s)$   
|  $\text{find-branch-branch-eqI: } \text{dc}' = \text{dc} \implies \text{find-branch } \text{dc}' \ (\text{AS-cons } (\text{AS-branch } \text{dc} \ x \ s) \ \text{css}) \ (\text{AS-branch } \text{dc} \ x \ s)$   
|  $\text{find-branch-branch-neqI: } \llbracket \text{dc} \neq \text{dc}'; \text{find-branch } \text{dc}' \ \text{css} \ \text{cs} \rrbracket \implies \text{find-branch } \text{dc}' \ (\text{AS-cons } (\text{AS-branch } \text{dc} \ x \ s) \ \text{css}) \ \text{cs}$

**equivariance**  $\text{find-branch}$

**nominal-inductive**  $\text{find-branch} \langle \text{proof} \rangle$

**inductive-cases**  $\text{find-branch-elim}[elim!]:$

$\text{find-branch } \text{dc} \ (\text{AS-final } \text{cs}') \ \text{cs}$   
 $\text{find-branch } \text{dc} \ (\text{AS-cons } \text{cs}' \ \text{css}) \ \text{cs}$

**nominal-function**  $\text{lookup-branch} :: \text{dc} \Rightarrow \text{branch-list} \Rightarrow \text{branch-s} \ \text{option}$  **where**

$\text{lookup-branch } \text{dc} \ (\text{AS-final } (\text{AS-branch } \text{dc}' \ x \ s)) = (\text{if } \text{dc} = \text{dc}' \ \text{then } (\text{Some } (\text{AS-branch } \text{dc}' \ x \ s)) \ \text{else } \text{None})$   
|  $\text{lookup-branch } \text{dc} \ (\text{AS-cons } (\text{AS-branch } \text{dc}' \ x \ s) \ \text{css}) = (\text{if } \text{dc} = \text{dc}' \ \text{then } (\text{Some } (\text{AS-branch } \text{dc}' \ x \ s)) \ \text{else } \text{lookup-branch } \text{dc} \ \text{css})$   
 $\langle \text{proof} \rangle$

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

## Reduction rules

**inductive** *reduce-stmt* ::  $\Phi \Rightarrow \delta \Rightarrow s \Rightarrow \delta \Rightarrow s \Rightarrow \text{bool}$  ( $\langle \cdot \vdash \langle \cdot, \cdot \rangle \longrightarrow \langle \cdot, \cdot \rangle$ ) [50, 50, 50] 50  
**where**

- reduce-if-trueI*:  $\Phi \vdash \langle \delta, AS\text{-if } [L\text{-true}]^v s1\ s2 \rangle \longrightarrow \langle \delta, s1 \rangle$
- | *reduce-if-falseI*:  $\Phi \vdash \langle \delta, AS\text{-if } [L\text{-false}]^v s1\ s2 \rangle \longrightarrow \langle \delta, s2 \rangle$
- | *reduce-let-valI*:  $\Phi \vdash \langle \delta, AS\text{-let } x\ (AE\text{-val } v)\ s \rangle \longrightarrow \langle \delta, s[x::=v]_{sv} \rangle$
- | *reduce-let-plusI*:  $\Phi \vdash \langle \delta, AS\text{-let } x\ (AE\text{-op } Plus\ ((V\text{-lit } (L\text{-num } n1)))\ ((V\text{-lit } (L\text{-num } n2))))\ s \rangle \longrightarrow$   
 $\langle \delta, AS\text{-let } x\ (AE\text{-val } (V\text{-lit } (L\text{-num } ((n1)+(n2))))\ s \rangle$
- | *reduce-let-leqI*:  $b = (\text{if } (n1 \leq n2)\ \text{then } L\text{-true}\ \text{else } L\text{-false}) \Longrightarrow$   
 $\Phi \vdash \langle \delta, AS\text{-let } x\ ((AE\text{-op } LEq\ (V\text{-lit } (L\text{-num } n1))\ (V\text{-lit } (L\text{-num } n2))))\ s \rangle \longrightarrow$   
 $\langle \delta, AS\text{-let } x\ (AE\text{-val } (V\text{-lit } b))\ s \rangle$
- | *reduce-let-eqI*:  $b = (\text{if } (n1 = n2)\ \text{then } L\text{-true}\ \text{else } L\text{-false}) \Longrightarrow$   
 $\Phi \vdash \langle \delta, AS\text{-let } x\ ((AE\text{-op } Eq\ (V\text{-lit } n1)\ (V\text{-lit } n2)))\ s \rangle \longrightarrow$   
 $\langle \delta, AS\text{-let } x\ (AE\text{-val } (V\text{-lit } b))\ s \rangle$
- | *reduce-let-appI*:  $\text{Some } (AF\text{-fundef } f\ (AF\text{-fun-typ-none } (AF\text{-fun-typ } z\ b\ c\ \tau\ s'))) = \text{lookup-fun } \Phi\ f \Longrightarrow$   
 $\Phi \vdash \langle \delta, AS\text{-let } x\ ((AE\text{-app } f\ v))\ s \rangle \longrightarrow \langle \delta, AS\text{-let2 } x\ \tau[z::=v]_{\tau v}\ s'[z::=v]_{sv}\ s \rangle$
- | *reduce-let-appPI*:  $\text{Some } (AF\text{-fundef } f\ (AF\text{-fun-typ-some } bv\ (AF\text{-fun-typ } z\ b\ c\ \tau\ s'))) = \text{lookup-fun } \Phi\ f \Longrightarrow$   
 $\Phi \vdash \langle \delta, AS\text{-let } x\ ((AE\text{-appP } f\ b'\ v))\ s \rangle \longrightarrow \langle \delta, AS\text{-let2 } x\ \tau[bv::=b']_{\tau b}[z::=v]_{\tau v}\ s'[bv::=b']_{sb}[z::=v]_{sv}\ s \rangle$
- | *reduce-let-fstI*:  $\Phi \vdash \langle \delta, AS\text{-let } x\ (AE\text{-fst } (V\text{-pair } v1\ v2))\ s \rangle \longrightarrow \langle \delta, AS\text{-let } x\ (AE\text{-val } v1)\ s \rangle$
- | *reduce-let-sndI*:  $\Phi \vdash \langle \delta, AS\text{-let } x\ (AE\text{-snd } (V\text{-pair } v1\ v2))\ s \rangle \longrightarrow \langle \delta, AS\text{-let } x\ (AE\text{-val } v2)\ s \rangle$
- | *reduce-let-concatI*:  $\Phi \vdash \langle \delta, AS\text{-let } x\ (AE\text{-concat } (V\text{-lit } (L\text{-bitvec } v1))\ (V\text{-lit } (L\text{-bitvec } v2)))\ s \rangle \longrightarrow$   
 $\langle \delta, AS\text{-let } x\ (AE\text{-val } (V\text{-lit } (L\text{-bitvec } (v1 @ v2))))\ s \rangle$
- | *reduce-let-splitI*:  $\text{split } n\ v\ (v1, v2) \Longrightarrow \Phi \vdash \langle \delta, AS\text{-let } x\ (AE\text{-split } (V\text{-lit } (L\text{-bitvec } v))\ (V\text{-lit } (L\text{-num } n)))\ s \rangle \longrightarrow$   
 $\langle \delta, AS\text{-let } x\ (AE\text{-val } (V\text{-pair } (V\text{-lit } (L\text{-bitvec } v1))\ (V\text{-lit } (L\text{-bitvec } v2))))\ s \rangle$
- | *reduce-let-lenI*:  $\Phi \vdash \langle \delta, AS\text{-let } x\ (AE\text{-len } (V\text{-lit } (L\text{-bitvec } v)))\ s \rangle \longrightarrow$   
 $\langle \delta, AS\text{-let } x\ (AE\text{-val } (V\text{-lit } (L\text{-num } (\text{int } (List.length\ v))))\ s \rangle$
- | *reduce-let-mvar*:  $(u, v) \in \text{set } \delta \Longrightarrow \Phi \vdash \langle \delta, AS\text{-let } x\ (AE\text{-mvar } u)\ s \rangle \longrightarrow \langle \delta, AS\text{-let } x\ (AE\text{-val } v)\ s \rangle$
- | *reduce-assert1I*:  $\Phi \vdash \langle \delta, AS\text{-assert } c\ (AS\text{-val } v) \rangle \longrightarrow \langle \delta, AS\text{-val } v \rangle$
- | *reduce-assert2I*:  $\Phi \vdash \langle \delta, s \rangle \longrightarrow \langle \delta', s' \rangle \Longrightarrow \Phi \vdash \langle \delta, AS\text{-assert } c\ s \rangle \longrightarrow \langle \delta', AS\text{-assert } c\ s' \rangle$
- | *reduce-varI*:  $\text{atom } u \ \# \ \delta \Longrightarrow \Phi \vdash \langle \delta, AS\text{-var } u\ \tau\ v\ s \rangle \longrightarrow \langle ((u, v) \# \delta), s \rangle$
- | *reduce-assignI*:  $\Phi \vdash \langle \delta, AS\text{-assign } u\ v \rangle \longrightarrow \langle \text{update-d } \delta\ u\ v, AS\text{-val } (V\text{-lit } L\text{-unit}) \rangle$
- | *reduce-seq1I*:  $\Phi \vdash \langle \delta, AS\text{-seq } (AS\text{-val } (V\text{-lit } L\text{-unit}))\ s \rangle \longrightarrow \langle \delta, s \rangle$
- | *reduce-seq2I*:  $\llbracket s1 \neq AS\text{-val } v ; \Phi \vdash \langle \delta, s1 \rangle \longrightarrow \langle \delta', s1' \rangle \rrbracket \Longrightarrow$   
 $\Phi \vdash \langle \delta, AS\text{-seq } s1\ s2 \rangle \longrightarrow \langle \delta', AS\text{-seq } s1'\ s2 \rangle$
- | *reduce-let2-valI*:  $\Phi \vdash \langle \delta, AS\text{-let2 } x\ t\ (AS\text{-val } v)\ s \rangle \longrightarrow \langle \delta, s[x::=v]_{sv} \rangle$
- | *reduce-let2I*:  $\Phi \vdash \langle \delta, s1 \rangle \longrightarrow \langle \delta', s1' \rangle \Longrightarrow \Phi \vdash \langle \delta, AS\text{-let2 } x\ t\ s1\ s2 \rangle \longrightarrow \langle \delta', AS\text{-let2 } x\ t\ s1'\ s2 \rangle$
- | *reduce-caseI*:  $\llbracket \text{Some } (AS\text{-branch } dc\ x'\ s') = \text{lookup-branch } dc\ cs \rrbracket \Longrightarrow \Phi \vdash \langle \delta, AS\text{-match } (V\text{-cons } tyid\ dc\ v)\ cs \rangle \longrightarrow \langle \delta, s'[x'::=v]_{sv} \rangle$
- | *reduce-whileI*:  $\llbracket \text{atom } x \ \# \ (s1, s2); \text{atom } z \ \# \ x \rrbracket \Longrightarrow$   
 $\Phi \vdash \langle \delta, AS\text{-while } s1\ s2 \rangle \longrightarrow$   
 $\langle \delta, AS\text{-let2 } x\ (\llbracket z : B\text{-bool} \mid TRUE \rrbracket) s1\ (AS\text{-if } (V\text{-var } x)\ (AS\text{-seq } s2\ (AS\text{-while } s1\ s2))\ (AS\text{-val } (V\text{-lit } L\text{-unit}))) \rangle$

**equivariance** *reduce-stmt*

**nominal-inductive** *reduce-stmt* (*proof*)

**inductive-cases** *reduce-stmt-elim*[elim]:

$\Phi \vdash \langle \delta, AS\text{-if } (V\text{-lit } L\text{-true}) s1 s2 \rangle \longrightarrow \langle \delta, s1 \rangle$   
 $\Phi \vdash \langle \delta, AS\text{-if } (V\text{-lit } L\text{-false}) s1 s2 \rangle \longrightarrow \langle \delta, s2 \rangle$   
 $\Phi \vdash \langle \delta, AS\text{-let } x (AE\text{-val } v) s \rangle \longrightarrow \langle \delta, s' \rangle$   
 $\Phi \vdash \langle \delta, AS\text{-let } x (AE\text{-op Plus } ((V\text{-lit } (L\text{-num } n1))) ((V\text{-lit } (L\text{-num } n2)))) s \rangle \longrightarrow$   
 $\langle \delta, AS\text{-let } x (AE\text{-val } (V\text{-lit } (L\text{-num } ((n1)+(n2)))))) s \rangle$   
 $\Phi \vdash \langle \delta, AS\text{-let } x ((AE\text{-op LEq } (V\text{-lit } (L\text{-num } n1)) (V\text{-lit } (L\text{-num } n2)))) s \rangle \longrightarrow \langle \delta, AS\text{-let } x (AE\text{-val } (V\text{-lit } b)) s \rangle$   
 $\Phi \vdash \langle \delta, AS\text{-let } x ((AE\text{-app } f v)) s \rangle \longrightarrow \langle \delta, AS\text{-let2 } x \tau (subst\text{-sv } s' x v) s \rangle$   
 $\Phi \vdash \langle \delta, AS\text{-let } x ((AE\text{-len } v)) s \rangle \longrightarrow \langle \delta, AS\text{-let } x v' s \rangle$   
 $\Phi \vdash \langle \delta, AS\text{-let } x ((AE\text{-concat } v1 v2)) s \rangle \longrightarrow \langle \delta, AS\text{-let } x v' s \rangle$   
 $\Phi \vdash \langle \delta, AS\text{-seq } s1 s2 \rangle \longrightarrow \langle \delta', s' \rangle$   
 $\Phi \vdash \langle \delta, AS\text{-let } x ((AE\text{-appP } f b v)) s \rangle \longrightarrow \langle \delta, AS\text{-let2 } x \tau (subst\text{-sv } s' z v) s \rangle$   
 $\Phi \vdash \langle \delta, AS\text{-let } x ((AE\text{-split } v1 v2)) s \rangle \longrightarrow \langle \delta, AS\text{-let } x v' s \rangle$   
 $\Phi \vdash \langle \delta, AS\text{-assert } c s \rangle \longrightarrow \langle \delta, s' \rangle$   
 $\Phi \vdash \langle \delta, AS\text{-let } x ((AE\text{-op Eq } (V\text{-lit } (n1)) (V\text{-lit } (n2)))) s \rangle \longrightarrow \langle \delta, AS\text{-let } x (AE\text{-val } (V\text{-lit } b)) s \rangle$

**inductive** *reduce-stmt-many* ::  $\Phi \Rightarrow \delta \Rightarrow s \Rightarrow \delta \Rightarrow s \Rightarrow bool$  ( $\langle \cdot \vdash \langle \cdot, \cdot \rangle \longrightarrow^* \langle \cdot, \cdot \rangle$  [50, 50, 50] 50) **where**

*reduce-stmt-many-oneI*:  $\Phi \vdash \langle \delta, s \rangle \longrightarrow \langle \delta', s' \rangle \implies \Phi \vdash \langle \delta, s \rangle \longrightarrow^* \langle \delta', s' \rangle$   
*reduce-stmt-many-manyI*:  $\llbracket \Phi \vdash \langle \delta, s \rangle \longrightarrow \langle \delta', s' \rangle ; \Phi \vdash \langle \delta', s' \rangle \longrightarrow^* \langle \delta'', s'' \rangle \rrbracket \implies \Phi \vdash \langle \delta, s \rangle \longrightarrow^* \langle \delta'', s'' \rangle$

**nominal-function** *convert-fds* :: *fun-def list*  $\Rightarrow$  (*f\*fun-def*) *list* **where**

*convert-fds* [] = []  
| *convert-fds* ((*AF-fundef f* (*AF-fun-typ-none* (*AF-fun-typ x b c τ s*)))#*fs*) = ((*f,AF-fundef f* (*AF-fun-typ-none* (*AF-fun-typ x b c τ s*)))#*convert-fds fs*)  
| *convert-fds* ((*AF-fundef f* (*AF-fun-typ-some* *bv* (*AF-fun-typ x b c τ s*)))#*fs*) = ((*f,AF-fundef f* (*AF-fun-typ-some* *bv* (*AF-fun-typ x b c τ s*)))#*convert-fds fs*)  
 $\langle proof \rangle$

**nominal-termination** (*eqvt*)  $\langle proof \rangle$

**nominal-function** *convert-tds* :: *type-def list*  $\Rightarrow$  (*f\*type-def*) *list* **where**

*convert-tds* [] = []  
| *convert-tds* ((*AF-typedef s dclist*)#*fs*) = ((*s,AF-typedef s dclist*)#*convert-tds fs*)  
| *convert-tds* ((*AF-typedef-poly s bv dclist*)#*fs*) = ((*s,AF-typedef-poly s bv dclist*)#*convert-tds fs*)  
 $\langle proof \rangle$

**nominal-termination** (*eqvt*)  $\langle proof \rangle$

**inductive** *reduce-prog* :: *p*  $\Rightarrow$  *v*  $\Rightarrow$  *bool* **where**

$\llbracket reduce\text{-stmt-many } \Phi \rrbracket s \delta (AS\text{-val } v) \rrbracket \implies reduce\text{-prog } (AP\text{-prog } \Theta \Phi \rrbracket s) v$

## 10.2 Reduction Typing

Checks that the store is consistent with  $\Delta$

**inductive** *delta-sim* ::  $\Theta \Rightarrow \delta \Rightarrow \Delta \Rightarrow bool$  ( $\langle \cdot \vdash \cdot \sim \cdot \rangle$  [50,50] 50) **where**

*delta-sim-nullI*:  $\Theta \vdash \llbracket \sim \rrbracket_{\Delta}$   
*delta-sim-consI*:  $\llbracket \Theta \vdash \delta \sim \Delta ; \Theta ; \{\llbracket \cdot \rrbracket\} ; GNil \vdash v \leftarrow \tau ; u \notin fst \text{ 'set } \delta \rrbracket \implies \Theta \vdash ((u,v)\#\delta) \sim ((u,\tau)\#\Delta)$

**equivariance** *delta-sim*

**nominal-inductive** *delta-sim*  $\langle proof \rangle$

**inductive-cases** *delta-sim-elim* $[elim!]$ :

$\Theta \vdash [] \sim []_{\Delta}$   
 $\Theta \vdash ((u,v)\#ds) \sim (u,\tau) \#_{\Delta} D$   
 $\Theta \vdash ((u,v)\#ds) \sim D$

A typing judgement that combines typing of the statement, the store and the condition that definitions are well-typed

**inductive** *config-type* ::  $\Theta \Rightarrow \Phi \Rightarrow \Delta \Rightarrow \delta \Rightarrow s \Rightarrow \tau \Rightarrow bool$  ( $\langle - ; - ; - \vdash \langle -, - \rangle \Leftarrow - \rangle$   $[50, 50, 50]$  50) **where**

*config-typeI*:  $\llbracket \Theta ; \Phi ; \{\} \rrbracket ; GNil ; \Delta \vdash s \Leftarrow \tau ;$   
 $(\forall fd \in set \Phi. \Theta ; \Phi \vdash fd) ;$   
 $\Theta \vdash \delta \sim \Delta \rrbracket$   
 $\implies \Theta ; \Phi ; \Delta \vdash \langle \delta, s \rangle \Leftarrow \tau$

**equivariance** *config-type*

**nominal-inductive** *config-type*  $\langle proof \rangle$

**inductive-cases** *config-type-elim*  $[elim!]$ :

$\Theta ; \Phi ; \Delta \vdash \langle \delta, s \rangle \Leftarrow \tau$

**nominal-function** *delta-of* :: *var-def list*  $\Rightarrow \delta$  **where**

$\delta\text{-of } [] = []$   
 $\mid \delta\text{-of } ((AV\text{-def } u \ t \ v)\#vs) = (u,v) \# (\delta\text{-of } vs)$   
 $\langle proof \rangle$

**nominal-termination** (*eqvt*)  $\langle proof \rangle$

**inductive** *config-type-prog* ::  $p \Rightarrow \tau \Rightarrow bool$  ( $\langle \vdash \langle - \rangle \Leftarrow - \rangle$ ) **where**

$\llbracket$   
 $\Theta ; \Phi ; \Delta\text{-of } \mathcal{G} \vdash \langle \delta\text{-of } \mathcal{G}, s \rangle \Leftarrow \tau$   
 $\rrbracket \implies \vdash \langle AP\text{-prog } \Theta \ \Phi \ \mathcal{G} \ s \rangle \Leftarrow \tau$

**inductive-cases** *config-type-prog-elim*  $[elim!]$ :

$\vdash \langle AP\text{-prog } \Theta \ \Phi \ \mathcal{G} \ s \rangle \Leftarrow \tau$

**end**

**theory** *SubstMethods*

**imports** *IVSubst WellformedL HOL-Eisbach.Eisbach-Tools*

**begin**

See *Eisbach/Examples.thy* as well as *Eisbach User Manual*.

Freshness for various substitution situations. It seems that if undirected and we throw all the facts at them to try to solve in one shot, the automatic methods are \*sometimes\* unable to handle the different variants, so some guidance is needed. First we split into subgoals using `fresh_prodN` and `intro conjI`

The 'add', for example, will be induction premises that will contain freshness facts or freshness conditions from prior obtains

Use different arguments for different things or just lump into one bucket

**method** *fresh-subst-mth-aux* **uses** *add* = (

```

  (match conclusion in atom z # (Γ::Γ)[x::=v]Γv for z x v Γ ⇒ ⟨auto simp add: fresh-subst-gv-if[of
atom z Γ v x] add⟩)
| (match conclusion in atom z # (v'::v)[x::=v]v'v for z x v v' ⇒ ⟨auto simp add: v.fresh fresh-subst-v-if
pure-fresh subst-v-v-def add⟩)
| (match conclusion in atom z # (ce::ce)[x::=v]cev for z x v ce ⇒ ⟨auto simp add: fresh-subst-v-if
subst-v-ce-def add⟩)
| (match conclusion in atom z # (Δ::Δ)[x::=v]Δv for z x v Δ ⇒ ⟨auto simp add: fresh-subst-v-if
fresh-subst-dv-if add⟩)
| (match conclusion in atom z # Γ'[x::=v]Γv @ Γ for z x v Γ' Γ ⇒ ⟨metis add ⟩)
| (match conclusion in atom z # (τ::τ)[x::=v]τv for z x v τ ⇒ ⟨auto simp add: v.fresh fresh-subst-v-if
pure-fresh subst-v-τ-def add⟩)
| (match conclusion in atom z # ({} :: bv fset) for z ⇒ ⟨auto simp add: fresh-empty-fset⟩)

| (auto simp add: add x-fresh-b pure-fresh)
)

```

```

method fresh-mth uses add = (
  (unfold fresh-prodN, intro conjI)?,
  (fresh-subst-mth-aux add: add)+)

```

```

notepad
begin
  ⟨proof⟩

```

```

end

```

```

end

```

```

hide-const Syntax.dom

```

# Chapter 11

## Refinement Constraint Logic Lemmas

### 11.1 Lemmas

**lemma** *wfI-domi*:

**assumes**  $\Theta ; \Gamma \vdash i$

**shows**  $\text{fst } \text{'toSet } \Gamma \subseteq \text{dom } i$

$\langle \text{proof} \rangle$

**lemma** *wfI-lookup*:

**fixes**  $G::\Gamma$  **and**  $b::b$

**assumes**  $\text{Some } (b,c) = \text{lookup } G \ x$  **and**  $P ; G \vdash i$  **and**  $\text{Some } s = i \ x$  **and**  $P ; B \vdash_{wf} G$

**shows**  $P \vdash s : b$

$\langle \text{proof} \rangle$

**lemma** *wfI-restrict-weakening*:

**assumes**  $wfI \ \Theta \ \Gamma' \ i'$  **and**  $i = \text{restrict-map } i' \ (\text{fst } \text{'toSet } \Gamma)$  **and**  $\text{toSet } \Gamma \subseteq \text{toSet } \Gamma'$

**shows**  $\Theta ; \Gamma \vdash i$

$\langle \text{proof} \rangle$

**lemma** *wfI-suffix*:

**fixes**  $G::\Gamma$

**assumes**  $wfI \ P \ (G'@G) \ i$  **and**  $P ; B \vdash_{wf} G$

**shows**  $P ; G \vdash i$

$\langle \text{proof} \rangle$

**lemma** *wfI-replace-inside*:

**assumes**  $wfI \ \Theta \ (\Gamma' @ (x, b, c) \#_{\Gamma} \Gamma) \ i$

**shows**  $wfI \ \Theta \ (\Gamma' @ (x, b, c') \#_{\Gamma} \Gamma) \ i$

$\langle \text{proof} \rangle$

### 11.2 Existence of evaluation

**lemma** *eval-l-base*:

$\Theta \vdash \llbracket l \rrbracket : (\text{base-for-lit } l)$

$\langle \text{proof} \rangle$

**lemma** *obtain-fresh-bv-dclist*:

**fixes**  $tm::'a::fs$

**assumes**  $(dc, \{ x : b \mid c \}) \in \text{set } dclist$

**obtains**  $bv1::bv$  **and**  $dclist1\ x1\ b1\ c1$  **where**  $AF\text{-typedef-poly } tyid\ bv\ dclist = AF\text{-typedef-poly } tyid\ bv1\ dclist1$

$\wedge (dc, \{ x1 : b1 \mid c1 \}) \in \text{set } dclist1 \wedge \text{atom } bv1 \# tm$

$\langle \text{proof} \rangle$

**lemma** *obtain-fresh-bv-dclist-b-iff*:

**fixes**  $tm::'a::fs$

**assumes**  $(dc, \{ x : b \mid c \}) \in \text{set } dclist$  **and**  $AF\text{-typedef-poly } tyid\ bv\ dclist \in \text{set } P$  **and**  $\vdash_{wf} P$

**obtains**  $bv1::bv$  **and**  $dclist1\ x1\ b1\ c1$  **where**  $AF\text{-typedef-poly } tyid\ bv\ dclist = AF\text{-typedef-poly } tyid\ bv1\ dclist1$

$\wedge (dc, \{ x1 : b1 \mid c1 \}) \in \text{set } dclist1 \wedge \text{atom } bv1 \# tm \wedge b[bv::=b']_{bb} = b1[bv1::=b']_{bb}$

$\langle \text{proof} \rangle$

**lemma** *eval-v-exist*:

**fixes**  $\Gamma::\Gamma$  **and**  $v::v$  **and**  $b::b$

**assumes**  $P ; \Gamma \vdash i$  **and**  $P ; B ; \Gamma \vdash_{wf} v : b$

**shows**  $\exists s. i \llbracket v \rrbracket \sim s \wedge P \vdash s : b$

$\langle \text{proof} \rangle$

**lemma** *eval-v-uniqueness*:

**fixes**  $v::v$

**assumes**  $i \llbracket v \rrbracket \sim s$  **and**  $i \llbracket v \rrbracket \sim s'$

**shows**  $s = s'$

$\langle \text{proof} \rangle$

**lemma** *eval-v-base*:

**fixes**  $\Gamma::\Gamma$  **and**  $v::v$  **and**  $b::b$

**assumes**  $P ; \Gamma \vdash i$  **and**  $P ; B ; \Gamma \vdash_{wf} v : b$  **and**  $i \llbracket v \rrbracket \sim s$

**shows**  $P \vdash s : b$

$\langle \text{proof} \rangle$

**lemma** *eval-e-uniqueness*:

**fixes**  $e::ce$

**assumes**  $i \llbracket e \rrbracket \sim s$  **and**  $i \llbracket e \rrbracket \sim s'$

**shows**  $s = s'$

$\langle \text{proof} \rangle$

**lemma** *wfV-eval-bitvec*:

**fixes**  $v::v$

**assumes**  $P ; B ; \Gamma \vdash_{wf} v : B\text{-bitvec}$  **and**  $P ; \Gamma \vdash i$

**shows**  $\exists bv. \text{eval-v } i\ v\ (SBitvec\ bv)$

$\langle \text{proof} \rangle$

**lemma** *wfV-eval-pair*:

**fixes**  $v::v$

**assumes**  $P ; B ; \Gamma \vdash_{wf} v : B\text{-pair } b1\ b2$  **and**  $P ; \Gamma \vdash i$

**shows**  $\exists s1\ s2. \text{eval-v } i\ v\ (SPair\ s1\ s2)$

$\langle \text{proof} \rangle$

**lemma** *wfV-eval-int*:

**fixes**  $v::v$

**assumes**  $P ; B ; \Gamma \vdash_{wf} v : B\text{-int}$  **and**  $P ; \Gamma \vdash i$

**shows**  $\exists n. \text{eval-}v\ i\ v\ (SNum\ n)$

*<proof>*

Well sorted value with a well sorted valuation evaluates

**lemma** *wfI-wfV-eval-v*:

**fixes**  $v::v$  **and**  $b::b$

**assumes**  $\Theta ; B ; \Gamma \vdash_{wf} v : b$  **and**  $wfI\ \Theta\ \Gamma\ i$

**shows**  $\exists s. i\ \llbracket v \rrbracket \sim s \wedge \Theta \vdash s : b$

*<proof>*

**lemma** *wfI-wfCE-eval-e*:

**fixes**  $e::ce$  **and**  $b::b$

**assumes**  $wfCE\ P\ B\ G\ e\ b$  **and**  $P ; G \vdash i$

**shows**  $\exists s. i\ \llbracket e \rrbracket \sim s \wedge P \vdash s : b$

*<proof>*

**lemma** *eval-e-exist*:

**fixes**  $\Gamma::\Gamma$  **and**  $e::ce$

**assumes**  $P ; \Gamma \vdash i$  **and**  $P ; B ; \Gamma \vdash_{wf} e : b$

**shows**  $\exists s. i\ \llbracket e \rrbracket \sim s$

*<proof>*

**lemma** *eval-c-exist*:

**fixes**  $\Gamma::\Gamma$  **and**  $c::c$

**assumes**  $P ; \Gamma \vdash i$  **and**  $P ; B ; \Gamma \vdash_{wf} c$

**shows**  $\exists s. i\ \llbracket c \rrbracket \sim s$

*<proof>*

**lemma** *eval-c-uniqueness*:

**fixes**  $c::c$

**assumes**  $i\ \llbracket c \rrbracket \sim s$  **and**  $i\ \llbracket c \rrbracket \sim s'$

**shows**  $s=s'$

*<proof>*

**lemma** *wfI-wfC-eval-c*:

**fixes**  $c::c$

**assumes**  $wfC\ P\ B\ G\ c$  **and**  $P ; G \vdash i$

**shows**  $\exists s. i\ \llbracket c \rrbracket \sim s$

*<proof>*

## 11.3 Satisfiability

**lemma** *satis-refl*:

**fixes**  $c::c$

**assumes**  $i \models ((x, b, c) \#_{\Gamma} G)$

**shows**  $i \models c$

*<proof>*

**lemma** *is-satis-mp*:

**fixes**  $c1::c$  **and**  $c2::c$

**assumes**  $i \models (c1 \text{ IMP } c2)$  **and**  $i \models c1$

**shows**  $i \models c2$

$\langle$ *proof* $\rangle$

**lemma** *is-satis-imp*:

**fixes**  $c1::c$  **and**  $c2::c$

**assumes**  $i \models c1 \longrightarrow i \models c2$  **and**  $i \llbracket c1 \rrbracket \sim b1$  **and**  $i \llbracket c2 \rrbracket \sim b2$

**shows**  $i \models (c1 \text{ IMP } c2)$

$\langle$ *proof* $\rangle$

**lemma** *is-satis-iff*:

$i \models G = (\forall x \ b \ c. (x, b, c) \in \text{toSet } G \longrightarrow i \models c)$

$\langle$ *proof* $\rangle$

**lemma** *is-satis-g-append*:

$i \models (G1 @ G2) = (i \models G1 \wedge i \models G2)$

$\langle$ *proof* $\rangle$

## 11.4 Substitution for Evaluation

**lemma** *eval-v-i-upd*:

**fixes**  $v::v$

**assumes**  $\text{atom } x \ \sharp \ v$  **and**  $i \llbracket v \rrbracket \sim s'$

**shows**  $\text{eval-v } ((i \ (x \mapsto s))) \ v \ s'$

$\langle$ *proof* $\rangle$

**lemma** *eval-e-i-upd*:

**fixes**  $e::ce$

**assumes**  $i \llbracket e \rrbracket \sim s'$  **and**  $\text{atom } x \ \sharp \ e$

**shows**  $(i \ (x \mapsto s)) \llbracket e \rrbracket \sim s'$

$\langle$ *proof* $\rangle$

**lemma** *eval-c-i-upd*:

**fixes**  $c::c$

**assumes**  $i \llbracket c \rrbracket \sim s'$  **and**  $\text{atom } x \ \sharp \ c$

**shows**  $((i \ (x \mapsto s)) \llbracket c \rrbracket \sim s'$

$\langle$ *proof* $\rangle$

**lemma** *subst-v-eval-v[simp]*:

**fixes**  $v::v$  **and**  $v'::v$

**assumes**  $i \llbracket v \rrbracket \sim s$  **and**  $i \llbracket (v'[x::=v]_{vv}) \rrbracket \sim s'$

**shows**  $(i \ (x \mapsto s)) \llbracket v' \rrbracket \sim s'$

$\langle$ *proof* $\rangle$

**lemma** *subst-e-eval-v[simp]*:

**fixes**  $y::x$  **and**  $e::ce$  **and**  $v::v$  **and**  $e'::ce$

**assumes**  $i \llbracket e' \rrbracket \sim s'$  **and**  $e' = (e[y::=v]_{cev})$  **and**  $i \llbracket v \rrbracket \sim s$

**shows**  $(i \ (y \mapsto s)) \llbracket e \rrbracket \sim s'$

$\langle$ *proof* $\rangle$

**lemma** *subst-c-eval-v[simp]*:

**fixes**  $v::v$  **and**  $c::c$   
**assumes**  $i \llbracket v \rrbracket \sim s$  **and**  $i \llbracket c[x::=v]_{cv} \rrbracket \sim s1$  **and**  
 $(i (x \mapsto s)) \llbracket c \rrbracket \sim s2$   
**shows**  $s1 = s2$   
 $\langle proof \rangle$

**lemma** *wfI-upd*:

**assumes**  $wfI \Theta \Gamma i$  **and**  $wfRCV \Theta s b$  **and**  $wfG \Theta B ((x, b, c) \#_{\Gamma} \Gamma)$   
**shows**  $wfI \Theta ((x, b, c) \#_{\Gamma} \Gamma) (i(x \mapsto s))$   
 $\langle proof \rangle$

**lemma** *wfI-upd-full*:

**fixes**  $v::v$   
**assumes**  $wfI \Theta G i$  **and**  $G = ((\Gamma[x::=v]_{\Gamma v})@_{\Gamma})$  **and**  $wfRCV \Theta s b$  **and**  $wfG \Theta B (\Gamma'@((x,b,c)\#_{\Gamma}\Gamma))$   
**and**  $\Theta ; B ; \Gamma \vdash_{wf} v : b$   
**shows**  $wfI \Theta (\Gamma'@((x,b,c)\#_{\Gamma}\Gamma)) (i(x \mapsto s))$   
 $\langle proof \rangle$

**lemma** *subst-c-satis[simp]*:

**fixes**  $v::v$   
**assumes**  $i \llbracket v \rrbracket \sim s$  **and**  $wfC \Theta B ((x,b,c')\#_{\Gamma}\Gamma) c$  **and**  $wfI \Theta \Gamma i$  **and**  $\Theta ; B ; \Gamma \vdash_{wf} v : b$   
**shows**  $i \models (c[x::=v]_{cv}) \longleftrightarrow (i (x \mapsto s)) \models c$   
 $\langle proof \rangle$

Key theorem telling us we can replace a substitution with an update to the valuation

**lemma** *subst-c-satis-full*:

**fixes**  $v::v$  **and**  $\Gamma::\Gamma$   
**assumes**  $i \llbracket v \rrbracket \sim s$  **and**  $wfC \Theta B (\Gamma'@((x,b,c')\#_{\Gamma}\Gamma)) c$  **and**  $wfI \Theta ((\Gamma[x::=v]_{\Gamma v})@_{\Gamma}) i$  **and**  $\Theta ; B ; \Gamma \vdash_{wf} v : b$   
**shows**  $i \models (c[x::=v]_{cv}) \longleftrightarrow (i (x \mapsto s)) \models c$   
 $\langle proof \rangle$

## 11.5 Validity

**lemma** *validI[intro]*:

**fixes**  $c::c$   
**assumes**  $wfC P B G c$  **and**  $\forall i. P ; G \vdash i \wedge i \models G \longrightarrow i \models c$   
**shows**  $P ; B ; G \models c$   
 $\langle proof \rangle$

**lemma** *valid-g-wf*:

**fixes**  $c::c$  **and**  $G::\Gamma$   
**assumes**  $P ; B ; G \models c$   
**shows**  $P ; B \vdash_{wf} G$   
 $\langle proof \rangle$

**lemma** *valid-refl [intro]*:

**fixes**  $b::b$   
**assumes**  $P ; B ; ((x,b,c1)\#_{\Gamma}G) \vdash_{wf} c1$  **and**  $c1 = c2$   
**shows**  $P ; B ; ((x,b,c1)\#_{\Gamma}G) \models c2$   
 $\langle proof \rangle$

### 11.5.1 Weakening and Strengthening

Adding to the domain of a valuation doesn't change the result

**lemma** *eval-v-weakening*:

**fixes**  $c::v$  **and**  $B::bv$  *fset*

**assumes**  $i = i' \mid' d$  **and**  $\text{supp } c \subseteq \text{atom } 'd \cup \text{supp } B$  **and**  $i \llbracket c \rrbracket \sim s$

**shows**  $i' \llbracket c \rrbracket \sim s$

*<proof>*

**lemma** *eval-v-restrict*:

**fixes**  $c::v$  **and**  $B::bv$  *fset*

**assumes**  $i = i' \mid' d$  **and**  $\text{supp } c \subseteq \text{atom } 'd \cup \text{supp } B$  **and**  $i' \llbracket c \rrbracket \sim s$

**shows**  $i \llbracket c \rrbracket \sim s$

*<proof>*

**lemma** *eval-e-weakening*:

**fixes**  $e::ce$  **and**  $B::bv$  *fset*

**assumes**  $i \llbracket e \rrbracket \sim s$  **and**  $i = i' \mid' d$  **and**  $\text{supp } e \subseteq \text{atom } 'd \cup \text{supp } B$

**shows**  $i' \llbracket e \rrbracket \sim s$

*<proof>*

**lemma** *eval-e-restrict* :

**fixes**  $e::ce$  **and**  $B::bv$  *fset*

**assumes**  $i' \llbracket e \rrbracket \sim s$  **and**  $i = i' \mid' d$  **and**  $\text{supp } e \subseteq \text{atom } 'd \cup \text{supp } B$

**shows**  $i \llbracket e \rrbracket \sim s$

*<proof>*

**lemma** *eval-c-i-weakening*:

**fixes**  $c::c$  **and**  $B::bv$  *fset*

**assumes**  $i \llbracket c \rrbracket \sim s$  **and**  $i = i' \mid' d$  **and**  $\text{supp } c \subseteq \text{atom } 'd \cup \text{supp } B$

**shows**  $i' \llbracket c \rrbracket \sim s$

*<proof>*

**lemma** *eval-c-i-restrict*:

**fixes**  $c::c$  **and**  $B::bv$  *fset*

**assumes**  $i' \llbracket c \rrbracket \sim s$  **and**  $i = i' \mid' d$  **and**  $\text{supp } c \subseteq \text{atom } 'd \cup \text{supp } B$

**shows**  $i \llbracket c \rrbracket \sim s$

*<proof>*

**lemma** *is-satis-i-weakening*:

**fixes**  $c::c$  **and**  $B::bv$  *fset*

**assumes**  $i = i' \mid' d$  **and**  $\text{supp } c \subseteq \text{atom } 'd \cup \text{supp } B$  **and**  $i \models c$

**shows**  $i' \models c$

*<proof>*

**lemma** *is-satis-i-restrict*:

**fixes**  $c::c$  **and**  $B::bv$  *fset*

**assumes**  $i = i' \mid' d$  **and**  $\text{supp } c \subseteq \text{atom } 'd \cup \text{supp } B$  **and**  $i' \models c$

**shows**  $i \models c$

*<proof>*

**lemma** *is-satis-g-restrict1*:

**fixes**  $\Gamma'::\Gamma$  **and**  $\Gamma::\Gamma$   
**assumes**  $\text{toSet } \Gamma \subseteq \text{toSet } \Gamma'$  **and**  $i \models \Gamma'$   
**shows**  $i \models \Gamma$   
 $\langle \text{proof} \rangle$

**lemma** *is-satis-g-restrict2*:

**fixes**  $\Gamma'::\Gamma$  **and**  $\Gamma::\Gamma$   
**assumes**  $i \models \Gamma$  **and**  $i' = i \mid^{\prime} d$  **and**  $\text{atom-dom } \Gamma \subseteq \text{atom } ^{\prime} d$  **and**  $\Theta ; B \vdash_{wf} \Gamma$   
**shows**  $i' \models \Gamma$   
 $\langle \text{proof} \rangle$

**lemma** *is-satis-g-restrict*:

**fixes**  $\Gamma'::\Gamma$  **and**  $\Gamma::\Gamma$   
**assumes**  $\text{toSet } \Gamma \subseteq \text{toSet } \Gamma'$  **and**  $i' \models \Gamma'$  **and**  $i = i' \mid^{\prime} (\text{fst } ^{\prime} \text{toSet } \Gamma)$  **and**  $\Theta ; B \vdash_{wf} \Gamma$   
**shows**  $i \models \Gamma$   
 $\langle \text{proof} \rangle$

## 11.5.2 Updating valuation

**lemma** *is-satis-c-i-upd*:

**fixes**  $c::c$   
**assumes**  $\text{atom } x \# c$  **and**  $i \models c$   
**shows**  $((i (x \mapsto s))) \models c$   
 $\langle \text{proof} \rangle$

**lemma** *is-satis-g-i-upd*:

**fixes**  $G::\Gamma$   
**assumes**  $\text{atom } x \# G$  **and**  $i \models G$   
**shows**  $((i (x \mapsto s))) \models G$   
 $\langle \text{proof} \rangle$

**lemma** *valid-weakening*:

**assumes**  $\Theta ; B ; \Gamma \models c$  **and**  $\text{toSet } \Gamma \subseteq \text{toSet } \Gamma'$  **and**  $\text{wfG } \Theta B \Gamma'$   
**shows**  $\Theta ; B ; \Gamma' \models c$   
 $\langle \text{proof} \rangle$

**lemma** *is-satis-g-suffix*:

**fixes**  $G::\Gamma$   
**assumes**  $i \models (G'@G)$   
**shows**  $i \models G$   
 $\langle \text{proof} \rangle$

**lemma** *wfG-inside-valid2*:

**fixes**  $x::x$  **and**  $\Gamma::\Gamma$  **and**  $c0::c$  **and**  $c0'::c$   
**assumes**  $\text{wfG } \Theta B (\Gamma'@((x,b0,c0')\#_{\Gamma}\Gamma))$  **and**  
 $\Theta ; B ; \Gamma'@((x,b0,c0)\#_{\Gamma}\Gamma) \models c0'$   
**shows**  $\text{wfG } \Theta B (\Gamma'@((x,b0,c0)\#_{\Gamma}\Gamma))$   
 $\langle \text{proof} \rangle$

**lemma** *valid-trans*:

**assumes**  $\Theta ; \mathcal{B} ; \Gamma \models c0[z::=v]_v$  **and**  $\Theta ; \mathcal{B} ; (z,b,c0)\#_{\Gamma}\Gamma \models c1$  **and**  $\text{atom } z \# \Gamma$  **and**  $\text{wfV } \Theta \mathcal{B}$   
 $\Gamma v b$   
**shows**  $\Theta ; \mathcal{B} ; \Gamma \models c1[z::=v]_v$

$\langle proof \rangle$

**lemma** *valid-trans-full*:

**assumes**  $\Theta ; \mathcal{B} ; ((x, b, c1[z1 ::= V-var x]_v) \#_{\Gamma} \Gamma) \models c2[z2 ::= V-var x]_v$  **and**  
 $\Theta ; \mathcal{B} ; ((x, b, c2[z2 ::= V-var x]_v) \#_{\Gamma} \Gamma) \models c3[z3 ::= V-var x]_v$   
**shows**  $\Theta ; \mathcal{B} ; ((x, b, c1[z1 ::= V-var x]_v) \#_{\Gamma} \Gamma) \models c3[z3 ::= V-var x]_v$   
 $\langle proof \rangle$

**lemma** *eval-v-weakening-x*:

**fixes**  $c::v$   
**assumes**  $i' \llbracket c \rrbracket \sim s$  **and**  $atom\ x \# c$  **and**  $i = i' (x \mapsto s')$   
**shows**  $i \llbracket c \rrbracket \sim s$   
 $\langle proof \rangle$

**lemma** *eval-e-weakening-x*:

**fixes**  $c::ce$   
**assumes**  $i' \llbracket c \rrbracket \sim s$  **and**  $atom\ x \# c$  **and**  $i = i' (x \mapsto s')$   
**shows**  $i \llbracket c \rrbracket \sim s$   
 $\langle proof \rangle$

**lemma** *eval-c-weakening-x*:

**fixes**  $c::c$   
**assumes**  $i' \llbracket c \rrbracket \sim s$  **and**  $atom\ x \# c$  **and**  $i = i' (x \mapsto s')$   
**shows**  $i \llbracket c \rrbracket \sim s$   
 $\langle proof \rangle$

**lemma** *is-satis-weakening-x*:

**fixes**  $c::c$   
**assumes**  $i' \models c$  **and**  $atom\ x \# c$  **and**  $i = i' (x \mapsto s)$   
**shows**  $i \models c$   
 $\langle proof \rangle$

**lemma** *is-satis-g-weakening-x*:

**fixes**  $G::\Gamma$   
**assumes**  $i' \models G$  **and**  $atom\ x \# G$  **and**  $i = i' (x \mapsto s)$   
**shows**  $i \models G$   
 $\langle proof \rangle$

## 11.6 Base Type Substitution

The idea of boxing is to take an smt val and its base type and at nodes in the smt val that correspond to type variables we wrap them in an SUT smt val node. Another way of looking at it is that s' where the node for the base type variable is an 'any node'. It is needed to prove subst\_b\_valid - the base-type variable substitution lemma for validity.

The first *rel-val* is the expanded form (has type with base-variables replaced with base-type terms) ; the second is its corresponding form

We only have one variable so we need to ensure that in all of the *bs-boxed-BVarI* cases, the s has the same base type.

For example is an SMT value is (SPair (SInt 1) (SBool true)) and it has sort (BPair (BVar x) BBool)[x::=BInt] then the boxed version is SPair (SUT (SInt 1)) (SBool true) and is has sort

(BPair (BVar x) BBool). We need to do this so that we can obtain from a valuation  $i$ , that gives values like the first smt value, to a valuation  $i'$  that gives values like the second.

**inductive** *boxed-b* ::  $\Theta \Rightarrow \text{rcl-val} \Rightarrow b \Rightarrow bv \Rightarrow b \Rightarrow \text{rcl-val} \Rightarrow \text{bool}$  (  $\langle - \vdash - \sim - [ - ::= - ] \setminus - \rangle$  [50,50] 50) **where**

*boxed-b-BVar1I*:  $\llbracket bv = bv' ; \text{wfRCV } P \ s \ b \rrbracket \Longrightarrow \text{boxed-b } P \ s \ (B\text{-var } bv') \ bv \ b \ (S\text{U}t \ s)$   
*boxed-b-BVar2I*:  $\llbracket bv \neq bv' ; \text{wfRCV } P \ s \ (B\text{-var } bv') \rrbracket \Longrightarrow \text{boxed-b } P \ s \ (B\text{-var } bv') \ bv \ b \ s$   
*boxed-b-BIntI*:  $\text{wfRCV } P \ s \ B\text{-int} \Longrightarrow \text{boxed-b } P \ s \ B\text{-int} \ - \ - \ s$   
*boxed-b-BBoolI*:  $\text{wfRCV } P \ s \ B\text{-bool} \Longrightarrow \text{boxed-b } P \ s \ B\text{-bool} \ - \ - \ s$   
*boxed-b-BUnitI*:  $\text{wfRCV } P \ s \ B\text{-unit} \Longrightarrow \text{boxed-b } P \ s \ B\text{-unit} \ - \ - \ s$   
*boxed-b-BPairI*:  $\llbracket \text{boxed-b } P \ s1 \ b1 \ bv \ b \ s1' ; \text{boxed-b } P \ s2 \ b2 \ bv \ b \ s2' \rrbracket \Longrightarrow \text{boxed-b } P \ (S\text{P}air \ s1 \ s2) \ (B\text{-pair } b1 \ b2) \ bv \ b \ (S\text{P}air \ s1' \ s2')$

| *boxed-b-BConsI*: $\llbracket$   
*AF-typedef tyid dclist*  $\in$  *set P*;  
 $(dc, \llbracket x : b \mid c \rrbracket) \in \text{set } dclist$  ;  
*boxed-b P s1 b bv b' s1'*  
 $\rrbracket \Longrightarrow$   
*boxed-b P (SCons tyid dc s1) (B-id tyid) bv b' (SCons tyid dc s1')*

| *boxed-b-BConspI*: $\llbracket$  *AF-typedef-poly tyid bva dclist*  $\in$  *set P*;  
*atom bva*  $\# (b1, bv, b', s1, s1')$ ;  
 $(dc, \llbracket x : b \mid c \rrbracket) \in \text{set } dclist$  ;  
*boxed-b P s1 (b[bva::=b1]<sub>bb</sub>) bv b' s1'*  
 $\rrbracket \Longrightarrow$   
*boxed-b P (SConsp tyid dc b1[bv::=b]<sub>bb</sub> s1) (B-app tyid b1) bv b' (SConsp tyid dc b1 s1')*

| *boxed-b-Bbitvec*:  $\text{wfRCV } P \ s \ B\text{-bitvec} \Longrightarrow \text{boxed-b } P \ s \ B\text{-bitvec} \ bv \ b \ s$

**equivariance** *boxed-b*

**nominal-inductive** *boxed-b*  $\langle \text{proof} \rangle$

**inductive-cases** *boxed-b-elim*:

*boxed-b P s (B-var bv) bv' b s'*  
*boxed-b P s B-int bv b s'*  
*boxed-b P s B-bool bv b s'*  
*boxed-b P s B-unit bv b s'*  
*boxed-b P s (B-pair b1 b2) bv b s'*  
*boxed-b P s (B-id dc) bv b s'*  
*boxed-b P s B-bitvec bv b s'*  
*boxed-b P s (B-app dc b') bv b s'*

**lemma** *boxed-b-wfRCV*:

**assumes** *boxed-b P s b bv b' s'* **and**  $\vdash_{\text{wf}} P$   
**shows**  $\text{wfRCV } P \ s \ b[bv::=b]_{bb} \wedge \text{wfRCV } P \ s' \ b$   
 $\langle \text{proof} \rangle$

**lemma** *subst-b-var*:

**assumes**  $B\text{-var } bv2 = b[bv::=b]_{bb}$   
**shows**  $(b = B\text{-var } bv \wedge b' = B\text{-var } bv2) \vee (b = B\text{-var } bv2 \wedge bv \neq bv2)$   
 $\langle \text{proof} \rangle$

Here the valuation  $i'$  is the conv wrap version of  $i$ . For every  $x$  in  $G$ ,  $i' x$  is the conv wrap

version of  $i\ x$ . The next lemma for a clearer explanation of what this is.  $i$  produces values of sort  $b[bv::=b']$  and  $i'$  produces values of sort  $b$

**inductive** *boxed-i* ::  $\Theta \Rightarrow \Gamma \Rightarrow b \Rightarrow bv \Rightarrow \text{valuation} \Rightarrow \text{valuation} \Rightarrow \text{bool} \ (\langle - ; - ; - , - \vdash - \approx \rightarrow [50,50]$   
50) **where**

*boxed-i-GNilI*:  $\Theta ; GNil ; b , bv \vdash i \approx i$   
| *boxed-i-GConsI*:  $\llbracket \text{Some } s = i\ x ; \text{boxed-b } \Theta\ s\ b\ bv\ b'\ s' ; \Theta ; \Gamma ; b' , bv \vdash i \approx i' \rrbracket \Longrightarrow \Theta ; ((x,b,c)\#\Gamma)$   
 $; b' , bv \vdash i \approx (i'(x \mapsto s'))$

**equivariance** *boxed-i*

**nominal-inductive** *boxed-i*  $\langle \text{proof} \rangle$

**inductive-cases** *boxed-i-elim*:

$\Theta ; GNil ; b , bv \vdash i \approx i'$   
 $\Theta ; ((x,b,c)\#\Gamma) ; b' , bv \vdash i \approx i'$

**lemma** *wfRCV-poly-elim*:

**fixes**  $tm::'a::fs$  **and**  $b::b$

**assumes**  $T \vdash SConsp\ \text{typid}\ dc\ bdc\ s : b$

**obtains**  $bva\ dclist\ x1\ b1\ c1$  **where**  $b = B\text{-app}\ \text{typid}\ bdc \wedge$

$AF\text{-typedef-poly}\ \text{typid}\ bva\ dclist \in \text{set } T \wedge (dc, \llbracket x1 : b1 \mid c1 \rrbracket) \in \text{set } dclist \wedge T \vdash s : b1[bva::=bdc]_{bb}$   
 $\wedge \text{atom } bva \ \#\ tm$

$\langle \text{proof} \rangle$

**lemma** *boxed-b-ex*:

**assumes**  $wfRCV\ T\ s\ b[bv::=b']_{bb}$  **and**  $wfTh\ T$

**shows**  $\exists s'. \text{boxed-b } T\ s\ b\ bv\ b'\ s'$

$\langle \text{proof} \rangle$

**lemma** *boxed-i-ex*:

**assumes**  $wfI\ T\ \Gamma[bv::=b]_{\Gamma b}\ i$  **and**  $wfTh\ T$

**shows**  $\exists i'. T ; \Gamma ; b , bv \vdash i \approx i'$

$\langle \text{proof} \rangle$

**lemma** *boxed-b-eq*:

**assumes**  $\text{boxed-b } \Theta\ s1\ b\ bv\ b'\ s1'$  **and**  $\vdash_{wf}\ \Theta$

**shows**  $wfTh\ \Theta \Longrightarrow \text{boxed-b } \Theta\ s2\ b\ bv\ b'\ s2' \Longrightarrow (s1 = s2) = (s1' = s2')$

$\langle \text{proof} \rangle$

**lemma** *bs-boxed-var*:

**assumes**  $\text{boxed-i } \Theta\ \Gamma\ b'\ bv\ i\ i'$

**shows**  $\text{Some } (b,c) = \text{lookup } \Gamma\ x \Longrightarrow \text{Some } s = i\ x \Longrightarrow \text{Some } s' = i'\ x \Longrightarrow \text{boxed-b } \Theta\ s\ b\ bv\ b'\ s'$

$\langle \text{proof} \rangle$

**lemma** *eval-l-boxed-b*:

**assumes**  $\llbracket l \rrbracket = s$

**shows**  $\text{boxed-b } \Theta\ s\ (\text{base-for-lit } l)\ bv\ b'\ s$

$\langle \text{proof} \rangle$

**lemma** *boxed-i-eval-v-boxed-b*:

**fixes**  $v::v$

**assumes**  $\text{boxed-i } \Theta\ \Gamma\ b'\ bv\ i\ i'$  **and**  $i\ \llbracket v[bv::=b']_{vb} \rrbracket \sim s$  **and**  $i'\ \llbracket v \rrbracket \sim s'$  **and**  $wfV\ \Theta\ B\ \Gamma\ v\ b$  **and**  
 $wfI\ \Theta\ \Gamma\ i'$

**shows**  $\text{boxed-b } \Theta\ s\ b\ bv\ b'\ s'$

$\langle proof \rangle$

**lemma** *boxed-b-eq-eq*:

**assumes**  $boxed-b \Theta n1 b1 bv b' n1'$  **and**  $boxed-b \Theta n2 b1 bv b' n2'$  **and**  $s = SBool (n1 = n2)$  **and**  
 $\vdash_{wf} \Theta$   
 $s' = SBool (n1' = n2')$   
**shows**  $s = s'$   
 $\langle proof \rangle$

**lemma** *boxed-i-eval-ce-boxed-b*:

**fixes**  $e::ce$   
**assumes**  $i' \llbracket e \rrbracket \sim s'$  **and**  $i \llbracket e[bv::=b]_{ceb} \rrbracket \sim s$  **and**  $wfCE \Theta B \Gamma e b$  **and**  $boxed-i \Theta \Gamma b' bv i i'$   
**and**  $wfI \Theta \Gamma i'$   
**shows**  $boxed-b \Theta s b bv b' s'$   
 $\langle proof \rangle$

**lemma** *eval-c-eq-bs-boxed*:

**fixes**  $c::c$   
**assumes**  $i \llbracket c[bv::=b]_{cb} \rrbracket \sim s$  **and**  $i' \llbracket c \rrbracket \sim s'$  **and**  $wfC \Theta B \Gamma c$  **and**  $wfI \Theta \Gamma i'$  **and**  $\Theta ; \Gamma[bv::=b]_{\Gamma b}$   
 $\vdash i$   
**and**  $boxed-i \Theta \Gamma b bv i i'$   
**shows**  $s = s'$   
 $\langle proof \rangle$

**lemma** *is-satis-bs-boxed*:

**fixes**  $c::c$   
**assumes**  $boxed-i \Theta \Gamma b bv i i'$  **and**  $wfC \Theta B \Gamma c$  **and**  $wfI \Theta \Gamma[bv::=b]_{\Gamma b} i$  **and**  $\Theta ; \Gamma \vdash i'$   
**and**  $(i \models c[bv::=b]_{cb})$   
**shows**  $(i' \models c)$   
 $\langle proof \rangle$

**lemma** *is-satis-bs-boxed-rev*:

**fixes**  $c::c$   
**assumes**  $boxed-i \Theta \Gamma b bv i i'$  **and**  $wfC \Theta B \Gamma c$  **and**  $wfI \Theta \Gamma[bv::=b]_{\Gamma b} i$  **and**  $\Theta ; \Gamma \vdash i'$  **and**  $wfC$   
 $\Theta \{|\} \Gamma[bv::=b]_{\Gamma b} (c[bv::=b]_{cb})$   
**and**  $(i' \models c)$   
**shows**  $(i \models c[bv::=b]_{cb})$   
 $\langle proof \rangle$

**lemma** *bs-boxed-wfi-aux*:

**fixes**  $b::b$  **and**  $bv::bv$  **and**  $\Theta::\Theta$  **and**  $B::\mathcal{B}$   
**assumes**  $boxed-i \Theta \Gamma b bv i i'$  **and**  $wfI \Theta \Gamma[bv::=b]_{\Gamma b} i$  **and**  $\vdash_{wf} \Theta$  **and**  $wfG \Theta B \Gamma$   
**shows**  $\Theta ; \Gamma \vdash i'$   
 $\langle proof \rangle$

**lemma** *is-satis-g-bs-boxed-aux*:

**fixes**  $G::\Gamma$   
**assumes**  $boxed-i \Theta G1 b bv i i'$  **and**  $wfI \Theta G1[bv::=b]_{\Gamma b} i$  **and**  $wfI \Theta G1 i'$  **and**  $G1 = (G2 @ G)$   
**and**  $wfG \Theta B G1$   
**and**  $(i \models G[bv::=b]_{\Gamma b})$   
**shows**  $(i' \models G)$   
 $\langle proof \rangle$

**lemma** *is-satis-g-bs-boxed*:

**fixes**  $G::\Gamma$

**assumes**  $\text{boxed-}i \ \Theta \ G \ b \ bv \ i \ i'$  **and**  $wfI \ \Theta \ G[bv::=b]_{\Gamma b} \ i$  **and**  $wfI \ \Theta \ G \ i'$  **and**  $wfG \ \Theta \ B \ G$   
**and**  $(i \models G[bv::=b]_{\Gamma b})$

**shows**  $(i' \models G)$

$\langle \text{proof} \rangle$

**lemma** *subst-b-valid*:

**fixes**  $s::s$  **and**  $b::b$

**assumes**  $\Theta ; \{|\}\vdash_{wf} b$  **and**  $B = \{|\ bv |\}$  **and**  $\Theta ; \{|\ bv |\} ; \Gamma \models c$

**shows**  $\Theta ; \{|\}\ ; \Gamma[bv::=b]_{\Gamma b} \models c[bv::=b]_{cb}$

$\langle \text{proof} \rangle$

## 11.7 Expression Operator Lemmas

**lemma** *is-satis-len-imp*:

**assumes**  $i \models (CE\text{-val} (V\text{-var } x) == CE\text{-val} (V\text{-lit} (L\text{-num} (int (length v)))) )$  **(is** *is-satis*  $i \ ?c1$ )

**shows**  $i \models (CE\text{-val} (V\text{-var } x) == CE\text{-len} [V\text{-lit} (L\text{-bitvec } v)]^{ce})$

$\langle \text{proof} \rangle$

**lemma** *is-satis-plus-imp*:

**assumes**  $i \models (CE\text{-val} (V\text{-var } x) == CE\text{-val} (V\text{-lit} (L\text{-num} (n1+n2))))$  **(is** *is-satis*  $i \ ?c1$ )

**shows**  $i \models (CE\text{-val} (V\text{-var } x) == CE\text{-op Plus} ([V\text{-lit} (L\text{-num } n1)]^{ce}) ([V\text{-lit} (L\text{-num } n2)]^{ce}))$

$\langle \text{proof} \rangle$

**lemma** *is-satis-leq-imp*:

**assumes**  $i \models (CE\text{-val} (V\text{-var } x) == CE\text{-val} (V\text{-lit} (if (n1 \leq n2) then L\text{-true} else L\text{-false})))$  **(is** *is-satis*  $i \ ?c1$ )

**shows**  $i \models (CE\text{-val} (V\text{-var } x) == CE\text{-op LEq} [(V\text{-lit} (L\text{-num } n1))]^{ce} [(V\text{-lit} (L\text{-num } n2))]^{ce})$

$\langle \text{proof} \rangle$

**lemma** *eval-lit-inj*:

**fixes**  $n1::l$  **and**  $n2::l$

**assumes**  $\llbracket n1 \rrbracket = s$  **and**  $\llbracket n2 \rrbracket = s$

**shows**  $n1=n2$

$\langle \text{proof} \rangle$

**lemma** *eval-e-lit-inj*:

**fixes**  $n1::l$  **and**  $n2::l$

**assumes**  $i \llbracket \llbracket n1 \rrbracket^v \rrbracket^{ce} \sim s$  **and**  $i \llbracket \llbracket n2 \rrbracket^v \rrbracket^{ce} \sim s$

**shows**  $n1=n2$

$\langle \text{proof} \rangle$

**lemma** *is-satis-eq-imp*:

**assumes**  $i \models (CE\text{-val} (V\text{-var } x) == CE\text{-val} (V\text{-lit} (if (n1 = n2) then L\text{-true} else L\text{-false})))$  **(is** *is-satis*  $i \ ?c1$ )

**shows**  $i \models (CE\text{-val} (V\text{-var } x) == CE\text{-op Eq} [(V\text{-lit} (n1))]^{ce} [(V\text{-lit} (n2))]^{ce})$

$\langle \text{proof} \rangle$

**lemma** *valid-eq-e*:

**assumes**  $\forall i \ s1 \ s2. \ wfG \ P \ B \ GNil \wedge \ wfI \ P \ GNil \ i \wedge \ eval\text{-e} \ i \ e1 \ s1 \wedge \ eval\text{-e} \ i \ e2 \ s2 \longrightarrow s1 = s2$

**and**  $wfCE\ P\ \mathcal{B}\ GNil\ e1\ b$  **and**  $wfCE\ P\ \mathcal{B}\ GNil\ e2\ b$   
**shows**  $P ; \mathcal{B} ; (x, b, CE\text{-val}\ (V\text{-var}\ x) == e1) \#_{\Gamma}\ GNil \models CE\text{-val}\ (V\text{-var}\ x) == e2$   
 $\langle proof \rangle$

**lemma** *valid-len*:

**assumes**  $\vdash_{wf}\ \Theta$   
**shows**  $\Theta ; \mathcal{B} ; (x, B\text{-int}, [[x]^v]^{ce} == [[L\text{-num}\ (int\ (length\ v)) ]^v]^{ce}) \#_{\Gamma}\ GNil \models [[x]^v]^{ce} == CE\text{-len}\ [[L\text{-bitvec}\ v ]^v]^{ce}$  (**is**  $\Theta ; \mathcal{B} ; ?G \models ?c$ )  
 $\langle proof \rangle$

**lemma** *valid-arith-bop*:

**assumes**  $wfG\ \Theta\ \mathcal{B}\ \Gamma$  **and**  $opp = Plus \wedge ll = (L\text{-num}\ (n1+n2)) \vee (opp = LEq \wedge ll = (if\ n1 \leq n2\ then\ L\text{-true}\ else\ L\text{-false}))$   
**and**  $(opp = Plus \longrightarrow b = B\text{-int}) \wedge (opp = LEq \longrightarrow b = B\text{-bool})$  **and**  
 $atom\ x\ \#_{\Gamma}\ \Gamma$   
**shows**  $\Theta ; \mathcal{B} ; (x, b, (CE\text{-val}\ (V\text{-var}\ x) == CE\text{-val}\ (V\text{-lit}\ (ll)))) \#_{\Gamma}\ \Gamma$   
 $\models (CE\text{-val}\ (V\text{-var}\ x) == CE\text{-op}\ opp\ ([V\text{-lit}\ (L\text{-num}\ n1)]^{ce})\ ([V\text{-lit}\ (L\text{-num}\ n2)]^{ce}))$   
**is**  $\Theta ; \mathcal{B} ; ?G \models ?c$   
 $\langle proof \rangle$

**lemma** *valid-eq-bop*:

**assumes**  $wfG\ \Theta\ \mathcal{B}\ \Gamma$  **and**  $atom\ x\ \#_{\Gamma}\ \Gamma$  **and**  $base\text{-for}\text{-lit}\ l1 = base\text{-for}\text{-lit}\ l2$   
**shows**  $\Theta ; \mathcal{B} ; (x, B\text{-bool}, (CE\text{-val}\ (V\text{-var}\ x) == CE\text{-val}\ (V\text{-lit}\ (if\ l1 = l2\ then\ L\text{-true}\ else\ L\text{-false})))) \#_{\Gamma}\ \Gamma$   
 $\models (CE\text{-val}\ (V\text{-var}\ x) == CE\text{-op}\ Eq\ ([V\text{-lit}\ (l1)]^{ce})\ ([V\text{-lit}\ (l2)]^{ce}))$  (**is**  $\Theta ; \mathcal{B} ; ?G \models ?c$ )  
 $\langle proof \rangle$

**lemma** *valid-fst*:

**fixes**  $x::x$  **and**  $v_1::v$  **and**  $v_2::v$   
**assumes**  $wfTh\ \Theta$  **and**  $wfV\ \Theta\ \mathcal{B}\ GNil\ (V\text{-pair}\ v_1\ v_2)\ (B\text{-pair}\ b_1\ b_2)$   
**shows**  $\Theta ; \mathcal{B} ; (x, b_1, [[x]^v]^{ce} == [v_1]^{ce}) \#_{\Gamma}\ GNil \models [[x]^v]^{ce} == [\#1\ [[v_1, v_2]^v]^{ce}]^{ce}$   
 $\langle proof \rangle$

**lemma** *valid-snd*:

**fixes**  $x::x$  **and**  $v_1::v$  **and**  $v_2::v$   
**assumes**  $wfTh\ \Theta$  **and**  $wfV\ \Theta\ \mathcal{B}\ GNil\ (V\text{-pair}\ v_1\ v_2)\ (B\text{-pair}\ b_1\ b_2)$   
**shows**  $\Theta ; \mathcal{B} ; (x, b_2, [[x]^v]^{ce} == [v_2]^{ce}) \#_{\Gamma}\ GNil \models [[x]^v]^{ce} == [\#2\ [[v_1, v_2]^v]^{ce}]^{ce}$   
 $\langle proof \rangle$

**lemma** *valid-concat*:

**fixes**  $v1::bit\ list$  **and**  $v2::bit\ list$   
**assumes**  $\vdash_{wf}\ \Pi$   
**shows**  $\Pi ; \mathcal{B} ; (x, B\text{-bitvec}, (CE\text{-val}\ (V\text{-var}\ x) == CE\text{-val}\ (V\text{-lit}\ (L\text{-bitvec}\ (v1 @ v2)))) \#_{\Gamma}\ GNil \models$   
 $(CE\text{-val}\ (V\text{-var}\ x) == CE\text{-concat}\ ([V\text{-lit}\ (L\text{-bitvec}\ v1)]^{ce})\ ([V\text{-lit}\ (L\text{-bitvec}\ v2)]^{ce}))$   
 $\langle proof \rangle$

**lemma** *valid-ce-eq*:

**fixes**  $ce::ce$   
**assumes**  $\Theta ; \mathcal{B} ; \Gamma \vdash_{wf}\ ce : b$   
**shows**  $(\Theta ; \mathcal{B} ; \Gamma \models ce == ce)$   
 $\langle proof \rangle$

**lemma valid-eq-imp:**

**fixes**  $c1::c$  **and**  $c2::c$

**assumes**  $\Theta ; \mathcal{B} ; (x, b, c2) \#_{\Gamma} \Gamma \vdash_{wf} c1 \text{ IMP } c2$

**shows**  $\Theta ; \mathcal{B} ; (x, b, c2) \#_{\Gamma} \Gamma \models c1 \text{ IMP } c2$

$\langle \text{proof} \rangle$

**lemma valid-range:**

**assumes**  $0 \leq n \wedge n \leq m$  **and**  $\vdash_{wf} \Theta$

**shows**  $\Theta ; \{\|\} ; (x, B\text{-int} , (C\text{-eq} (CE\text{-val} (V\text{-var } x)) (CE\text{-val} (V\text{-lit} (L\text{-num } n)))) \#_{\Gamma} GNil \models$   
 $(C\text{-eq} (CE\text{-op} LEq (CE\text{-val} (V\text{-var } x)) (CE\text{-val} (V\text{-lit} (L\text{-num } m)))) \llbracket L\text{-true} \rrbracket^v ]^{ce}) \text{ AND}$

$(C\text{-eq} (CE\text{-op} LEq (CE\text{-val} (V\text{-lit} (L\text{-num } 0))) (CE\text{-val} (V\text{-var } x))) \llbracket L\text{-true} \rrbracket^v ]^{ce})$

$(\text{is } \Theta ; \{\|\} ; ?G \models ?c1 \text{ AND } ?c2)$

$\langle \text{proof} \rangle$

**lemma valid-range-length:**

**fixes**  $\Gamma::\Gamma$

**assumes**  $0 \leq n \wedge n \leq \text{int} (\text{length } v)$  **and**  $\Theta ; \{\|\} \vdash_{wf} \Gamma$  **and**  $\text{atom } x \# \Gamma$

**shows**  $\Theta ; \{\|\} ; (x, B\text{-int} , (C\text{-eq} (CE\text{-val} (V\text{-var } x)) (CE\text{-val} (V\text{-lit} (L\text{-num } n)))) \#_{\Gamma} \Gamma \models$   
 $(C\text{-eq} (CE\text{-op} LEq (CE\text{-val} (V\text{-lit} (L\text{-num } 0))) (CE\text{-val} (V\text{-var } x))) \llbracket L\text{-true} \rrbracket^v ]^{ce})$

**AND**

$(C\text{-eq} (CE\text{-op} LEq (CE\text{-val} (V\text{-var } x)) (\llbracket [ [ L\text{-bitvec } v ]^v ]^{ce} \rrbracket^{ce} )) \llbracket L\text{-true} \rrbracket^v ]^{ce})$

$(\text{is } \Theta ; \{\|\} ; ?G \models ?c1 \text{ AND } ?c2)$

$\langle \text{proof} \rangle$

**lemma valid-range-length-inv-gnil:**

**fixes**  $\Gamma::\Gamma$

**assumes**  $\vdash_{wf} \Theta$

**and**  $\Theta ; \{\|\} ; (x, B\text{-int} , (C\text{-eq} (CE\text{-val} (V\text{-var } x)) (CE\text{-val} (V\text{-lit} (L\text{-num } n)))) \#_{\Gamma} GNil \models$   
 $(C\text{-eq} (CE\text{-op} LEq (CE\text{-val} (V\text{-lit} (L\text{-num } 0))) (CE\text{-val} (V\text{-var } x))) \llbracket L\text{-true} \rrbracket^v ]^{ce})$

**AND**

$(C\text{-eq} (CE\text{-op} LEq (CE\text{-val} (V\text{-var } x)) (\llbracket [ [ L\text{-bitvec } v ]^v ]^{ce} \rrbracket^{ce} )) \llbracket L\text{-true} \rrbracket^v ]^{ce})$

$(\text{is } \Theta ; \{\|\} ; ?G \models ?c1 \text{ AND } ?c2)$

**shows**  $0 \leq n \wedge n \leq \text{int} (\text{length } v)$

$\langle \text{proof} \rangle$

**lemma wFI-cons:**

**fixes**  $i::\text{valuation}$  **and**  $\Gamma::\Gamma$

**assumes**  $i' \models \Gamma$  **and**  $\Theta ; \Gamma \vdash i'$  **and**  $i = i' (x \mapsto s)$  **and**  $\Theta \vdash s : b$  **and**  $\text{atom } x \# \Gamma$

**shows**  $\Theta ; (x, b, c) \#_{\Gamma} \Gamma \vdash i$

$\langle \text{proof} \rangle$

**lemma valid-range-length-inv:**

**fixes**  $\Gamma::\Gamma$

**assumes**  $\Theta ; B \vdash_{wf} \Gamma$  **and**  $\text{atom } x \# \Gamma$  **and**  $\exists i. i \models \Gamma \wedge \Theta ; \Gamma \vdash i$

**and**  $\Theta ; B ; (x, B\text{-int} , (C\text{-eq} (CE\text{-val} (V\text{-var } x)) (CE\text{-val} (V\text{-lit} (L\text{-num } n)))) \#_{\Gamma} \Gamma \models$   
 $(C\text{-eq} (CE\text{-op} LEq (CE\text{-val} (V\text{-lit} (L\text{-num } 0))) (CE\text{-val} (V\text{-var } x))) \llbracket L\text{-true} \rrbracket^v ]^{ce})$

**AND**

$(C\text{-eq } (CE\text{-op } LEq (CE\text{-val } (V\text{-var } x)) ([ [ [ L\text{-bitvec } v ]^v ]^{ce} ]^{ce} ])) [ [ L\text{-true } ]^v ]^{ce}$

(is  $\Theta$  ; ?B ; ?G  $\models$  ?c1 AND ?c2)  
**shows**  $0 \leq n \wedge n \leq \text{int } (\text{length } v)$   
 $\langle \text{proof} \rangle$

**lemma** *eval-c-conj2I*[intro]:  
**assumes**  $i [ [ c1 ] ] \sim \text{True}$  **and**  $i [ [ c2 ] ] \sim \text{True}$   
**shows**  $i [ [ (C\text{-conj } c1 c2) ] ] \sim \text{True}$   
 $\langle \text{proof} \rangle$

**lemma** *valid-split*:  
**assumes** *split*  $n v (v1, v2)$  **and**  $\vdash_{wf} \Theta$   
**shows**  $\Theta$  ;  $\{\|\}$  ;  $(z, [B\text{-bitvec } , B\text{-bitvec } ]^b, [ [ z ]^v ]^{ce} == [ [ [ L\text{-bitvec } v1 ]^v , [ L\text{-bitvec } v2 ]^v ]^v ]^{ce}) \#_{\Gamma} GNil$   
 $\models ([ [ L\text{-bitvec } v ]^v ]^{ce} == [ [\#1 [ [ z ]^v ]^{ce}]^{ce} @@ [\#2 [ [ z ]^v ]^{ce}]^{ce} ]^{ce}) \text{ AND } ([ [\#1 [ [ z ]^v ]^{ce}]^{ce} ]^{ce} == [ [ L\text{-num } n ]^v ]^{ce})$   
(is  $\Theta$  ;  $\{\|\}$  ; ?G  $\models$  ?c1 AND ?c2)  
 $\langle \text{proof} \rangle$

**lemma** *is-satis-eq*:  
**assumes** *wfI*  $\Theta G i$  **and** *wfCE*  $\Theta \mathcal{B} G e b$   
**shows** *is-satis*  $i (e == e)$   
 $\langle \text{proof} \rangle$

**lemma** *is-satis-g-i-upd2*:  
**assumes** *eval-v*  $i v s$  **and** *is-satis*  $((i (x \mapsto s))) c0$  **and** *atom*  $x \# G$  **and** *wfG*  $\Theta \mathcal{B} (G\mathcal{B}@((x,b,c0)\#_{\Gamma} G))$   
**and** *wfV*  $\Theta \mathcal{B} G v b$  **and** *wfI*  $\Theta (G\mathcal{B}[x::=v]_{\Gamma v}@G) i$   
**and** *is-satis-g*  $i (G\mathcal{B}[x::=v]_{\Gamma v}@G)$   
**shows** *is-satis-g*  $(i (x \mapsto s)) (G\mathcal{B}@((x,b,c0)\#_{\Gamma} G))$   
 $\langle \text{proof} \rangle$

**end**

# Chapter 12

## Typing Lemmas

### 12.1 Prelude

Needed as the typing elimination rules give us facts for an alpha-equivalent version of a term and so need to be able to 'jump back' to a typing judgement for the original term

**lemma**  $\tau$ -fresh-c[simp]:

**assumes**  $atom\ x \# \{z : b \mid c\}$  **and**  $atom\ z \# x$   
**shows**  $atom\ x \# c$   
*<proof>*

**lemmas**  $subst-defs = subst-b-b-def\ subst-b-c-def\ subst-b-\tau-def\ subst-v-v-def\ subst-v-c-def\ subst-v-\tau-def$

**lemma**  $wfT$ - $wfT$ -if1:

**assumes**  $wfT\ \Theta\ \mathcal{B}\ \Gamma\ (\{z : b-of\ t \mid CE-val\ v == CE-val\ (V-lit\ L-false)\ IMP\ c-of\ t\ z\ \})$  **and**  $atom\ z \# (\Gamma, t)$   
**shows**  $wfT\ \Theta\ \mathcal{B}\ \Gamma\ t$   
*<proof>*

**lemma**  $fresh-u-replace-true$ :

**fixes**  $bv::bv$  **and**  $\Gamma::\Gamma$   
**assumes**  $atom\ bv \# \Gamma' @ (x, b, c) \#_{\Gamma}\ \Gamma$   
**shows**  $atom\ bv \# \Gamma' @ (x, b, TRUE) \#_{\Gamma}\ \Gamma$   
*<proof>*

**lemma**  $wf-replace-true1$ :

**fixes**  $\Gamma::\Gamma$  **and**  $\Phi::\Phi$  **and**  $\Theta::\Theta$  **and**  $\Gamma':\Gamma$  **and**  $v::v$  **and**  $e::e$  **and**  $c::c$  **and**  $c'::c$  **and**  $c'::c$  **and**  $\tau::\tau$   
**and**  $ts::(string*\tau)\ list$  **and**  $\Delta::\Delta$  **and**  $b':b$  **and**  $b::b$  **and**  $s::s$   
**and**  $ftq::fun-ty-p-q$  **and**  $ft::fun-ty-p$  **and**  $ce::ce$  **and**  $td::type-def$  **and**  $cs::branch-s$  **and**  $css::branch-list$

**shows**  $\Theta; \mathcal{B}; G \vdash_{wf} v : b' \implies G = \Gamma' @ (x, b, c) \#_{\Gamma}\ \Gamma \implies \Theta; \mathcal{B}; \Gamma' @ ((x, b, TRUE) \#_{\Gamma}\ \Gamma) \vdash_{wf} v : b'$  **and**

$\Theta; \mathcal{B}; G \vdash_{wf} c'' \implies G = \Gamma' @ (x, b, c) \#_{\Gamma}\ \Gamma \implies \Theta; \mathcal{B}; \Gamma' @ ((x, b, TRUE) \#_{\Gamma}\ \Gamma) \vdash_{wf} c''$   
**and**

$\Theta; \mathcal{B} \vdash_{wf} G \implies G = \Gamma' @ (x, b, c) \#_{\Gamma}\ \Gamma \implies \Theta; \mathcal{B} \vdash_{wf} \Gamma' @ ((x, b, TRUE) \#_{\Gamma}\ \Gamma)$  **and**  
 $\Theta; \mathcal{B}; G \vdash_{wf} \tau \implies G = \Gamma' @ (x, b, c) \#_{\Gamma}\ \Gamma \implies \Theta; \mathcal{B}; \Gamma' @ ((x, b, TRUE) \#_{\Gamma}\ \Gamma) \vdash_{wf} \tau$  **and**  
 $\Theta; \mathcal{B}; \Gamma \vdash_{wf} ts \implies True$  **and**  
 $\vdash_{wf} P \implies True$  **and**  
 $\Theta; \mathcal{B} \vdash_{wf} b \implies True$  **and**

$\Theta ; \mathcal{B} ; G \vdash_{wf} ce : b' \implies G = \Gamma' @ (x, b, c) \#_{\Gamma} \Gamma \implies \Theta ; \mathcal{B} ; \Gamma' @ ((x, b, TRUE) \#_{\Gamma} \Gamma) \vdash_{wf} ce : b' \text{ and}$   
 $\Theta \vdash_{wf} td \implies True$   
 $\langle proof \rangle$

**lemma** *wf-replace-true2*:

**fixes**  $\Gamma::\Gamma$  **and**  $\Phi::\Phi$  **and**  $\Theta::\Theta$  **and**  $\Gamma':\Gamma$  **and**  $v::v$  **and**  $e::e$  **and**  $c::c$  **and**  $c'':c$  **and**  $c':c$  **and**  $\tau::\tau$   
**and**  $ts::(string*\tau)$  **list** **and**  $\Delta::\Delta$  **and**  $b':b$  **and**  $b::b$  **and**  $s::s$   
**and**  $ftq::fun\text{-}typ\text{-}q$  **and**  $ft::fun\text{-}typ$  **and**  $ce::ce$  **and**  $td::type\text{-}def$  **and**  $cs::branch\text{-}s$  **and**  $css::branch\text{-}list$

**shows**  $\Theta ; \Phi ; \mathcal{B} ; G ; D \vdash_{wf} e : b' \implies G = \Gamma' @ (x, b, c) \#_{\Gamma} \Gamma \implies \Theta ; \Phi ; \mathcal{B} ; \Gamma' @ ((x, b, TRUE) \#_{\Gamma} \Gamma) ; D \vdash_{wf} e : b' \text{ and}$

$\Theta ; \Phi ; \mathcal{B} ; G ; \Delta \vdash_{wf} s : b' \implies G = \Gamma' @ (x, b, c) \#_{\Gamma} \Gamma \implies \Theta ; \Phi ; \mathcal{B} ; \Gamma' @ ((x, b, TRUE) \#_{\Gamma} \Gamma) ; \Delta \vdash_{wf} s : b' \text{ and}$

$\Theta ; \Phi ; \mathcal{B} ; G ; \Delta ; tid ; dc ; t \vdash_{wf} cs : b' \implies G = \Gamma' @ (x, b, c) \#_{\Gamma} \Gamma \implies \Theta ; \Phi ; \mathcal{B} ; \Gamma' @ ((x, b, TRUE) \#_{\Gamma} \Gamma) ; \Delta ; tid ; dc ; t \vdash_{wf} cs : b' \text{ and}$

$\Theta ; \Phi ; \mathcal{B} ; G ; \Delta ; tid ; dclist \vdash_{wf} css : b' \implies G = \Gamma' @ (x, b, c) \#_{\Gamma} \Gamma \implies \Theta ; \Phi ; \mathcal{B} ; \Gamma' @ ((x, b, TRUE) \#_{\Gamma} \Gamma) ; \Delta ; tid ; dclist \vdash_{wf} css : b' \text{ and}$

$\Theta \vdash_{wf} \Phi \implies True \text{ and}$

$\Theta ; \mathcal{B} ; G \vdash_{wf} \Delta \implies G = \Gamma' @ (x, b, c) \#_{\Gamma} \Gamma \implies \Theta ; \mathcal{B} ; \Gamma' @ ((x, b, TRUE) \#_{\Gamma} \Gamma) \vdash_{wf} \Delta \text{ and}$

$\Theta ; \Phi \vdash_{wf} ftq \implies True \text{ and}$

$\Theta ; \Phi ; \mathcal{B} \vdash_{wf} ft \implies True$

$\langle proof \rangle$

**lemmas** *wf-replace-true = wf-replace-true1 wf-replace-true2*

## 12.2 Subtyping

**lemma** *subtype-refl2*:

**fixes**  $\tau::\tau$

**assumes**  $\Theta ; \mathcal{B} ; \Gamma \vdash_{wf} \tau$

**shows**  $\Theta ; \mathcal{B} ; \Gamma \vdash \tau \lesssim \tau$

$\langle proof \rangle$

**lemma** *subtype-refl1*:

**assumes**  $\{ \{ z1 : b \mid c1 \} = \{ \{ z2 : b \mid c2 \} \}$  **and**  $wf1 : \Theta ; \mathcal{B} ; \Gamma \vdash_{wf} (\{ \{ z1 : b \mid c1 \} \})$

**shows**  $\Theta ; \mathcal{B} ; \Gamma \vdash (\{ \{ z1 : b \mid c1 \} \}) \lesssim (\{ \{ z2 : b \mid c2 \} \})$

$\langle proof \rangle$

**nominal-function** *base-eq*  $:: \Gamma \Rightarrow \tau \Rightarrow \tau \Rightarrow bool$  **where**

*base-eq*  $- \{ \{ z1 : b1 \mid c1 \} \} \{ \{ z2 : b2 \mid c2 \} \} = (b1 = b2)$

$\langle proof \rangle$

**nominal-termination** (*eqvt*)  $\langle proof \rangle$

**lemma** *subtype-wfT*:

**fixes**  $t1::\tau$  **and**  $t2::\tau$

**assumes**  $\Theta ; \mathcal{B} ; \Gamma \vdash t1 \lesssim t2$

**shows**  $\Theta ; \mathcal{B} ; \Gamma \vdash_{wf} t1 \wedge \Theta ; \mathcal{B} ; \Gamma \vdash_{wf} t2$

$\langle proof \rangle$

**lemma** *subtype-eq-base*:  
**assumes**  $\Theta; \mathcal{B}; \Gamma \vdash (\{ z1 : b1 \mid c1 \}) \lesssim (\{ z2 : b2 \mid c2 \})$   
**shows**  $b1=b2$   
 $\langle proof \rangle$

**lemma** *subtype-eq-base2*:  
**assumes**  $\Theta; \mathcal{B}; \Gamma \vdash t1 \lesssim t2$   
**shows**  $b\text{-of } t1 = b\text{-of } t2$   
 $\langle proof \rangle$

**lemma** *subtype-wf*:  
**fixes**  $\tau1::\tau$  **and**  $\tau2::\tau$   
**assumes**  $\Theta; \mathcal{B}; \Gamma \vdash \tau1 \lesssim \tau2$   
**shows**  $\Theta; \mathcal{B}; \Gamma \vdash_{wf} \tau1 \wedge \Theta; \mathcal{B}; \Gamma \vdash_{wf} \tau2$   
 $\langle proof \rangle$

**lemma** *subtype-g-wf*:  
**fixes**  $\tau1::\tau$  **and**  $\tau2::\tau$  **and**  $\Gamma::\Gamma$   
**assumes**  $\Theta; \mathcal{B}; \Gamma \vdash \tau1 \lesssim \tau2$   
**shows**  $\Theta; \mathcal{B} \vdash_{wf} \Gamma$   
 $\langle proof \rangle$

For when we have a particular  $y$  that satisfies the freshness conditions that we want the validity check to use

**lemma** *valid-flip-simple*:  
**assumes**  $\Theta; \mathcal{B}; (z, b, c) \#_{\Gamma} \Gamma \models c'$  **and**  $atom\ z \# \Gamma$  **and**  $atom\ x \# (z, c, z, c', \Gamma)$   
**shows**  $\Theta; \mathcal{B}; (x, b, (z \leftrightarrow x) \cdot c) \#_{\Gamma} \Gamma \models (z \leftrightarrow x) \cdot c'$   
 $\langle proof \rangle$

**lemma** *valid-wf-all*:  
**assumes**  $\Theta; \mathcal{B}; (z0, b, c0) \#_{\Gamma} G \models c$   
**shows**  $wfG\ \Theta\ \mathcal{B}\ G$  **and**  $wfC\ \Theta\ \mathcal{B}\ ((z0, b, c0) \#_{\Gamma} G)\ c$  **and**  $atom\ z0 \# G$   
 $\langle proof \rangle$

**lemma** *valid-wfT*:  
**fixes**  $z::x$   
**assumes**  $\Theta; \mathcal{B}; (z0, b, c0[z::=V\text{-var } z0]_v) \#_{\Gamma} G \models c[z::=V\text{-var } z0]_v$  **and**  $atom\ z0 \# (\Theta, \mathcal{B}, G, c, c0)$   
**shows**  $\Theta; \mathcal{B}; G \vdash_{wf} \{ z : b \mid c0 \}$  **and**  $\Theta; \mathcal{B}; G \vdash_{wf} \{ z : b \mid c \}$   
 $\langle proof \rangle$

**lemma** *valid-flip*:  
**fixes**  $c::c$  **and**  $z::x$  **and**  $z0::x$  **and**  $xx2::x$   
**assumes**  $\Theta; \mathcal{B}; (xx2, b, c0[z0::=V\text{-var } xx2]_v) \#_{\Gamma} \Gamma \models c[z::=V\text{-var } xx2]_v$  **and**  
 $atom\ xx2 \# (c0, \Gamma, c, z)$  **and**  $atom\ z0 \# (\Gamma, c, z)$   
**shows**  $\Theta; \mathcal{B}; (z0, b, c0) \#_{\Gamma} \Gamma \models c[z::=V\text{-var } z0]_v$   
 $\langle proof \rangle$

**lemma** *subtype-valid*:  
**assumes**  $\Theta; \mathcal{B}; \Gamma \vdash t1 \lesssim t2$  **and**  $atom\ y \# \Gamma$  **and**  $t1 = \{ z1 : b \mid c1 \}$  **and**  $t2 = \{ z2 : b \mid c2 \}$   
**shows**  $\Theta; \mathcal{B}; ((y, b, c1[z1::=V\text{-var } y]_v) \#_{\Gamma} \Gamma) \models c2[z2::=V\text{-var } y]_v$   
 $\langle proof \rangle$

**lemma** *subtype-valid-simple*:

**assumes**  $\Theta; \mathcal{B}; \Gamma \vdash t1 \lesssim t2$  **and**  $\text{atom } z \# \Gamma$  **and**  $t1 = \{ z : b \mid c1 \}$  **and**  $t2 = \{ z : b \mid c2 \}$   
**shows**  $\Theta; \mathcal{B}; ((z, b, c1) \#_{\Gamma} \Gamma) \models c2$   
 $\langle \text{proof} \rangle$

**lemma** *obtain-for-t-with-fresh*:

**assumes**  $\text{atom } x \# t$   
**shows**  $\exists b c. t = \{ x : b \mid c \}$   
 $\langle \text{proof} \rangle$

**lemma** *subtype-trans*:

**assumes**  $\Theta; \mathcal{B}; \Gamma \vdash \tau1 \lesssim \tau2$  **and**  $\Theta; \mathcal{B}; \Gamma \vdash \tau2 \lesssim \tau3$   
**shows**  $\Theta; \mathcal{B}; \Gamma \vdash \tau1 \lesssim \tau3$   
 $\langle \text{proof} \rangle$

**lemma** *subtype-eq-e*:

**assumes**  $\forall i s1 s2 G. \text{wf}G P \mathcal{B} G \wedge \text{wf}I P G i \wedge \text{eval-e } i e1 s1 \wedge \text{eval-e } i e2 s2 \longrightarrow s1 = s2$  **and**  
 $\text{atom } z1 \# e1$  **and**  $\text{atom } z2 \# e2$  **and**  $\text{atom } z1 \# \Gamma$  **and**  $\text{atom } z2 \# \Gamma$   
**and**  $\text{wf}CE P \mathcal{B} \Gamma e1 b$  **and**  $\text{wf}CE P \mathcal{B} \Gamma e2 b$   
**shows**  $P; \mathcal{B}; \Gamma \vdash \{ z1 : b \mid \text{CE-val } (V\text{-var } z1) == e1 \} \lesssim (\{ z2 : b \mid \text{CE-val } (V\text{-var } z2) == e2 \})$   
 $\langle \text{proof} \rangle$

**lemma** *subtype-eq-e-nil*:

**assumes**  $\forall i s1 s2 G. \text{wf}G P \mathcal{B} G \wedge \text{wf}I P G i \wedge \text{eval-e } i e1 s1 \wedge \text{eval-e } i e2 s2 \longrightarrow s1 = s2$  **and**  
 $\text{supp } e1 = \{ \}$  **and**  $\text{supp } e2 = \{ \}$  **and**  $\text{wf}Th P$   
**and**  $\text{wf}CE P \mathcal{B} GNil e1 b$  **and**  $\text{wf}CE P \mathcal{B} GNil e2 b$  **and**  $\text{atom } z1 \# GNil$  **and**  $\text{atom } z2 \# GNil$   
**shows**  $P; \mathcal{B}; GNil \vdash \{ z1 : b \mid \text{CE-val } (V\text{-var } z1) == e1 \} \lesssim (\{ z2 : b \mid \text{CE-val } (V\text{-var } z2) == e2 \})$   
 $\langle \text{proof} \rangle$

**lemma** *subtype-gnil-fst-aux*:

**assumes**  $\text{supp } v1 = \{ \}$  **and**  $\text{supp } (V\text{-pair } v1 v2) = \{ \}$  **and**  $\text{wf}Th P$  **and**  $\text{wf}CE P \mathcal{B} GNil (\text{CE-val } v1)$   
 $b$  **and**  $\text{wf}CE P \mathcal{B} GNil (\text{CE-fst } [V\text{-pair } v1 v2]^{ce}) b$  **and**  
 $\text{wf}CE P \mathcal{B} GNil (\text{CE-val } v2) b2$  **and**  $\text{atom } z1 \# GNil$  **and**  $\text{atom } z2 \# GNil$   
**shows**  $P; \mathcal{B}; GNil \vdash (\{ z1 : b \mid \text{CE-val } (V\text{-var } z1) == \text{CE-val } v1 \}) \lesssim (\{ z2 : b \mid \text{CE-val } (V\text{-var } z2) == \text{CE-fst } [V\text{-pair } v1 v2]^{ce} \})$   
 $\langle \text{proof} \rangle$

**lemma** *subtype-gnil-snd-aux*:

**assumes**  $\text{supp } v2 = \{ \}$  **and**  $\text{supp } (V\text{-pair } v1 v2) = \{ \}$  **and**  $\text{wf}Th P$  **and**  $\text{wf}CE P \mathcal{B} GNil (\text{CE-val } v2)$   
 $b$  **and**  
 $\text{wf}CE P \mathcal{B} GNil (\text{CE-snd } [(V\text{-pair } v1 v2)]^{ce}) b$  **and**  
 $\text{wf}CE P \mathcal{B} GNil (\text{CE-val } v1) b2$  **and**  $\text{atom } z1 \# GNil$  **and**  $\text{atom } z2 \# GNil$   
**shows**  $P; \mathcal{B}; GNil \vdash (\{ z1 : b \mid \text{CE-val } (V\text{-var } z1) == \text{CE-val } v2 \}) \lesssim (\{ z2 : b \mid \text{CE-val } (V\text{-var } z2) == \text{CE-snd } [(V\text{-pair } v1 v2)]^{ce} \})$   
 $\langle \text{proof} \rangle$

**lemma** *subtype-gnil-fst*:

**assumes**  $\Theta; \{ \}; GNil \vdash_{wf} [\#1[[v1, v2]^v]^{ce}]^{ce} : b$   
**shows**  $\Theta; \{ \}; GNil \vdash (\{ z1 : b \mid [[z1]^v]^{ce} == [v1]^{ce} \}) \lesssim$   
 $(\{ z2 : b \mid [[z2]^v]^{ce} == [\#1[[v1, v2]^v]^{ce}]^{ce} \})$

$\langle \text{proof} \rangle$

**lemma** *subtype-gnil-snd*:

**assumes**  $wfCE\ P\ \{\|\}\ GNil\ (CE\text{-snd}\ ([V\text{-pair}\ v_1\ v_2]^{ce}))\ b$   
**shows**  $P ; \{\|\} ; GNil \vdash (\{\!| z1 : b \mid CE\text{-val}\ (V\text{-var}\ z1) == CE\text{-val}\ v_2 \!\}) \lesssim (\{\!| z2 : b \mid CE\text{-val}\ (V\text{-var}\ z2) == CE\text{-snd}\ [(V\text{-pair}\ v_1\ v_2)]^{ce} \!\})$   
 $\langle \text{proof} \rangle$

**lemma** *subtype-fresh-tau*:

**fixes**  $x::x$   
**assumes**  $atom\ x \# t1$  **and**  $atom\ x \# \Gamma$  **and**  $P; \mathcal{B}; \Gamma \vdash t1 \lesssim t2$   
**shows**  $atom\ x \# t2$   
 $\langle \text{proof} \rangle$

**lemma** *subtype-if-simp*:

**assumes**  $wfT\ P\ \mathcal{B}\ GNil\ (\{\!| z1 : b \mid CE\text{-val}\ (V\text{-lit}\ l) == CE\text{-val}\ (V\text{-lit}\ l) \text{ IMP } c[z::=V\text{-var}\ z1]_v \!\})$  **and**  
 $wfT\ P\ \mathcal{B}\ GNil\ (\{\!| z : b \mid c \!\})$  **and**  $atom\ z1 \# c$   
**shows**  $P; \mathcal{B}; GNil \vdash (\{\!| z1 : b \mid CE\text{-val}\ (V\text{-lit}\ l) == CE\text{-val}\ (V\text{-lit}\ l) \text{ IMP } c[z::=V\text{-var}\ z1]_v \!\})$   
 $\lesssim (\{\!| z : b \mid c \!\})$   
 $\langle \text{proof} \rangle$

**lemma** *subtype-if*:

**assumes**  $P; \mathcal{B}; \Gamma \vdash \{\!| z : b \mid c \!\} \lesssim \{\!| z' : b \mid c' \!\}$  **and**  
 $wfT\ P\ \mathcal{B}\ \Gamma\ (\{\!| z1 : b \mid CE\text{-val}\ v == CE\text{-val}\ (V\text{-lit}\ l) \text{ IMP } c[z::=V\text{-var}\ z1]_v \!\})$  **and**  
 $wfT\ P\ \mathcal{B}\ \Gamma\ (\{\!| z2 : b \mid CE\text{-val}\ v == CE\text{-val}\ (V\text{-lit}\ l) \text{ IMP } c'[z'::=V\text{-var}\ z2]_v \!\})$  **and**  
 $atom\ z1 \# v$  **and**  $atom\ z \# \Gamma$  **and**  $atom\ z1 \# c$  **and**  $atom\ z2 \# c'$  **and**  $atom\ z2 \# v$   
**shows**  $P; \mathcal{B}; \Gamma \vdash \{\!| z1 : b \mid CE\text{-val}\ v == CE\text{-val}\ (V\text{-lit}\ l) \text{ IMP } c[z::=V\text{-var}\ z1]_v \!\} \lesssim \{\!| z2 : b \mid CE\text{-val}\ v == CE\text{-val}\ (V\text{-lit}\ l) \text{ IMP } c'[z'::=V\text{-var}\ z2]_v \!\}$   
 $\langle \text{proof} \rangle$

**lemma** *eval-e-concat-eq*:

**assumes**  $wfI\ \Theta\ \Gamma\ i$   
**shows**  $\exists s. eval\text{-e}\ i\ (CE\text{-val}\ (V\text{-lit}\ (L\text{-bitvec}\ (v1\ @\ v2))))\ s \wedge eval\text{-e}\ i\ (CE\text{-concat}\ [(V\text{-lit}\ (L\text{-bitvec}\ v1))]^{ce}\ [(V\text{-lit}\ (L\text{-bitvec}\ v2))]^{ce})\ s$   
 $\langle \text{proof} \rangle$

**lemma** *is-satis-eval-e-eq-imp*:

**assumes**  $wfI\ \Theta\ \Gamma\ i$  **and**  $eval\text{-e}\ i\ e1\ s$  **and**  $eval\text{-e}\ i\ e2\ s$   
**and**  $is\text{-satis}\ i\ (CE\text{-val}\ (V\text{-var}\ x) == e1)$  (**is**  $is\text{-satis}\ i\ ?c1$ )  
**shows**  $is\text{-satis}\ i\ (CE\text{-val}\ (V\text{-var}\ x) == e2)$   
 $\langle \text{proof} \rangle$

**lemma** *valid-eval-e-eq*:

**fixes**  $e1::ce$  **and**  $e2::ce$   
**assumes**  $\forall \Gamma\ i. wfI\ \Theta\ \Gamma\ i \longrightarrow (\exists s. eval\text{-e}\ i\ e1\ s \wedge eval\text{-e}\ i\ e2\ s)$  **and**  $\Theta; \mathcal{B}; GNil \vdash_{wf} e1 : b$  **and**  
 $\Theta; \mathcal{B}; GNil \vdash_{wf} e2 : b$   
**shows**  $\Theta; \mathcal{B}; (x, b, (CE\text{-val}\ (V\text{-var}\ x) == e1)) \#_{\Gamma} GNil \models (CE\text{-val}\ (V\text{-var}\ x) == e2)$   
 $\langle \text{proof} \rangle$

**lemma** *subtype-concat*:

**assumes**  $\vdash_{wf}\ \Theta$

**shows**  $\Theta; \mathcal{B}; GNil \vdash \{ z : B\text{-bitvec} \mid CE\text{-val} (V\text{-var } z) == CE\text{-val} (V\text{-lit} (L\text{-bitvec} (v1 @ v2))) \}$   
 $\lesssim$   
 $\{ z : B\text{-bitvec} \mid CE\text{-val} (V\text{-var } z) == CE\text{-concat} [(V\text{-lit} (L\text{-bitvec } v1))]^{ce} [(V\text{-lit} (L\text{-bitvec } v2))]^{ce} \}$  (**is**  $\Theta; \mathcal{B}; GNil \vdash ?t1 \lesssim ?t2$ )  
 $\langle \text{proof} \rangle$

**lemma subtype-len:**

**assumes**  $\vdash_{wf} \Theta$   
**shows**  $\Theta; \mathcal{B}; GNil \vdash \{ z' : B\text{-int} \mid CE\text{-val} (V\text{-var } z') == CE\text{-val} (V\text{-lit} (L\text{-num} (int (length v)))) \}$   
 $\lesssim$   
 $\{ z : B\text{-int} \mid CE\text{-val} (V\text{-var } z) == CE\text{-len} [(V\text{-lit} (L\text{-bitvec } v))]^{ce} \}$  (**is**  $\Theta; \mathcal{B}; GNil \vdash ?t1 \lesssim ?t2$ )  
 $\langle \text{proof} \rangle$

**lemma subtype-base-fresh:**

**assumes**  $\Theta; \mathcal{B}; \Gamma \vdash_{wf} \{ z : b \mid c \}$  **and**  $\Theta; \mathcal{B}; \Gamma \vdash_{wf} \{ z : b \mid c' \}$  **and**  
 $atom\ z \# \Gamma$  **and**  $\Theta; \mathcal{B}; (z, b, c) \#_{\Gamma} \Gamma \models c'$   
**shows**  $\Theta; \mathcal{B}; \Gamma \vdash \{ z : b \mid c \} \lesssim \{ z : b \mid c' \}$   
 $\langle \text{proof} \rangle$

**lemma subtype-bop-arith:**

**assumes**  $wfG \Theta \mathcal{B} \Gamma$  **and**  $(opp = Plus \wedge ll = (L\text{-num} (n1+n2))) \vee (opp = LEq \wedge ll = (if\ n1 \leq n2\ then\ L\text{-true}\ else\ L\text{-false}))$   
**and**  $(opp = Plus \longrightarrow b = B\text{-int}) \wedge (opp = LEq \longrightarrow b = B\text{-bool})$   
**shows**  $\Theta; \mathcal{B}; \Gamma \vdash (\{ z : b \mid C\text{-eq} (CE\text{-val} (V\text{-var } z)) (CE\text{-val} (V\text{-lit} (ll))) \} \lesssim$   
 $\{ z : b \mid C\text{-eq} (CE\text{-val} (V\text{-var } z)) (CE\text{-op } opp [(V\text{-lit} (L\text{-num } n1))]^{ce} [(V\text{-lit} (L\text{-num } n2))]^{ce}) \}$  (**is**  $\Theta; \mathcal{B}; \Gamma \vdash ?T1 \lesssim ?T2$ )  
 $\langle \text{proof} \rangle$

**lemma subtype-bop-eq:**

**assumes**  $wfG \Theta \mathcal{B} \Gamma$  **and**  $base\text{-for-lit } l1 = base\text{-for-lit } l2$   
**shows**  $\Theta; \mathcal{B}; \Gamma \vdash (\{ z : B\text{-bool} \mid C\text{-eq} (CE\text{-val} (V\text{-var } z)) (CE\text{-val} (V\text{-lit} (if\ l1 = l2\ then\ L\text{-true}\ else\ L\text{-false}))) \} \lesssim$   
 $\{ z : B\text{-bool} \mid C\text{-eq} (CE\text{-val} (V\text{-var } z)) (CE\text{-op } Eq [(V\text{-lit } l1)]^{ce} [(V\text{-lit } l2)]^{ce}) \}$  (**is**  $\Theta; \mathcal{B}; \Gamma \vdash ?T1 \lesssim ?T2$ )  
 $\langle \text{proof} \rangle$

**lemma subtype-top:**

**assumes**  $wfT \Theta \mathcal{B} G (\{ z : b \mid c \})$   
**shows**  $\Theta; \mathcal{B}; G \vdash (\{ z : b \mid c \}) \lesssim (\{ z : b \mid TRUE \})$   
 $\langle \text{proof} \rangle$

**lemma if-simp:**

$(if\ x = x\ then\ e1\ else\ e2) = e1$   
 $\langle \text{proof} \rangle$

**lemma subtype-split:**

**assumes**  $split\ n\ v\ (v1, v2)$  **and**  $\vdash_{wf} \Theta$   
**shows**  $\Theta; \{ \} \}; GNil \vdash \{ z : [ B\text{-bitvec}, B\text{-bitvec} ]^b \mid [ [ z ]^v ]^{ce} == [ [ [ L\text{-bitvec} v1 ]^v, [ L\text{-bitvec} v2 ]^v ]^v ]^{ce} \}$   
 $\lesssim \{ z : [ B\text{-bitvec}, B\text{-bitvec} ]^b \mid [ [ L\text{-bitvec} v ]^v ]^{ce} == [ [\#1 [ [ z ]^v ]^{ce} ]^{ce} @@ [\#2 [ [ z ]^v ]^{ce} ]^{ce} ]^{ce} \text{ AND } [ [\#1 [ [ z ]^v ]^{ce} ]^{ce} ]^{ce} == [$

[ *L-num*

(**is**  $\Theta ; ?B ; GNil \vdash \{ z : [ B-bitvec , B-bitvec ]^b \mid ?c1 \} \lesssim \{ z : [ B-bitvec , B-bitvec ]^b \mid ?c2 \}$ )  
 ⟨*proof*⟩

**lemma** *subtype-range*:

**fixes**  $n::int$  and  $\Gamma::\Gamma$

**assumes**  $0 \leq n \wedge n \leq int (length\ v)$  and  $\Theta ; \{\|\} \vdash_{wf} \Gamma$

**shows**  $\Theta ; \{\|\} ; \Gamma \vdash \{ z : B-int \mid [ [ z ]^v ]^{ce} == [ [ L-num\ n ]^v ]^{ce} \} \lesssim$   
 $\{ z : B-int \mid ([ leq [ [ L-num\ 0 ]^v ]^{ce} [ [ z ]^v ]^{ce} ]^{ce} == [ [ L-true ]^v ]^{ce} ) \text{ AND } ($   
 $[ leq [ [ z ]^v ]^{ce} [ [ [ L-bitvec\ v ]^v ]^{ce} ]^{ce} ]^{ce} == [ [ L-true ]^v ]^{ce} ) \}$

(**is**  $\Theta ; ?B ; \Gamma \vdash \{ z : B-int \mid ?c1 \} \lesssim \{ z : B-int \mid ?c2 \text{ AND } ?c3 \}$ )

⟨*proof*⟩

**lemma** *check-num-range*:

**assumes**  $0 \leq n \wedge n \leq int (length\ v)$  and  $\vdash_{wf} \Theta$

**shows**  $\Theta ; \{\|\} ; GNil \vdash ([ L-num\ n ]^v) \leftarrow \{ z : B-int \mid ([ leq [ [ L-num\ 0 ]^v ]^{ce} [ [ z ]^v ]^{ce} ]^{ce} == [ [ L-true ]^v ]^{ce} ) \text{ AND}$

$[ leq [ [ z ]^v ]^{ce} [ [ [ L-bitvec\ v ]^v ]^{ce} ]^{ce} ]^{ce} == [ [ L-true ]^v ]^{ce} \}$

⟨*proof*⟩

## 12.3 Literals

**nominal-function** *type-for-lit* ::  $l \Rightarrow \tau$  **where**

*type-for-lit* (*L-true*) = ( $\{ z : B-bool \mid [[z]^v]^{ce} == [V-lit\ L-true]^{ce} \}$ )

| *type-for-lit* (*L-false*) = ( $\{ z : B-bool \mid [[z]^v]^{ce} == [V-lit\ L-false]^{ce} \}$ )

| *type-for-lit* (*L-num*  $n$ ) = ( $\{ z : B-int \mid [[z]^v]^{ce} == [V-lit\ (L-num\ n)]^{ce} \}$ )

| *type-for-lit* (*L-unit*) = ( $\{ z : B-unit \mid [[z]^v]^{ce} == [V-lit\ (L-unit)]^{ce} \}$ )

| *type-for-lit* (*L-bitvec*  $v$ ) = ( $\{ z : B-bitvec \mid [[z]^v]^{ce} == [V-lit\ (L-bitvec\ v)]^{ce} \}$ )

⟨*proof*⟩

**nominal-termination** (*eqvt*) ⟨*proof*⟩

**nominal-function** *type-for-var* ::  $\Gamma \Rightarrow \tau \Rightarrow x \Rightarrow \tau$  **where**

*type-for-var*  $G\ \tau\ x = (case\ lookup\ G\ x\ of$

*None*  $\Rightarrow \tau$

| *Some* ( $b, c$ )  $\Rightarrow (\{ x : b \mid c \})$ )

⟨*proof*⟩

**nominal-termination** (*eqvt*) ⟨*proof*⟩

**lemma** *infer-l-form*:

**fixes**  $l::l$  and  $tm::'a::fs$

**assumes**  $\vdash l \Rightarrow \tau$

**shows**  $\exists z\ b. \tau = (\{ z : b \mid C-eq\ (CE-val\ (V-var\ z))\ (CE-val\ (V-lit\ l)) \}) \wedge atom\ z \# tm$

⟨*proof*⟩

**lemma** *infer-l-form3*:

**fixes**  $l::l$

**assumes**  $\vdash l \Rightarrow \tau$

**shows**  $\exists z. \tau = (\{ z : base-for-lit\ l \mid C-eq\ (CE-val\ (V-var\ z))\ (CE-val\ (V-lit\ l)) \})$

⟨*proof*⟩

**lemma** *infer-l-form4* [*simp*]:

**fixes**  $\Gamma::\Gamma$

**assumes**  $\Theta ; \mathcal{B} \vdash_{wf} \Gamma$

**shows**  $\exists z. \vdash l \Rightarrow (\{ z : \text{base-for-lit } l \mid C\text{-eq } (CE\text{-val } (V\text{-var } z)) (CE\text{-val } (V\text{-lit } l)) \})$

$\langle \text{proof} \rangle$

**lemma** *infer-v-unit-form*:

**fixes**  $v::v$

**assumes**  $P ; \mathcal{B} ; \Gamma \vdash v \Rightarrow (\{ z1 : B\text{-unit} \mid c1 \})$  **and**  $\text{supp } v = \{ \}$

**shows**  $v = V\text{-lit } L\text{-unit}$

$\langle \text{proof} \rangle$

**lemma** *base-for-lit-wf*:

**assumes**  $\vdash_{wf} \Theta$

**shows**  $\Theta ; \mathcal{B} \vdash_{wf} \text{base-for-lit } l$

$\langle \text{proof} \rangle$

**lemma** *infer-l-t-wf*:

**fixes**  $\Gamma::\Gamma$

**assumes**  $\Theta ; \mathcal{B} \vdash_{wf} \Gamma \wedge \text{atom } z \not\# \Gamma$

**shows**  $\Theta ; \mathcal{B} ; \Gamma \vdash_{wf} \{ z : \text{base-for-lit } l \mid C\text{-eq } (CE\text{-val } (V\text{-var } z)) (CE\text{-val } (V\text{-lit } l)) \}$

$\langle \text{proof} \rangle$

**lemma** *infer-l-wf*:

**fixes**  $l::l$  **and**  $\Gamma::\Gamma$  **and**  $\tau::\tau$  **and**  $\Theta::\Theta$

**assumes**  $\vdash l \Rightarrow \tau$  **and**  $\Theta ; \mathcal{B} \vdash_{wf} \Gamma$

**shows**  $\vdash_{wf} \Theta$  **and**  $\Theta ; \mathcal{B} \vdash_{wf} \Gamma$  **and**  $\Theta ; \mathcal{B} ; \Gamma \vdash_{wf} \tau$

$\langle \text{proof} \rangle$

**lemma** *infer-l-uniqueness*:

**fixes**  $l::l$

**assumes**  $\vdash l \Rightarrow \tau$  **and**  $\vdash l \Rightarrow \tau'$

**shows**  $\tau = \tau'$

$\langle \text{proof} \rangle$

## 12.4 Values

**lemma** *type-v-eq*:

**assumes**  $\{ z1 : b1 \mid c1 \} = \{ z : b \mid C\text{-eq } (CE\text{-val } (V\text{-var } z)) (CE\text{-val } (V\text{-var } x)) \}$  **and**  $\text{atom } z \not\# x$

**shows**  $b = b1$  **and**  $c1 = C\text{-eq } (CE\text{-val } (V\text{-var } z1)) (CE\text{-val } (V\text{-var } x))$

$\langle \text{proof} \rangle$

**lemma** *infer-var2* [*elim*]:

**assumes**  $P ; \mathcal{B} ; G \vdash V\text{-var } x \Rightarrow \tau$

**shows**  $\exists b c. \text{Some } (b,c) = \text{lookup } G x$

$\langle \text{proof} \rangle$

**lemma** *infer-var3* [*elim*]:

**assumes**  $\Theta ; \mathcal{B} ; \Gamma \vdash V\text{-var } x \Rightarrow \tau$

**shows**  $\exists z b c. \text{Some } (b,c) = \text{lookup } \Gamma x \wedge \tau = (\{ z : b \mid C\text{-eq } (CE\text{-val } (V\text{-var } z)) (CE\text{-val } (V\text{-var } x)) \}) \wedge \text{atom } z \not\# x \wedge \text{atom } z \not\# (\Theta, \mathcal{B}, \Gamma)$

$\langle \text{proof} \rangle$

**lemma** *infer-bool-options2*:

**fixes**  $v::v$

**assumes**  $\Theta; \mathcal{B}; \Gamma \vdash v \Rightarrow \{\!| z : b \mid c \!\!\}$  **and**  $\text{supp } v = \{\}$   $\wedge b = B\text{-bool}$

**shows**  $v = V\text{-lit } L\text{-true} \vee (v = (V\text{-lit } L\text{-false}))$

$\langle \text{proof} \rangle$

**lemma** *infer-bool-options*:

**fixes**  $v::v$

**assumes**  $\Theta; \mathcal{B}; \Gamma \vdash v \Rightarrow \{\!| z : B\text{-bool} \mid c \!\!\}$  **and**  $\text{supp } v = \{\}$

**shows**  $v = V\text{-lit } L\text{-true} \vee (v = (V\text{-lit } L\text{-false}))$

$\langle \text{proof} \rangle$

**lemma** *infer-int2*:

**fixes**  $v::v$

**assumes**  $\Theta; \mathcal{B}; \Gamma \vdash v \Rightarrow \{\!| z : b \mid c \!\!\}$

**shows**  $\text{supp } v = \{\} \wedge b = B\text{-int} \longrightarrow (\exists n. v = V\text{-lit } (L\text{-num } n))$

$\langle \text{proof} \rangle$

**lemma** *infer-bitvec*:

**fixes**  $\Theta::\Theta$  **and**  $v::v$

**assumes**  $\Theta; \mathcal{B}; \Gamma \vdash v \Rightarrow \{\!| z' : B\text{-bitvec} \mid c' \!\!\}$  **and**  $\text{supp } v = \{\}$

**shows**  $\exists bv. v = V\text{-lit } (L\text{-bitvec } bv)$

$\langle \text{proof} \rangle$

**lemma** *infer-int*:

**assumes** *infer-v*  $\Theta \mathcal{B} \Gamma v (\{\!| z : B\text{-int} \mid c \!\!\})$  **and**  $\text{supp } v = \{\}$

**shows**  $\exists n. V\text{-lit } (L\text{-num } n) = v$

$\langle \text{proof} \rangle$

**lemma** *infer-lit*:

**assumes** *infer-v*  $\Theta \mathcal{B} \Gamma v (\{\!| z : b \mid c \!\!\})$  **and**  $\text{supp } v = \{\}$  **and**  $b \in \{ B\text{-bool}, B\text{-int}, B\text{-unit} \}$

**shows**  $\exists l. V\text{-lit } l = v$

$\langle \text{proof} \rangle$

**lemma** *infer-v-form[simp]*:

**fixes**  $v::v$

**assumes**  $\Theta; \mathcal{B}; \Gamma \vdash v \Rightarrow \tau$

**shows**  $\exists z b. \tau = (\{\!| z : b \mid C\text{-eq } (CE\text{-val } (V\text{-var } z)) (CE\text{-val } v) \!\!\}) \wedge \text{atom } z \# v \wedge \text{atom } z \# (\Theta, \mathcal{B}, \Gamma)$

$\langle \text{proof} \rangle$

**lemma** *infer-v-form2*:

**fixes**  $v::v$

**assumes**  $\Theta; \mathcal{B}; \Gamma \vdash v \Rightarrow (\{\!| z : b \mid c \!\!\})$  **and**  $\text{atom } z \# v$

**shows**  $c = C\text{-eq } (CE\text{-val } (V\text{-var } z)) (CE\text{-val } v)$

$\langle \text{proof} \rangle$

**lemma** *infer-v-form3*:

**fixes**  $v::v$

**assumes**  $\Theta; \mathcal{B}; \Gamma \vdash v \Rightarrow \tau$  **and**  $\text{atom } z \# (v, \Gamma)$

**shows**  $\Theta; \mathcal{B}; \Gamma \vdash v \Rightarrow \{\!| z : b\text{-of } \tau \mid C\text{-eq } (CE\text{-val } (V\text{-var } z)) (CE\text{-val } v) \!\!\}$

$\langle \text{proof} \rangle$

**lemma** *infer-v-form4*:

**fixes**  $v::v$

**assumes**  $\Theta; \mathcal{B}; \Gamma \vdash v \Rightarrow \tau$  **and**  $\text{atom } z \# (v, \Gamma)$  **and**  $b = \text{b-of } \tau$

**shows**  $\Theta; \mathcal{B}; \Gamma \vdash v \Rightarrow \{ z : b \mid C\text{-eq } (CE\text{-val } (V\text{-var } z)) (CE\text{-val } v) \}$

$\langle \text{proof} \rangle$

**lemma** *infer-v-v-wf*:

**fixes**  $v::v$

**shows**  $\Theta; \mathcal{B}; G \vdash v \Rightarrow \tau \Longrightarrow \Theta; \mathcal{B}; G \vdash_{wf} v : (\text{b-of } \tau)$

$\langle \text{proof} \rangle$

**lemma** *infer-v-t-form-wf*:

**assumes**  $\text{wfB } \Theta \mathcal{B} b$  **and**  $\text{wfV } \Theta \mathcal{B} \Gamma v b$  **and**  $\text{atom } z \# \Gamma$

**shows**  $\text{wfT } \Theta \mathcal{B} \Gamma \{ z : b \mid C\text{-eq } (CE\text{-val } (V\text{-var } z)) (CE\text{-val } v) \}$

$\langle \text{proof} \rangle$

**lemma** *infer-v-t-wf*:

**fixes**  $v::v$

**assumes**  $\Theta; \mathcal{B}; G \vdash v \Rightarrow \tau$

**shows**  $\text{wfT } \Theta \mathcal{B} G \tau \wedge \text{wfB } \Theta \mathcal{B} (\text{b-of } \tau)$

$\langle \text{proof} \rangle$

**lemma** *infer-v-wf*:

**fixes**  $v::v$

**assumes**  $\Theta; \mathcal{B}; G \vdash v \Rightarrow \tau$

**shows**  $\Theta; \mathcal{B}; G \vdash_{wf} v : (\text{b-of } \tau)$  **and**  $\text{wfT } \Theta \mathcal{B} G \tau$  **and**  $\text{wfTh } \Theta$  **and**  $\text{wfG } \Theta \mathcal{B} G$

$\langle \text{proof} \rangle$

**lemma** *check-bool-options*:

**assumes**  $\Theta; \mathcal{B}; \Gamma \vdash v \Leftarrow \{ z : B\text{-bool} \mid \text{TRUE} \}$  **and**  $\text{supp } v = \{ \}$

**shows**  $v = V\text{-lit } L\text{-true} \vee v = V\text{-lit } L\text{-false}$

$\langle \text{proof} \rangle$

**lemma** *check-v-wf*:

**fixes**  $v::v$  **and**  $\Gamma::\Gamma$  **and**  $\tau::\tau$

**assumes**  $\Theta; \mathcal{B}; \Gamma \vdash v \Leftarrow \tau$

**shows**  $\Theta; \mathcal{B} \vdash_{wf} \Gamma$  **and**  $\Theta; \mathcal{B}; \Gamma \vdash_{wf} v : \text{b-of } \tau$  **and**  $\Theta; \mathcal{B}; \Gamma \vdash_{wf} \tau$

$\langle \text{proof} \rangle$

**lemma** *infer-v-form-fresh*:

**fixes**  $v::v$  **and**  $t::'a::fs$

**assumes**  $\Theta; \mathcal{B}; \Gamma \vdash v \Rightarrow \tau$

**shows**  $\exists z b. \tau = \{ z : b \mid C\text{-eq } (CE\text{-val } (V\text{-var } z)) (CE\text{-val } v) \} \wedge \text{atom } z \# (t, v)$

$\langle \text{proof} \rangle$

More generally, if support of a term is empty then any G will do

**lemma** *infer-v-form-consp*:

**assumes**  $\Theta; \mathcal{B}; \Gamma \vdash V\text{-consp } s \text{ dc } b \text{ v} \Rightarrow \tau$

**shows**  $\text{b-of } \tau = B\text{-app } s \text{ b}$

$\langle \text{proof} \rangle$

**lemma** *lookup-in-rig-b*:

**assumes** *Some* ( $b2, c2$ ) = *lookup* ( $\Gamma[x \mapsto c']$ )  $x'$  **and**

*Some* ( $b1, c1$ ) = *lookup*  $\Gamma x'$

**shows**  $b1 = b2$

*<proof>*

**lemma** *infer-v-uniqueness-rig*:

**fixes**  $x::x$  **and**  $c::c$

**assumes** *infer-v*  $P B G v \tau$  **and** *infer-v*  $P B$  (*replace-in-g*  $G x c'$ )  $v \tau'$

**shows**  $\tau = \tau'$

*<proof>*

**lemma** *infer-v-uniqueness*:

**assumes** *infer-v*  $P B G v \tau$  **and** *infer-v*  $P B G v \tau'$

**shows**  $\tau = \tau'$

*<proof>*

**lemma** *infer-v-tid-form*:

**fixes**  $v::v$

**assumes**  $\Theta ; B ; \Gamma \vdash v \Rightarrow \{ z : B\text{-id } tid \mid c \}$  **and** *AF-typedef*  $tid$   $dclist \in set \Theta$  **and** *supp*  $v = \{ \}$

**shows**  $\exists dc v' t. v = V\text{-cons } tid \ dc \ v' \wedge (dc, t) \in set \ dclist$

*<proof>*

**lemma** *check-v-tid-form*:

**assumes**  $\Theta ; B ; \Gamma \vdash v \Leftarrow \{ z : B\text{-id } tid \mid TRUE \}$  **and** *AF-typedef*  $tid$   $dclist \in set \Theta$  **and** *supp*  $v = \{ \}$

**shows**  $\exists dc v' t. v = V\text{-cons } tid \ dc \ v' \wedge (dc, t) \in set \ dclist$

*<proof>*

**lemma** *check-v-num-leq*:

**fixes**  $n::int$  **and**  $\Gamma::\Gamma$

**assumes**  $0 \leq n \wedge n \leq int \ (length \ v)$  **and**  $\vdash_{wf} \Theta$  **and**  $\Theta ; \{ \} \vdash_{wf} \Gamma$

**shows**  $\Theta ; \{ \} ; \Gamma \vdash [L\text{-num } n]^v \Leftarrow \{ z : B\text{-int} \mid ([leq \ [ \ [L\text{-num } 0]^v ]^{ce} \ [ [z]^v ]^{ce} ]^{ce} == [ [L\text{-true}]^v ]^{ce} )$

*AND*  $([leq \ [ [z]^v ]^{ce} \ [ [ [L\text{-bitvec } v]^v ]^{ce} ]^{ce} ]^{ce} == [ [L\text{-true}]^v ]^{ce} ) \}$

*<proof>*

**lemma** *check-int*:

**assumes** *check-v*  $\Theta \mathcal{B} \Gamma v$  ( $\{ z : B\text{-int} \mid c \}$ ) **and** *supp*  $v = \{ \}$

**shows**  $\exists n. V\text{-lit } (L\text{-num } n) = v$

*<proof>*

**definition** *sble* ::  $\Theta \Rightarrow \Gamma \Rightarrow bool$  **where**

*sble*  $\Theta \Gamma = (\exists i. i \models \Gamma \wedge \Theta ; \Gamma \vdash i)$

**lemma** *check-v-range*:

**assumes**  $\Theta ; B ; \Gamma \vdash v2 \Leftarrow \{ z : B\text{-int} \mid [leq \ [ [L\text{-num } 0]^v ]^{ce} \ [ [z]^v ]^{ce} ]^{ce} == [ [L\text{-true}]^v ]^{ce}$

*AND*

$[leq \ [ [z]^v ]^{ce} \ [ [v1]^{ce} ]^{ce} ]^{ce} == [ [L\text{-true}]^v ]^{ce} \}$

(*is*  $\Theta ; ?B ; \Gamma \vdash v2 \Leftarrow \{ z : B\text{-int} \mid ?c1 \}$ )

**and**  $v1 = V\text{-lit } (L\text{-bitvec } bv) \wedge v2 = V\text{-lit } (L\text{-num } n)$  **and** *atom*  $z \not\# \Gamma$  **and** *sble*  $\Theta \Gamma$

**shows**  $0 \leq n \wedge n \leq int \ (length \ bv)$

$\langle \text{proof} \rangle$

## 12.5 Expressions

**lemma** *infer-e-plus*[*elim*]:

**fixes**  $v1::v$  **and**  $v2::v$

**assumes**  $\Theta ; \Phi ; \mathcal{B} ; \Gamma ; \Delta \vdash AE\text{-op Plus } v1 \ v2 \Rightarrow \tau$

**shows**  $\exists z . (\{ z : B\text{-int} \mid C\text{-eq } (CE\text{-val } (V\text{-var } z)) (CE\text{-op Plus } [v1]^{ce} [v2]^{ce}) \} = \tau)$

$\langle \text{proof} \rangle$

**lemma** *infer-e-leq*[*elim*]:

**assumes**  $\Theta ; \Phi ; \mathcal{B} ; \Gamma ; \Delta \vdash AE\text{-op LEq } v1 \ v2 \Rightarrow \tau$

**shows**  $\exists z . (\{ z : B\text{-bool} \mid C\text{-eq } (CE\text{-val } (V\text{-var } z)) (CE\text{-op LEq } [v1]^{ce} [v2]^{ce}) \} = \tau)$

$\langle \text{proof} \rangle$

**lemma** *infer-e-eq*[*elim*]:

**assumes**  $\Theta ; \Phi ; \mathcal{B} ; \Gamma ; \Delta \vdash AE\text{-op Eq } v1 \ v2 \Rightarrow \tau$

**shows**  $\exists z . (\{ z : B\text{-bool} \mid C\text{-eq } (CE\text{-val } (V\text{-var } z)) (CE\text{-op Eq } [v1]^{ce} [v2]^{ce}) \} = \tau)$

$\langle \text{proof} \rangle$

**lemma** *infer-e-e-wf*:

**fixes**  $e::e$

**assumes**  $\Theta ; \Phi ; \mathcal{B} ; \Gamma ; \Delta \vdash e \Rightarrow \tau$

**shows**  $\Theta ; \Phi ; \mathcal{B} ; \Gamma ; \Delta \vdash_{wf} e : b\text{-of } \tau$

$\langle \text{proof} \rangle$

**lemma** *infer-e-t-wf*:

**fixes**  $e::e$  **and**  $\Gamma::\Gamma$  **and**  $\tau::\tau$  **and**  $\Delta::\Delta$  **and**  $\Phi::\Phi$

**assumes**  $\Theta ; \Phi ; \mathcal{B} ; \Gamma ; \Delta \vdash e \Rightarrow \tau$

**shows**  $\Theta ; \mathcal{B}; \Gamma \vdash_{wf} \tau \wedge \Theta \vdash_{wf} \Phi$

$\langle \text{proof} \rangle$

**lemma** *infer-e-wf*:

**fixes**  $e::e$  **and**  $\Gamma::\Gamma$  **and**  $\tau::\tau$  **and**  $\Delta::\Delta$  **and**  $\Phi::\Phi$

**assumes**  $\Theta ; \Phi ; \mathcal{B} ; \Gamma ; \Delta \vdash e \Rightarrow \tau$

**shows**  $\Theta ; \mathcal{B}; \Gamma \vdash_{wf} \tau$  **and**  $\Theta ; \mathcal{B} \vdash_{wf} \Gamma$  **and**  $\Theta ; \mathcal{B}; \Gamma \vdash_{wf} \Delta$  **and**  $\Theta \vdash_{wf} \Phi$  **and**  $\Theta ; \Phi ; \mathcal{B} ; \Gamma ; \Delta \vdash_{wf} e : (b\text{-of } \tau)$

$\langle \text{proof} \rangle$

**lemma** *infer-e-fresh*:

**fixes**  $x::x$

**assumes**  $\Theta ; \Phi ; \mathcal{B} ; \Gamma ; \Delta \vdash e \Rightarrow \tau$  **and**  $\text{atom } x \# \Gamma$

**shows**  $\text{atom } x \# (e, \tau)$

$\langle \text{proof} \rangle$

**inductive** *check-e* ::  $\Theta \Rightarrow \Phi \Rightarrow \mathcal{B} \Rightarrow \Gamma \Rightarrow \Delta \Rightarrow e \Rightarrow \tau \Rightarrow \text{bool}$  ( $\langle - ; - ; - ; - ; - \vdash - \Leftarrow \rightarrow [50, 50, 50] 50 \rangle$  **where**

*check-e-subtypeI*:  $\llbracket \text{infer-e } T P B G D e \tau' ; \text{subtype } T B G \tau' \tau \rrbracket \Longrightarrow \text{check-e } T P B G D e \tau$

**equivariance** *check-e*

**nominal-inductive** *check-e*  $\langle \text{proof} \rangle$

**inductive-cases** *check-e-elim*[*elim*!]:

$check-e\ F\ D\ B\ G\ \Theta\ (AE-val\ v)\ \tau$   
 $check-e\ F\ D\ B\ G\ \Theta\ e\ \tau$

**lemma** *infer-e-fst-pair*:

**fixes**  $v1::v$

**assumes**  $\Theta ; \Phi ; \{\|\}; GNil ; \Delta \vdash [\#1[ v1 , v2 ]^v]^e \Rightarrow \tau$

**shows**  $\exists \tau'. \Theta ; \Phi ; \{\|\}; GNil ; \Delta \vdash [v1]^e \Rightarrow \tau' \wedge$

$\Theta ; \{\|\}; GNil \vdash \tau' \lesssim \tau$

$\langle proof \rangle$

**lemma** *infer-e-snd-pair*:

**assumes**  $\Theta ; \Phi ; \{\|\}; GNil ; \Delta \vdash AE-snd\ (V-pair\ v1\ v2) \Rightarrow \tau$

**shows**  $\exists \tau'. \Theta ; \Phi ; \{\|\}; GNil ; \Delta \vdash AE-val\ v2 \Rightarrow \tau' \wedge \Theta ; \{\|\}; GNil \vdash \tau' \lesssim \tau$

$\langle proof \rangle$

## 12.6 Statements

**lemma** *check-s-v-unit*:

**assumes**  $\Theta ; \mathcal{B} ; \Gamma \vdash (\{ z : B-unit \mid TRUE \}) \lesssim \tau$  **and**  $wfD\ \Theta\ \mathcal{B}\ \Gamma\ \Delta$  **and**  $wfPhi\ \Theta\ \Phi$

**shows**  $\Theta ; \Phi ; \mathcal{B} ; \Gamma ; \Delta \vdash AS-val\ (V-lit\ L-unit) \Leftarrow \tau$

$\langle proof \rangle$

**lemma** *check-s-check-branch-s-wf*:

**fixes**  $s::s$  **and**  $cs::branch-s$  **and**  $\Theta::\Theta$  **and**  $\Phi::\Phi$  **and**  $\Gamma::\Gamma$  **and**  $\Delta::\Delta$  **and**  $v::v$  **and**  $\tau::\tau$  **and**  $css::branch-list$

**shows**  $\Theta ; \Phi ; B ; \Gamma ; \Delta \vdash s \Leftarrow \tau \implies \Theta ; B \vdash_{wf} \Gamma \wedge wfTh\ \Theta \wedge wfD\ \Theta\ B\ \Gamma\ \Delta \wedge wfT\ \Theta\ B\ \Gamma$

$\tau \wedge wfPhi\ \Theta\ \Phi$  **and**

$check-branch-s\ \Theta\ \Phi\ B\ \Gamma\ \Delta\ tid\ cons\ const\ v\ cs\ \tau \implies \Theta ; B \vdash_{wf} \Gamma \wedge wfTh\ \Theta \wedge wfD\ \Theta\ B\ \Gamma\ \Delta \wedge$

$wfT\ \Theta\ B\ \Gamma\ \tau \wedge wfPhi\ \Theta\ \Phi$

$check-branch-list\ \Theta\ \Phi\ B\ \Gamma\ \Delta\ tid\ dclist\ v\ css\ \tau \implies \Theta ; B \vdash_{wf} \Gamma \wedge wfTh\ \Theta \wedge wfD\ \Theta\ B\ \Gamma\ \Delta \wedge$

$wfT\ \Theta\ B\ \Gamma\ \tau \wedge wfPhi\ \Theta\ \Phi$

$\langle proof \rangle$

**lemma** *check-s-check-branch-s-wfS*:

**fixes**  $s::s$  **and**  $cs::branch-s$  **and**  $\Theta::\Theta$  **and**  $\Phi::\Phi$  **and**  $\Gamma::\Gamma$  **and**  $\Delta::\Delta$  **and**  $v::v$  **and**  $\tau::\tau$  **and**  $css::branch-list$

**shows**  $\Theta ; \Phi ; B ; \Gamma ; \Delta \vdash s \Leftarrow \tau \implies \Theta ; \Phi ; B ; \Gamma ; \Delta \vdash_{wf} s : b-of\ \tau$  **and**

$check-branch-s\ \Theta\ \Phi\ B\ \Gamma\ \Delta\ tid\ cons\ const\ v\ cs\ \tau \implies wfCS\ \Theta\ \Phi\ B\ \Gamma\ \Delta\ tid\ cons\ const\ cs\ (b-of\ \tau)$

$check-branch-list\ \Theta\ \Phi\ B\ \Gamma\ \Delta\ tid\ dclist\ v\ css\ \tau \implies wfCSS\ \Theta\ \Phi\ B\ \Gamma\ \Delta\ tid\ dclist\ css\ (b-of\ \tau)$

$\langle proof \rangle$

**lemma** *check-s-wf*:

**fixes**  $s::s$

**assumes**  $\Theta ; \Phi ; B ; \Gamma ; \Delta \vdash s \Leftarrow \tau$

**shows**  $\Theta ; B \vdash_{wf} \Gamma \wedge wfT\ \Theta\ B\ \Gamma\ \tau \wedge wfPhi\ \Theta\ \Phi \wedge wfTh\ \Theta \wedge wfD\ \Theta\ B\ \Gamma\ \Delta \wedge wfS\ \Theta\ \Phi\ B\ \Gamma\ \Delta\ s$

$(b-of\ \tau)$

$\langle proof \rangle$

**lemma** *check-s-flip-u1*:

**fixes**  $s::s$  **and**  $u::u$  **and**  $u'::u$

**assumes**  $\Theta ; \Phi ; \mathcal{B} ; \Gamma ; \Delta \vdash s \Leftarrow \tau$

**shows**  $\Theta ; \Phi ; \mathcal{B} ; \Gamma ; (u \leftrightarrow u') \cdot \Delta \vdash (u \leftrightarrow u') \cdot s \Leftarrow \tau$

$\langle proof \rangle$

**lemma** *check-s-flip-u2*:

**fixes**  $s::s$  **and**  $u::u$  **and**  $u'::u$

**assumes**  $\Theta ; \Phi ; B ; \Gamma ; (u \leftrightarrow u') \cdot \Delta \vdash (u \leftrightarrow u') \cdot s \Leftarrow \tau$

**shows**  $\Theta ; \Phi ; B ; \Gamma ; \Delta \vdash s \Leftarrow \tau$

*<proof>*

**lemma** *check-s-flip-u*:

**fixes**  $s::s$  **and**  $u::u$  **and**  $u'::u$

**shows**  $\Theta ; \Phi ; B ; \Gamma ; (u \leftrightarrow u') \cdot \Delta \vdash (u \leftrightarrow u') \cdot s \Leftarrow \tau = (\Theta ; \Phi ; B ; \Gamma ; \Delta \vdash s \Leftarrow \tau)$

*<proof>*

**lemma** *check-s-abs-u*:

**fixes**  $s::s$  **and**  $s'::s$  **and**  $u::u$  **and**  $u'::u$  **and**  $\tau'::\tau$

**assumes**  $[[atom\ u]]lst.\ s = [[atom\ u']]lst.\ s'$  **and**  $atom\ u \# \Delta$  **and**  $atom\ u' \# \Delta$

**and**  $\Theta ; B ; \Gamma \vdash_{wf} \tau'$

**and**  $\Theta ; \Phi ; B ; \Gamma ; (u, \tau') \#_{\Delta} \Delta \vdash s \Leftarrow \tau$

**shows**  $\Theta ; \Phi ; B ; \Gamma ; (u', \tau') \#_{\Delta} \Delta \vdash s' \Leftarrow \tau$

*<proof>*

## 12.7 Additional Elimination and Intros

### 12.7.1 Values

**nominal-function** *b-for* ::  $opp \Rightarrow b$  **where**

*b-for Plus* = *B-int*

| *b-for LEq* = *B-bool* | *b-for Eq* = *B-bool*

*<proof>*

**nominal-termination** (*eqvt*) *<proof>*

**lemma** *infer-v-pair2I*:

**fixes**  $v_1::v$  **and**  $v_2::v$

**assumes**  $\Theta ; \mathcal{B} ; \Gamma \vdash v_1 \Rightarrow \tau_1$  **and**  $\Theta ; \mathcal{B} ; \Gamma \vdash v_2 \Rightarrow \tau_2$

**shows**  $\exists \tau. \Theta ; \mathcal{B} ; \Gamma \vdash V\text{-pair}\ v_1\ v_2 \Rightarrow \tau \wedge b\text{-of}\ \tau = B\text{-pair}\ (b\text{-of}\ \tau_1)\ (b\text{-of}\ \tau_2)$

*<proof>*

**lemma** *infer-v-pair2I-zbc*:

**fixes**  $v_1::v$  **and**  $v_2::v$

**assumes**  $\Theta ; \mathcal{B} ; \Gamma \vdash v_1 \Rightarrow \tau_1$  **and**  $\Theta ; \mathcal{B} ; \Gamma \vdash v_2 \Rightarrow \tau_2$

**shows**  $\exists z \tau. \Theta ; \mathcal{B} ; \Gamma \vdash V\text{-pair}\ v_1\ v_2 \Rightarrow \tau \wedge \tau = (\{\!| z : B\text{-pair}\ (b\text{-of}\ \tau_1)\ (b\text{-of}\ \tau_2) \mid C\text{-eq}\ (CE\text{-val}\ (V\text{-var}\ z))\ (CE\text{-val}\ (V\text{-pair}\ v_1\ v_2)) \}\!|) \wedge atom\ z \# (v_1, v_2) \wedge atom\ z \# \Gamma$

*<proof>*

**lemma** *infer-v-pair2E*:

**assumes**  $\Theta ; \mathcal{B} ; \Gamma \vdash V\text{-pair}\ v_1\ v_2 \Rightarrow \tau$

**shows**  $\exists \tau_1\ \tau_2\ z. \Theta ; \mathcal{B} ; \Gamma \vdash v_1 \Rightarrow \tau_1 \wedge \Theta ; \mathcal{B} ; \Gamma \vdash v_2 \Rightarrow \tau_2 \wedge$

$\tau = (\{\!| z : B\text{-pair}\ (b\text{-of}\ \tau_1)\ (b\text{-of}\ \tau_2) \mid C\text{-eq}\ (CE\text{-val}\ (V\text{-var}\ z))\ (CE\text{-val}\ (V\text{-pair}\ v_1\ v_2)) \}\!|) \wedge atom\ z \# (v_1, v_2)$

*<proof>*

## 12.7.2 Expressions

**lemma** *infer-e-app2E*:

**fixes**  $\Phi::\Phi$  **and**  $\Theta::\Theta$   
**assumes**  $\Theta ; \Phi ; \mathcal{B} ; \Gamma ; \Delta \vdash AE\text{-app } f v \Rightarrow \tau$   
**shows**  $\exists x b c s' \tau'. \text{ wfD } \Theta \mathcal{B} \Gamma \Delta \wedge \text{Some } (AF\text{-fundef } f (AF\text{-fun-typ-none } (AF\text{-fun-typ } x b c \tau' s')))$   
 $= \text{lookup-fun } \Phi f \wedge \Theta \vdash_{wf} \Phi \wedge$   
 $\Theta ; \mathcal{B} ; \Gamma \vdash v \Leftarrow \{ \! \! \! \{ x : b \mid c \} \! \! \! \} \wedge \tau = \tau'[x::=v]_{\tau v} \wedge \text{atom } x \# (\Theta, \Phi, \mathcal{B}, \Gamma, \Delta, v, \tau)$   
 $\langle \text{proof} \rangle$

**lemma** *infer-e-appP2E*:

**fixes**  $\Phi::\Phi$  **and**  $\Theta::\Theta$   
**assumes**  $\Theta ; \Phi ; \mathcal{B} ; \Gamma ; \Delta \vdash AE\text{-appP } f b v \Rightarrow \tau$   
**shows**  $\exists bv x ba c s' \tau'. \text{ wfD } \Theta \mathcal{B} \Gamma \Delta \wedge \text{Some } (AF\text{-fundef } f (AF\text{-fun-typ-some } bv (AF\text{-fun-typ } x ba c \tau' s')))$   
 $= \text{lookup-fun } \Phi f \wedge \Theta \vdash_{wf} \Phi \wedge \Theta ; \mathcal{B} \vdash_{wf} b \wedge$   
 $(\Theta ; \mathcal{B} ; \Gamma \vdash v \Leftarrow \{ \! \! \! \{ x : ba[bv::=b]_{bb} \mid c[bv::=b]_{cb} \} \! \! \! \}) \wedge (\tau = \tau'[bv::=b]_{\tau b}[x::=v]_{\tau v}) \wedge \text{atom } x \# \Gamma \wedge$   
 $\text{atom } bv \# v$   
 $\langle \text{proof} \rangle$

## 12.8 Weakening

Lemmas showing that typing judgements hold when a context is extended

**lemma** *subtype-weakening*:

**fixes**  $\Gamma':\Gamma$   
**assumes**  $\Theta ; \mathcal{B} ; \Gamma \vdash \tau 1 \lesssim \tau 2$  **and**  $\text{toSet } \Gamma \subseteq \text{toSet } \Gamma'$  **and**  $\Theta ; \mathcal{B} \vdash_{wf} \Gamma'$   
**shows**  $\Theta ; \mathcal{B} ; \Gamma' \vdash \tau 1 \lesssim \tau 2$   
 $\langle \text{proof} \rangle$

**method** *many-rules uses*  $\text{add} = ( (\text{rule+}), ((\text{simp add: add})+)?)$

**lemma** *infer-v-g-weakening*:

**fixes**  $e::e$  **and**  $\Gamma':\Gamma$  **and**  $v::v$   
**assumes**  $\Theta ; \mathcal{B} ; \Gamma \vdash v \Rightarrow \tau$  **and**  $\text{toSet } \Gamma \subseteq \text{toSet } \Gamma'$  **and**  $\Theta ; \mathcal{B} \vdash_{wf} \Gamma'$   
**shows**  $\Theta ; \mathcal{B} ; \Gamma' \vdash v \Rightarrow \tau$   
 $\langle \text{proof} \rangle$

**lemma** *check-v-g-weakening*:

**fixes**  $e::e$  **and**  $\Gamma':\Gamma$   
**assumes**  $\Theta ; \mathcal{B} ; \Gamma \vdash v \Leftarrow \tau$  **and**  $\text{toSet } \Gamma \subseteq \text{toSet } \Gamma'$  **and**  $\Theta ; \mathcal{B} \vdash_{wf} \Gamma'$   
**shows**  $\Theta ; \mathcal{B} ; \Gamma' \vdash v \Leftarrow \tau$   
 $\langle \text{proof} \rangle$

**lemma** *infer-e-g-weakening*:

**fixes**  $e::e$  **and**  $\Gamma':\Gamma$   
**assumes**  $\Theta ; \Phi ; \mathcal{B} ; \Gamma ; \Delta \vdash e \Rightarrow \tau$  **and**  $\text{toSet } \Gamma \subseteq \text{toSet } \Gamma'$  **and**  $\Theta ; \mathcal{B} \vdash_{wf} \Gamma'$   
**shows**  $\Theta ; \Phi ; \mathcal{B} ; \Gamma' ; \Delta \vdash e \Rightarrow \tau$   
 $\langle \text{proof} \rangle$

Special cases proved explicitly, other cases at the end with method  $+$

**lemma** *infer-e-d-weakening*:

**fixes**  $e::e$

**assumes**  $\Theta ; \Phi ; \mathcal{B} ; \Gamma ; \Delta \vdash e \Rightarrow \tau$  **and**  $setD \Delta \subseteq setD \Delta'$  **and**  $wfD \Theta \mathcal{B} \Gamma \Delta'$   
**shows**  $\Theta ; \Phi ; \mathcal{B} ; \Gamma ; \Delta' \vdash e \Rightarrow \tau$   
 $\langle proof \rangle$

**lemma** *wfG-x-fresh-in-v-simple*:

**fixes**  $x::x$  **and**  $v::v$   
**assumes**  $\Theta ; \mathcal{B} ; \Gamma \vdash v \Rightarrow \tau$  **and**  $atom\ x \# \Gamma$   
**shows**  $atom\ x \# v$   
 $\langle proof \rangle$

**lemma** *check-s-g-weakening*:

**fixes**  $v::v$  **and**  $s::s$  **and**  $cs::branch-s$  **and**  $x::x$  **and**  $c::c$  **and**  $b::b$  **and**  $\Gamma'::\Gamma$  **and**  $\Theta::\Theta$  **and**  $css::branch-list$   
**shows**  $check-s \Theta \Phi \mathcal{B} \Gamma \Delta\ s\ t \Longrightarrow toSet \Gamma \subseteq toSet \Gamma' \Longrightarrow \Theta ; \mathcal{B} \vdash_{wf} \Gamma' \Longrightarrow check-s \Theta \Phi \mathcal{B} \Gamma' \Delta$   
 $s\ t$  **and**  
 $check-branch-s \Theta \Phi \mathcal{B} \Gamma \Delta\ tid\ cons\ const\ v\ cs\ t \Longrightarrow toSet \Gamma \subseteq toSet \Gamma' \Longrightarrow \Theta ; \mathcal{B} \vdash_{wf} \Gamma' \Longrightarrow$   
 $check-branch-s \Theta \Phi \mathcal{B} \Gamma' \Delta\ tid\ cons\ const\ v\ cs\ t$  **and**  
 $check-branch-list \Theta \Phi \mathcal{B} \Gamma \Delta\ tid\ dclist\ v\ css\ t \Longrightarrow toSet \Gamma \subseteq toSet \Gamma' \Longrightarrow \Theta ; \mathcal{B} \vdash_{wf} \Gamma' \Longrightarrow$   
 $check-branch-list \Theta \Phi \mathcal{B} \Gamma' \Delta\ tid\ dclist\ v\ css\ t$   
 $\langle proof \rangle$

**lemma** *wfG-xa-fresh-in-v*:

**fixes**  $c::c$  **and**  $\Gamma::\Gamma$  **and**  $G::\Gamma$  **and**  $v::v$  **and**  $xa::x$   
**assumes**  $\Theta ; \mathcal{B} ; \Gamma \vdash v \Rightarrow \tau$  **and**  $G=(\Gamma'@ (x, b, c[z::=V-var\ x]_v) \#_{\Gamma} \Gamma)$  **and**  $atom\ xa \# G$  **and**  $\Theta ; \mathcal{B} \vdash_{wf} G$   
**shows**  $atom\ xa \# v$   
 $\langle proof \rangle$

**lemma** *fresh-z-subst-g*:

**fixes**  $G::\Gamma$   
**assumes**  $atom\ z' \# (x, v)$  **and**  $\langle atom\ z' \# G \rangle$   
**shows**  $atom\ z' \# G[x::=v]_{\Gamma v}$   
 $\langle proof \rangle$

**lemma** *wfG-xa-fresh-in-subst-v*:

**fixes**  $c::c$  **and**  $v::v$  **and**  $x::x$  **and**  $\Gamma::\Gamma$  **and**  $G::\Gamma$  **and**  $xa::x$   
**assumes**  $\Theta ; \mathcal{B} ; \Gamma \vdash v \Rightarrow \tau$  **and**  $G=(\Gamma'@ (x, b, c[z::=V-var\ x]_v) \#_{\Gamma} \Gamma)$  **and**  $atom\ xa \# G$  **and**  $\Theta ; \mathcal{B} \vdash_{wf} G$   
**shows**  $atom\ xa \# (subst-gv\ G\ x\ v)$   
 $\langle proof \rangle$

## 12.8.1 Weakening Immutable Variable Context

**declare**  $check-s-check-branch-s-check-branch-list.intros[simp]$

**declare**  $check-s-check-branch-s-check-branch-list.intros[intro]$

**lemma** *check-s-d-weakening*:

**fixes**  $s::s$  **and**  $v::v$  **and**  $cs::branch-s$  **and**  $css::branch-list$   
**shows**  $\Theta ; \Phi ; \mathcal{B} ; \Gamma ; \Delta \vdash s \Leftarrow \tau \Longrightarrow setD \Delta \subseteq setD \Delta' \Longrightarrow wfD \Theta \mathcal{B} \Gamma \Delta' \Longrightarrow \Theta ; \Phi ; \mathcal{B} ; \Gamma ; \Delta' \vdash s \Leftarrow \tau$  **and**  
 $check-branch-s \Theta \Phi \mathcal{B} \Gamma \Delta\ tid\ cons\ const\ v\ cs\ \tau \Longrightarrow setD \Delta \subseteq setD \Delta' \Longrightarrow wfD \Theta \mathcal{B} \Gamma \Delta' \Longrightarrow$   
 $check-branch-s \Theta \Phi \mathcal{B} \Gamma \Delta'\ tid\ cons\ const\ v\ cs\ \tau$  **and**  
 $check-branch-list \Theta \Phi \mathcal{B} \Gamma \Delta\ tid\ dclist\ v\ css\ \tau \Longrightarrow setD \Delta \subseteq setD \Delta' \Longrightarrow wfD \Theta \mathcal{B} \Gamma \Delta' \Longrightarrow$   
 $check-branch-list \Theta \Phi \mathcal{B} \Gamma \Delta'\ tid\ dclist\ v\ css\ \tau$

$\langle proof \rangle$

**lemma** *valid-ce-eq*:

**fixes**  $v::v$  **and**  $ce2::ce$

**assumes**  $ce1 = ce2[x::=v]_{cev}$  **and**  $wfV \Theta \ \mathcal{B} \ GNil \ v \ b$  **and**  $wfCE \Theta \ \mathcal{B} \ ((x, b, TRUE) \#_{\Gamma} \ GNil)$   
 $ce2 \ b'$  **and**  $wfCE \Theta \ \mathcal{B} \ GNil \ ce1 \ b'$

**shows**  $\langle \Theta; \mathcal{B}; (x, b, ([[x]^v]^{ce} == [v]^{ce})) \#_{\Gamma} \ GNil \models ce1 == ce2 \rangle$

$\langle proof \rangle$

**lemma** *check-v-top*:

**fixes**  $v::v$

**assumes**  $\Theta; \mathcal{B}; GNil \vdash v \Leftarrow \tau$  **and**  $ce1 = ce2[z::=v]_{cev}$  **and**  $\Theta; \mathcal{B}; GNil \vdash_{wf} \{ z : b\text{-of } \tau \mid ce1 == ce2 \}$

**and**  $supp \ ce1 \subseteq supp \ \mathcal{B}$

**shows**  $\Theta; \mathcal{B}; GNil \vdash v \Leftarrow \{ z : b\text{-of } \tau \mid ce1 == ce2 \}$

$\langle proof \rangle$

**end**

**declare** *freshers*[*simp del*]

## Chapter 13

# Context Subtyping Lemmas

Lemmas allowing us to replace the type of a variable in the context with a subtype and have the judgement remain valid. Also known as narrowing.

### 13.1 Replace or exchange type of variable in a context

Because the G-context is extended by the statements like let, we will need a generalised substitution lemma for statements. For this we setup a function that replaces in G (rig) for a particular x the constraint for it. We also define a well-formedness relation for RIGs that ensures that each new constraint implies the old one

**nominal-function** *replace-in-g-many* ::  $\Gamma \Rightarrow (x*c) \text{ list} \Rightarrow \Gamma$  **where**  
*replace-in-g-many* G xcs = *List.foldr* ( $\lambda(x,c) G. G[x \mapsto c]$ ) xcs G  
*<proof>*

**nominal-termination** (*eqvt*) *<proof>*

**inductive** *replace-in-g-subtyped* ::  $\Theta \Rightarrow \mathcal{B} \Rightarrow \Gamma \Rightarrow (x*c) \text{ list} \Rightarrow \Gamma \Rightarrow \text{bool}$  ( $\langle - ; - \vdash - \rangle \rightsquigarrow -$ )  
[100,50,50] 50) **where**

*replace-in-g-subtyped-nil*:  $\Theta; \mathcal{B} \vdash G \langle [] \rangle \rightsquigarrow G$   
| *replace-in-g-subtyped-consI*:  $\llbracket$   
  *Some* (b,c') = *lookup* G x ;  
   $\Theta; \mathcal{B}; G \vdash_{wf} c$  ;  
   $\Theta; \mathcal{B}; G[x \mapsto c] \models c'$  ;  
   $\Theta; \mathcal{B} \vdash G[x \mapsto c] \langle xcs \rangle \rightsquigarrow G'; x \notin \text{fst 'set xcs} \rrbracket \implies$   
   $\Theta; \mathcal{B} \vdash G \langle (x,c)\#xcs \rangle \rightsquigarrow G'$

**equivariance** *replace-in-g-subtyped*

**nominal-inductive** *replace-in-g-subtyped* *<proof>*

**inductive-cases** *replace-in-g-subtyped-elim*[*elim!*]:

$\Theta; \mathcal{B} \vdash G \langle [] \rangle \rightsquigarrow G'$   
 $\Theta; \mathcal{B} \vdash ((x,b,c)\#_{\Gamma} G) \langle acs \rangle \rightsquigarrow ((x,b,c)\#_{\Gamma} G')$   
 $\Theta; \mathcal{B} \vdash G' \langle (x,c)\# acs \rangle \rightsquigarrow G$

**lemma** *rigs-atom-dom-eq*:

**assumes**  $\Theta; \mathcal{B} \vdash G \langle xcs \rangle \rightsquigarrow G'$   
**shows** *atom-dom* G = *atom-dom* G'  
*<proof>*

**lemma** *replace-in-g-wfG*:  
**assumes**  $\Theta; \mathcal{B} \vdash G \langle xcs \rangle \rightsquigarrow G'$  **and**  $wfG \Theta \mathcal{B} G$   
**shows**  $wfG \Theta \mathcal{B} G'$   
 $\langle proof \rangle$

**lemma** *wfD-rig-single*:  
**fixes**  $\Delta::\Delta$  **and**  $x::x$  **and**  $c::c$  **and**  $G::\Gamma$   
**assumes**  $\Theta; \mathcal{B}; G \vdash_{wf} \Delta$  **and**  $wfG \Theta \mathcal{B} (G[x \mapsto c])$   
**shows**  $\Theta; \mathcal{B}; G[x \mapsto c] \vdash_{wf} \Delta$   
 $\langle proof \rangle$

**lemma** *wfD-rig*:  
**assumes**  $\Theta; \mathcal{B} \vdash G \langle xcs \rangle \rightsquigarrow G'$  **and**  $wfD \Theta \mathcal{B} G \Delta$   
**shows**  $wfD \Theta \mathcal{B} G' \Delta$   
 $\langle proof \rangle$

**lemma** *replace-in-g-fresh*:  
**fixes**  $x::x$   
**assumes**  $\Theta; \mathcal{B} \vdash \Gamma \langle xcs \rangle \rightsquigarrow \Gamma'$  **and**  $wfG \Theta \mathcal{B} \Gamma$  **and**  $wfG \Theta \mathcal{B} \Gamma'$  **and**  $atom\ x \# \Gamma$   
**shows**  $atom\ x \# \Gamma'$   
 $\langle proof \rangle$

**lemma** *replace-in-g-fresh1*:  
**fixes**  $x::x$   
**assumes**  $\Theta; \mathcal{B} \vdash \Gamma \langle xcs \rangle \rightsquigarrow \Gamma'$  **and**  $wfG \Theta \mathcal{B} \Gamma$  **and**  $atom\ x \# \Gamma$   
**shows**  $atom\ x \# \Gamma'$   
 $\langle proof \rangle$

Wellscoping for an eXchange list

**inductive**  $wsX::\Gamma \Rightarrow (x*c)\ list \Rightarrow bool$  **where**  
 $wsX\ NilI: wsX\ G\ []$   
 $| wsX\ ConsI: [ wsX\ G\ xcs ; atom\ x \in atom\text{-}dom\ G ; x \notin fst\ 'set\ xcs ] \Longrightarrow wsX\ G\ ((x,c)\#xcs)$   
**equivariance**  $wsX$   
**nominal-inductive**  $wsX\ \langle proof \rangle$

**lemma** *wsX-if1*:  
**assumes**  $wsX\ G\ xcs$   
**shows**  $((atom\ 'fst\ 'set\ xcs) \subseteq atom\text{-}dom\ G) \wedge List.distinct\ (List.map\ fst\ xcs)$   
 $\langle proof \rangle$

**lemma** *wsX-if2*:  
**assumes**  $((atom\ 'fst\ 'set\ xcs) \subseteq atom\text{-}dom\ G) \wedge List.distinct\ (List.map\ fst\ xcs)$   
**shows**  $wsX\ G\ xcs$   
 $\langle proof \rangle$

**lemma** *wsX-iff*:  
 $wsX\ G\ xcs = (((atom\ 'fst\ 'set\ xcs) \subseteq atom\text{-}dom\ G) \wedge List.distinct\ (List.map\ fst\ xcs))$   
 $\langle proof \rangle$

**inductive-cases**  $wsX\ elims[elim!]$ :  
 $wsX\ G\ []$

$wsX\ G\ ((x,c)\#xcs)$

**lemma** *wsX-cons*:

**assumes**  $wsX\ \Gamma\ xcs$  **and**  $x \notin fst\ 'set\ xcs$   
**shows**  $wsX\ ((x, b, c1)\#_{\Gamma}\ \Gamma)\ ((x, c2)\#xcs)$   
*<proof>*

**lemma** *wsX-cons2*:

**assumes**  $wsX\ \Gamma\ xcs$  **and**  $x \notin fst\ 'set\ xcs$   
**shows**  $wsX\ ((x, b, c1)\#_{\Gamma}\ \Gamma)\ xcs$   
*<proof>*

**lemma** *wsX-cons3*:

**assumes**  $wsX\ \Gamma\ xcs$   
**shows**  $wsX\ ((x, b, c1)\#_{\Gamma}\ \Gamma)\ xcs$   
*<proof>*

**lemma** *wsX-fresh*:

**assumes**  $wsX\ G\ xcs$  **and**  $atom\ x\ \#G$  **and**  $wfG\ \Theta\ \mathcal{B}\ G$   
**shows**  $x \notin fst\ 'set\ xcs$   
*<proof>*

**lemma** *replace-in-g-dist*:

**assumes**  $x' \neq x$   
**shows**  $replace-in-g\ ((x, b, c)\#_{\Gamma}\ G)\ x'\ c'' = ((x, b, c)\#_{\Gamma}\ (replace-in-g\ G\ x'\ c''))$  *<proof>*

**lemma** *wfG-replace-inside-rig*:

**fixes**  $c''::c$   
**assumes**  $\langle \Theta; \mathcal{B} \vdash_{wf}\ G[x' \mapsto c''] \rangle$   $\langle \Theta; \mathcal{B} \vdash_{wf}\ (x, b, c)\#_{\Gamma}\ G \rangle$   
**shows**  $\Theta; \mathcal{B} \vdash_{wf}\ (x, b, c)\#_{\Gamma}\ G[x' \mapsto c'']$   
*<proof>*

**lemma** *replace-in-g-valid-weakening*:

**assumes**  $\Theta; \mathcal{B}; \Gamma[x' \mapsto c''] \models c'$  **and**  $x' \neq x$  **and**  $\Theta; \mathcal{B} \vdash_{wf}\ (x, b, c)\#_{\Gamma}\ \Gamma[x' \mapsto c'']$   
**shows**  $\Theta; \mathcal{B}; ((x, b, c)\#_{\Gamma}\ \Gamma)[x' \mapsto c''] \models c'$   
*<proof>*

**lemma** *replace-in-g-subtyped-cons*:

**assumes**  $replace-in-g-subtyped\ \Theta\ \mathcal{B}\ G\ xcs\ G'$  **and**  $wfG\ \Theta\ \mathcal{B}\ ((x,b,c)\#_{\Gamma}\ G)$   
**shows**  $x \notin fst\ 'set\ xcs \implies replace-in-g-subtyped\ \Theta\ \mathcal{B}\ ((x,b,c)\#_{\Gamma}\ G)\ xcs\ ((x,b,c)\#_{\Gamma}\ G')$   
*<proof>*

**lemma** *replace-in-g-split*:

**fixes**  $G::\Gamma$   
**assumes**  $\Gamma = replace-in-g\ \Gamma'\ x\ c$  **and**  $\Gamma' = G'@_{(x,b,c)}\#_{\Gamma}\ G$  **and**  $wfG\ \Theta\ \mathcal{B}\ \Gamma'$   
**shows**  $\Gamma = G'@_{(x,b,c)}\#_{\Gamma}\ G$   
*<proof>*

**lemma** *replace-in-g-subtyped-split0*:

**fixes**  $G::\Gamma$   
**assumes**  $replace-in-g-subtyped\ \Theta\ \mathcal{B}\ \Gamma'[(x,c)]\ \Gamma$  **and**  $\Gamma' = G'@_{(x,b,c)}\#_{\Gamma}\ G$  **and**  $wfG\ \Theta\ \mathcal{B}\ \Gamma'$   
**shows**  $\Gamma = G'@_{(x,b,c)}\#_{\Gamma}\ G$

$\langle \text{proof} \rangle$

**lemma** *replace-in-g-subtyped-split*:

**assumes** *Some*  $(b, c') = \text{lookup } G \ x \ \text{and } \Theta; \mathcal{B}; \text{replace-in-g } G \ x \ c \models c' \ \text{and } \text{wf}G \ \Theta \ \mathcal{B} \ G$

**shows**  $\exists \Gamma \Gamma'. G = \Gamma' @ (x, b, c') \#_{\Gamma} \Gamma \wedge \Theta; \mathcal{B}; \Gamma' @ (x, b, c) \#_{\Gamma} \Gamma \models c'$

$\langle \text{proof} \rangle$

## 13.2 Validity and Subtyping

**lemma** *wfC-replace-in-g*:

**fixes**  $c::c \ \text{and} \ c0::c$

**assumes**  $\Theta; \mathcal{B}; \Gamma' @ (x, b, c0') \#_{\Gamma} \Gamma \vdash_{\text{wf}} c \ \text{and} \ \Theta; \mathcal{B}; (x, b, \text{TRUE}) \#_{\Gamma} \Gamma \vdash_{\text{wf}} c0$

**shows**  $\Theta; \mathcal{B}; \Gamma' @ (x, b, c0) \#_{\Gamma} \Gamma \vdash_{\text{wf}} c$

$\langle \text{proof} \rangle$

**lemma** *ctx-subtype-valid*:

**assumes**  $\Theta; \mathcal{B}; \Gamma' @ (x, b, c0') \#_{\Gamma} \Gamma \models c \ \text{and}$

$\Theta; \mathcal{B}; \Gamma' @ (x, b, c0) \#_{\Gamma} \Gamma \models c0'$

**shows**  $\Theta; \mathcal{B}; \Gamma' @ (x, b, c0) \#_{\Gamma} \Gamma \models c$

$\langle \text{proof} \rangle$

**lemma** *ctx-subtype-subtype*:

**fixes**  $\Gamma::\Gamma$

**shows**  $\Theta; \mathcal{B}; G \vdash t1 \lesssim t2 \implies G = \Gamma' @ (x, b0, c0') \#_{\Gamma} \Gamma \implies \Theta; \mathcal{B}; \Gamma' @ (x, b0, c0) \#_{\Gamma} \Gamma \models c0' \implies \Theta; \mathcal{B}; \Gamma' @ (x, b0, c0) \#_{\Gamma} \Gamma \vdash t1 \lesssim t2$

$\langle \text{proof} \rangle$

**lemma** *ctx-subtype-subtype-rig*:

**assumes** *replace-in-g-subtyped*  $\Theta \ \mathcal{B} \ \Gamma' [(x, c0)] \ \Gamma \ \text{and} \ \Theta; \mathcal{B}; \Gamma' \vdash t1 \lesssim t2$

**shows**  $\Theta; \mathcal{B}; \Gamma \vdash t1 \lesssim t2$

$\langle \text{proof} \rangle$

We now prove versions of the *ctx-subtype* lemmas above using *replace-in-g*. First we do case where the replace is just for a single variable (indicated by suffix rig) and then the general case for multiple replacements (indicated by suffix rigs)

**lemma** *ctx-subtype-subtype-rigs*:

**assumes** *replace-in-g-subtyped*  $\Theta \ \mathcal{B} \ \Gamma' \ xcs \ \Gamma \ \text{and} \ \Theta; \mathcal{B}; \Gamma' \vdash t1 \lesssim t2$

**shows**  $\Theta; \mathcal{B}; \Gamma \vdash t1 \lesssim t2$

$\langle \text{proof} \rangle$

**lemma** *replace-in-g-inside-valid*:

**assumes** *replace-in-g-subtyped*  $\Theta \ \mathcal{B} \ \Gamma' [(x, c0)] \ \Gamma \ \text{and} \ \text{wf}G \ \Theta \ \mathcal{B} \ \Gamma'$

**shows**  $\exists b \ c0' \ G \ G'. \Gamma' = G' @ (x, b, c0') \#_{\Gamma} G \wedge \Gamma = G' @ (x, b, c0) \#_{\Gamma} G \wedge \Theta; \mathcal{B}; G' @ (x, b, c0) \#_{\Gamma} G \models c0'$

$\langle \text{proof} \rangle$

**lemma** *replace-in-g-valid*:

**assumes**  $\Theta; \mathcal{B} \vdash G \langle xcs \rangle \rightsquigarrow G' \ \text{and} \ \Theta; \mathcal{B}; G \models c$

**shows**  $\langle \Theta; \mathcal{B}; G' \models c \rangle$

$\langle \text{proof} \rangle$

## 13.3 Literals

## 13.4 Values

**lemma** *lookup-inside-unique-b[simp]*:

**assumes**  $\Theta ; B \vdash_{wf} (\Gamma' @ (x, b0, c0) \#_{\Gamma} \Gamma)$  **and**  $\Theta ; B \vdash_{wf} (\Gamma' @ (x, b0, c0') \#_{\Gamma} \Gamma)$   
**and** *Some*  $(b, c) = \text{lookup} (\Gamma' @ (x, b0, c0') \#_{\Gamma} \Gamma) y$  **and** *Some*  $(b0, c0) = \text{lookup} (\Gamma' @ ((x, b0, c0)) \#_{\Gamma} \Gamma)$   
*x* **and**  $x=y$   
**shows**  $b = b0$   
 $\langle \text{proof} \rangle$

**lemma** *ctx-subtype-v-aux*:

**fixes**  $v::v$   
**assumes**  $\Theta ; \mathcal{B}; \Gamma' @ ((x, b0, c0') \#_{\Gamma} \Gamma) \vdash v \Rightarrow t1$  **and**  $\Theta ; \mathcal{B}; \Gamma' @ (x, b0, c0) \#_{\Gamma} \Gamma \models c0'$   
**shows**  $\Theta ; \mathcal{B}; \Gamma' @ ((x, b0, c0) \#_{\Gamma} \Gamma) \vdash v \Rightarrow t1$   
 $\langle \text{proof} \rangle$

**lemma** *ctx-subtype-v*:

**fixes**  $v::v$   
**assumes**  $\Theta ; \mathcal{B}; \Gamma' @ ((x, b0, c0') \#_{\Gamma} \Gamma) \vdash v \Rightarrow t1$  **and**  $\Theta ; \mathcal{B}; \Gamma' @ (x, b0, c0) \#_{\Gamma} \Gamma \models c0'$   
**shows**  $\exists t2. \Theta ; \mathcal{B}; \Gamma' @ ((x, b0, c0) \#_{\Gamma} \Gamma) \vdash v \Rightarrow t2 \wedge \Theta ; \mathcal{B}; \Gamma' @ ((x, b0, c0) \#_{\Gamma} \Gamma) \vdash t2 \lesssim t1$   
 $\langle \text{proof} \rangle$

**lemma** *ctx-subtype-v-eq*:

**fixes**  $v::v$   
**assumes**  
 $\Theta ; \mathcal{B}; \Gamma' @ ((x, b0, c0') \#_{\Gamma} \Gamma) \vdash v \Rightarrow t1$  **and**  
 $\Theta ; \mathcal{B}; \Gamma' @ (x, b0, c0) \#_{\Gamma} \Gamma \models c0'$   
**shows**  $\Theta ; \mathcal{B}; \Gamma' @ ((x, b0, c0) \#_{\Gamma} \Gamma) \vdash v \Rightarrow t1$   
 $\langle \text{proof} \rangle$

**lemma** *ctx-subtype-check-v-eq*:

**assumes**  $\Theta ; \mathcal{B}; \Gamma' @ ((x, b0, c0') \#_{\Gamma} \Gamma) \vdash v \Leftarrow t1$  **and**  $\Theta ; \mathcal{B}; \Gamma' @ (x, b0, c0) \#_{\Gamma} \Gamma \models c0'$   
**shows**  $\Theta ; \mathcal{B}; \Gamma' @ ((x, b0, c0) \#_{\Gamma} \Gamma) \vdash v \Leftarrow t1$   
 $\langle \text{proof} \rangle$

Basically the same as *ctx-subtype-v-eq* but in a different form

**lemma** *ctx-subtype-v-rig-eq*:

**fixes**  $v::v$   
**assumes** *replace-in-g-subtyped*  $\Theta \mathcal{B} \Gamma' [(x, c0)] \Gamma$  **and**  
 $\Theta ; \mathcal{B}; \Gamma' \vdash v \Rightarrow t1$   
**shows**  $\Theta ; \mathcal{B}; \Gamma \vdash v \Rightarrow t1$   
 $\langle \text{proof} \rangle$

**lemma** *ctx-subtype-v-rigs-eq*:

**fixes**  $v::v$   
**assumes** *replace-in-g-subtyped*  $\Theta \mathcal{B} \Gamma' xcs \Gamma$  **and**  
 $\Theta ; \mathcal{B}; \Gamma' \vdash v \Rightarrow t1$   
**shows**  $\Theta ; \mathcal{B}; \Gamma \vdash v \Rightarrow t1$   
 $\langle \text{proof} \rangle$

**lemma** *ctx-subtype-check-v-rigs-eq*:

**assumes** *replace-in-g-subtyped*  $\Theta \mathcal{B} \Gamma' \text{ xcs } \Gamma$  **and**  
 $\Theta ; \mathcal{B} ; \Gamma' \vdash v \Leftarrow t1$   
**shows**  $\Theta ; \mathcal{B} ; \Gamma \vdash v \Leftarrow t1$   
 $\langle \text{proof} \rangle$

## 13.5 Expressions

**lemma** *valid-wfC*:  
**fixes**  $c0::c$   
**assumes**  $\Theta ; \mathcal{B} ; \Gamma' @ (x, b0, c0) \#_{\Gamma} \Gamma \models c0'$   
**shows**  $\Theta ; \mathcal{B} ; (x, b0, TRUE) \#_{\Gamma} \Gamma \vdash_{wf} c0$   
 $\langle \text{proof} \rangle$

**lemma** *ctx-subtype-e-eq*:  
**fixes**  $G::\Gamma$   
**assumes**  
 $\Theta ; \Phi ; \mathcal{B} ; G ; \Delta \vdash e \Rightarrow t1$  **and**  $G = \Gamma' @ ((x, b0, c0') \#_{\Gamma} \Gamma)$   
 $\Theta ; \mathcal{B} ; \Gamma' @ (x, b0, c0) \#_{\Gamma} \Gamma \models c0'$   
**shows**  $\Theta ; \Phi ; \mathcal{B} ; \Gamma' @ ((x, b0, c0) \#_{\Gamma} \Gamma) ; \Delta \vdash e \Rightarrow t1$   
 $\langle \text{proof} \rangle$

**lemma** *ctx-subtype-e-rig-eq*:  
**assumes** *replace-in-g-subtyped*  $\Theta \mathcal{B} \Gamma' [(x, c0)] \Gamma$  **and**  
 $\Theta ; \Phi ; \mathcal{B} ; \Gamma' ; \Delta \vdash e \Rightarrow t1$   
**shows**  $\Theta ; \Phi ; \mathcal{B} ; \Gamma ; \Delta \vdash e \Rightarrow t1$   
 $\langle \text{proof} \rangle$

**lemma** *ctx-subtype-e-rigs-eq*:  
**assumes** *replace-in-g-subtyped*  $\Theta \mathcal{B} \Gamma' \text{ xcs } \Gamma$  **and**  
 $\Theta ; \Phi ; \mathcal{B} ; \Gamma' ; \Delta \vdash e \Rightarrow t1$   
**shows**  $\Theta ; \Phi ; \mathcal{B} ; \Gamma ; \Delta \vdash e \Rightarrow t1$   
 $\langle \text{proof} \rangle$

## 13.6 Statements

**lemma** *ctx-subtype-s-rigs*:  
**fixes**  $c0::c$  **and**  $s::s$  **and**  $G'::\Gamma$  **and**  $\text{xcs} :: (x*c)$  *list* **and**  $\text{css}::\text{branch-list}$   
**shows**  
 $\text{check-s } \Theta \Phi \mathcal{B} G \Delta \ s \ t1 \Longrightarrow \text{wsX } G \ \text{xcs} \Longrightarrow \text{replace-in-g-subtyped } \Theta \mathcal{B} G \ \text{xcs } G' \Longrightarrow \text{check-s } \Theta$   
 $\Phi \mathcal{B} G' \Delta \ s \ t1$  **and**  
 $\text{check-branch-s } \Theta \Phi \mathcal{B} G \Delta \ \text{tid } \text{cons } \text{const } v \ \text{cs } \ t1 \Longrightarrow \text{wsX } G \ \text{xcs} \Longrightarrow \text{replace-in-g-subtyped } \Theta \mathcal{B}$   
 $G \ \text{xcs } G' \Longrightarrow \text{check-branch-s } \Theta \Phi \mathcal{B} G' \Delta \ \text{tid } \text{cons } \text{const } v \ \text{cs } \ t1$   
 $\text{check-branch-list } \Theta \Phi \mathcal{B} G \Delta \ \text{tid } \ \text{dclist } v \ \text{css } \ t1 \Longrightarrow \text{wsX } G \ \text{xcs} \Longrightarrow \text{replace-in-g-subtyped } \Theta \mathcal{B} G$   
 $\text{xcs } G' \Longrightarrow \text{check-branch-list } \Theta \Phi \mathcal{B} G' \Delta \ \text{tid } \ \text{dclist } v \ \text{css } \ t1$   
 $\langle \text{proof} \rangle$

**lemma** *replace-in-g-subtyped-empty*:  
**assumes**  $\text{wfG } \Theta \mathcal{B} (\Gamma' @ (x, b, c[z::=V\text{-var } x]_{cv}) \#_{\Gamma} \Gamma)$   
**shows**  $\text{replace-in-g-subtyped } \Theta \mathcal{B} (\text{replace-in-g } (\Gamma' @ (x, b, c[z::=V\text{-var } x]_{cv}) \#_{\Gamma} \Gamma) \ x \ (c'[z'::=V\text{-var } x]_{cv})) \ \square \ (\Gamma' @ (x, b, c'[z'::=V\text{-var } x]_{cv}) \#_{\Gamma} \Gamma)$   
 $\langle \text{proof} \rangle$

**lemma** *ctx-subtype-s*:  
**fixes**  $s::s$   
**assumes**  $\Theta ; \Phi ; \mathcal{B} ; \Gamma'@((x,b,c[z::=V\text{-var } x]_{cv})\#_{\Gamma}\Gamma) ; \Delta \vdash s \Leftarrow \tau$  **and**  
 $\Theta ; \mathcal{B} ; \Gamma \vdash \{ z' : b \mid c' \} \lesssim \{ z : b \mid c \}$  **and**  
*atom*  $x \# (z, z', c, c')$   
**shows**  $\Theta ; \Phi ; \mathcal{B} ; \Gamma'@(x,b,c'[z'::=V\text{-var } x]_{cv})\#_{\Gamma}\Gamma ; \Delta \vdash s \Leftarrow \tau$   
*<proof>*  
**end**

## Chapter 14

# Immutable Variable Substitution Lemmas

Lemmas that show that types are preserved, in some way, under immutable variable substitution

### 14.1 Proof Methods

**method** *subst-mth* = (*metis subst-g-inside infer-e-wf infer-v-wf infer-v-wf*)

**method** *subst-tuple-mth* **uses** *add* = (  
 (*unfold fresh-prodN*), (*simp add: add*) +,  
 (*rule,metis fresh-z-subst-g add fresh-Pair*),  
 (*metis fresh-subst-dv add fresh-Pair*) )

### 14.2 Prelude

**lemma** *subst-top-eq*:

$\{ z : b \mid TRUE \} = \{ z : b \mid TRUE \} [x ::= v]_{\tau v}$   
*<proof>*

**lemma** *wfD-subst*:

**fixes**  $\tau_1 :: \tau$  **and**  $v :: v$  **and**  $\Delta :: \Delta$  **and**  $\Theta :: \Theta$  **and**  $\Gamma :: \Gamma$   
**assumes**  $\Theta ; \mathcal{B} ; \Gamma \vdash v \Rightarrow \tau_1$  **and** *wfD*  $\Theta \ \mathcal{B} \ (\Gamma' @ ((x, b_1, c_0 [z_0 ::= [x]^v]_{cv}) \#_{\Gamma} \Gamma)) \ \Delta$  **and** *b-of*  $\tau_1 = b_1$   
**shows**  $\Theta ; \mathcal{B} ; \Gamma [x ::= v]_{\Gamma v} @ \Gamma \vdash_{wf} \Delta [x ::= v]_{\Delta v}$   
*<proof>*

**lemma** *subst-v-c-of*:

**assumes** *atom*  $xa \ \# \ (v, x)$   
**shows** *c-of*  $t [x ::= v]_{\tau v} \ xa = (c\text{-of } t \ xa) [x ::= v]_{cv}$   
*<proof>*

### 14.3 Context

**lemma** *subst-lookup*:

**assumes** *Some*  $(b, c) = \text{lookup} \ (\Gamma' @ ((x, b_1, c_1) \#_{\Gamma} \Gamma)) \ y$  **and**  $x \neq y$  **and** *wfG*  $\Theta \ \mathcal{B} \ (\Gamma' @ ((x, b_1, c_1) \#_{\Gamma} \Gamma))$   
**shows**  $\exists d. \text{Some } (b, d) = \text{lookup} \ ((\Gamma [x ::= v]_{\Gamma v}) @ \Gamma) \ y$

$\langle \text{proof} \rangle$

## 14.4 Validity

**lemma** *subst-self-valid*:

**fixes**  $v::v$   
**assumes**  $\Theta ; \mathcal{B} ; G \vdash v \Rightarrow \{ \!| z : b \mid c \! \}$  **and**  $\text{atom } z \# v$   
**shows**  $\Theta ; \mathcal{B} ; G \models c[z::=v]_{cv}$   
 $\langle \text{proof} \rangle$

**lemma** *subst-valid-simple*:

**fixes**  $v::v$   
**assumes**  $\Theta ; \mathcal{B} ; G \vdash v \Rightarrow \{ \!| z0 : b \mid c0 \! \}$  **and**  
 $\text{atom } z0 \# c$  **and**  $\text{atom } z0 \# v$   
 $\Theta ; \mathcal{B} ; (z0, b, c0) \#_{\Gamma} G \models c[z::=V\text{-var } z0]_{cv}$   
**shows**  $\Theta ; \mathcal{B} ; G \models c[z::=v]_{cv}$   
 $\langle \text{proof} \rangle$

**lemma** *wfI-subst1*:

**assumes**  $\text{wfI } \Theta (G'[x::=v]_{\Gamma v} @ G) i$  **and**  $\text{wfG } \Theta \mathcal{B} (G' @ (x, b, c[z::=[x]^v]_{cv}) \#_{\Gamma} G)$  **and**  $\text{eval-v } i v$   
 $sv$  **and**  $\text{wfRCV } \Theta sv b$   
**shows**  $\text{wfI } \Theta (G' @ (x, b, c[z::=[x]^v]_{cv}) \#_{\Gamma} G) (i(x \mapsto sv))$   
 $\langle \text{proof} \rangle$

**lemma** *subst-valid*:

**fixes**  $v::v$  **and**  $c'::c$  **and**  $\Gamma ::\Gamma$   
**assumes**  $\Theta ; \mathcal{B} ; \Gamma \models c[z::=v]_{cv}$  **and**  $\Theta ; \mathcal{B} ; \Gamma \vdash_{wf} v : b$  **and**  
 $\Theta ; \mathcal{B} \vdash_{wf} \Gamma$  **and**  $\text{atom } x \# c$  **and**  $\text{atom } x \# \Gamma$  **and**  
 $\Theta ; \mathcal{B} \vdash_{wf} (\Gamma @ (x, b, c[z::=[x]^v]_{cv})) \#_{\Gamma} \Gamma$  **and**  
 $\Theta ; \mathcal{B} ; \Gamma @ (x, b, c[z::=[x]^v]_{cv}) \#_{\Gamma} \Gamma \models c'$  (**is**  $\Theta ; \mathcal{B} ; ?G \models c'$ )  
**shows**  $\Theta ; \mathcal{B} ; \Gamma[x::=v]_{\Gamma v} @ \Gamma \models c'[x::=v]_{cv}$   
 $\langle \text{proof} \rangle$

**lemma** *subst-valid-infer-v*:

**fixes**  $v::v$  **and**  $c'::c$   
**assumes**  $\Theta ; \mathcal{B} ; G \vdash v \Rightarrow \{ \!| z0 : b \mid c0 \! \}$  **and**  $\text{atom } x \# c$  **and**  $\text{atom } x \# G$  **and**  $\text{wfG } \Theta \mathcal{B}$   
 $(G' @ (x, b, c[z::=[x]^v]_{cv})) \#_{\Gamma} G$  **and**  $\text{atom } z0 \# v$   
 $\Theta ; \mathcal{B} ; (z0, b, c0) \#_{\Gamma} G \models c[z::=V\text{-var } z0]_{cv}$  **and**  $\text{atom } z0 \# c$  **and**  
 $\Theta ; \mathcal{B} ; G' @ (x, b, c[z::=[x]^v]_{cv}) \#_{\Gamma} G \models c'$  (**is**  $\Theta ; \mathcal{B} ; ?G \models c'$ )  
**shows**  $\Theta ; \mathcal{B} ; G'[x::=v]_{\Gamma v} @ G \models c'[x::=v]_{cv}$   
 $\langle \text{proof} \rangle$

## 14.5 Subtyping

**lemma** *subst-subtype*:

**fixes**  $v::v$   
**assumes**  $\Theta ; \mathcal{B} ; \Gamma \vdash v \Rightarrow (\{ \!| z0 : b \mid c0 \! \})$  **and**  
 $\Theta ; \mathcal{B} ; \Gamma \vdash (\{ \!| z0 : b \mid c0 \! \}) \lesssim (\{ \!| z : b \mid c \! \})$  **and**  
 $\Theta ; \mathcal{B} ; \Gamma @ ((x, b, c[z::=[x]^v]_{cv}) \#_{\Gamma} \Gamma) \vdash (\{ \!| z1 : b1 \mid c1 \! \}) \lesssim (\{ \!| z2 : b1 \mid c2 \! \})$  (**is**  $\Theta ; \mathcal{B} ; ?G1 \vdash ?t1 \lesssim ?t2$ ) **and**  
 $\text{atom } z \# (x, v) \wedge \text{atom } z0 \# (c, x, v, z, \Gamma) \wedge \text{atom } z1 \# (x, v) \wedge \text{atom } z2 \# (x, v)$  **and**  $\text{wsV } \Theta \mathcal{B} \Gamma v$

**shows**  $\Theta; \mathcal{B}; \Gamma[x::=v]_{\Gamma v} @ \Gamma \vdash \{ z1 : b1 \mid c1 \} [x::=v]_{\tau v} \lesssim \{ z2 : b1 \mid c2 \} [x::=v]_{\tau v}$   
 ⟨proof⟩

**lemma** *subst-subtype-tau*:

**fixes**  $v::v$

**assumes**  $\Theta; \mathcal{B}; \Gamma \vdash v \Rightarrow \tau$  **and**

$\Theta; \mathcal{B}; \Gamma \vdash \tau \lesssim (\{ z : b \mid c \})$

$\Theta; \mathcal{B}; \Gamma' @ ((x, b, c[z::=[x]^v]_{cv}) \# \Gamma) \vdash \tau 1 \lesssim \tau 2$  **and**

$atom\ z \# (x, v)$

**shows**  $\Theta; \mathcal{B}; \Gamma[x::=v]_{\Gamma v} @ \Gamma \vdash \tau 1 [x::=v]_{\tau v} \lesssim \tau 2 [x::=v]_{\tau v}$

⟨proof⟩

**lemma** *subtype-if1*:

**fixes**  $v::v$

**assumes**  $P; \mathcal{B}; \Gamma \vdash t1 \lesssim t2$  **and**  $wfV\ P\ \mathcal{B}\ \Gamma\ v$  (*base-for-lit l*) **and**

$atom\ z1 \# v$  **and**  $atom\ z2 \# v$  **and**  $atom\ z1 \# t1$  **and**  $atom\ z2 \# t2$  **and**  $atom\ z1 \# \Gamma$  **and**  $atom\ z2 \# \Gamma$

**shows**  $P; \mathcal{B}; \Gamma \vdash \{ z1 : b\text{-of } t1 \mid CE\text{-val } v == CE\text{-val } (V\text{-lit } l) \text{ IMP } (c\text{-of } t1\ z1) \} \lesssim \{ z2 : b\text{-of } t2 \mid CE\text{-val } v == CE\text{-val } (V\text{-lit } l) \text{ IMP } (c\text{-of } t2\ z2) \}$

⟨proof⟩

## 14.6 Values

**lemma** *subst-infer-aux*:

**fixes**  $\tau_1::\tau$  **and**  $v'::v$

**assumes**  $\Theta; \mathcal{B}; \Gamma \vdash v'[x::=v]_{vv} \Rightarrow \tau_1$  **and**  $\Theta; \mathcal{B}; \Gamma' \vdash v' \Rightarrow \tau_2$  **and**  $b\text{-of } \tau_1 = b\text{-of } \tau_2$

**shows**  $\tau_1 = (\tau_2[x::=v]_{\tau v})$

⟨proof⟩

**lemma** *subst-t-b-eq*:

**fixes**  $x::x$  **and**  $v::v$

**shows**  $b\text{-of } (\tau[x::=v]_{\tau v}) = b\text{-of } \tau$

⟨proof⟩

**lemma** *fresh-g-fresh-v*:

**fixes**  $x::x$

**assumes**  $atom\ x \# \Gamma$  **and**  $wfV\ \Theta\ \mathcal{B}\ \Gamma\ v\ b$

**shows**  $atom\ x \# v$

⟨proof⟩

**lemma** *infer-v-fresh-g-fresh-v*:

**fixes**  $x::x$  **and**  $\Gamma::\Gamma$  **and**  $v::v$

**assumes**  $atom\ x \# \Gamma' @ \Gamma$  **and**  $\Theta; \mathcal{B}; \Gamma \vdash v \Rightarrow \tau$

**shows**  $atom\ x \# v$

⟨proof⟩

**lemma** *infer-v-fresh-g-fresh-xv*:

**fixes**  $xa::x$  **and**  $v::v$  **and**  $\Gamma::\Gamma$

**assumes**  $atom\ xa \# \Gamma' @ ((x, b, c[z::=[x]^v]_{cv}) \# \Gamma)$  **and**  $\Theta; \mathcal{B}; \Gamma \vdash v \Rightarrow \tau$

**shows**  $atom\ xa \# (x, v)$

⟨proof⟩

**lemma** *wfG-subst-infer-v*:

**fixes**  $v::v$

**assumes**  $\Theta ; \mathcal{B} \vdash_{wf} \Gamma' @ (x, b, c0[z0::=[x]^v]_{cv}) \#_{\Gamma} \Gamma$  **and**  $\Theta ; \mathcal{B} ; \Gamma \vdash v \Rightarrow \tau$  **and** *b-of*  $\tau = b$

**shows**  $\Theta ; \mathcal{B} \vdash_{wf} \Gamma'[x::=v]_{\Gamma v} @ \Gamma$

*<proof>*

**lemma** *fresh-subst-gv-inside*:

**fixes**  $\Gamma::\Gamma$

**assumes** *atom*  $z \# \Gamma' @ (x, b_1, c0[z0::=[x]^v]_{cv}) \#_{\Gamma} \Gamma$  **and** *atom*  $z \# v$

**shows** *atom*  $z \# \Gamma'[x::=v]_{\Gamma v} @ \Gamma$

*<proof>*

**lemma** *subst-t*:

**fixes**  $x::x$  **and**  $b::b$  **and**  $xa::x$

**assumes** *atom*  $z \# x$  **and** *atom*  $z \# v$

**shows**  $(\{ z : b \mid [[z]^v]^{ce} == [v'[x::=v]_{vv}]^{ce} \}) = (\{ z : b \mid [[z]^v]^{ce} == [v]^{ce} \} \{ [x::=v]_{\tau v} \})$

*<proof>*

**lemma** *infer-l-fresh*:

**assumes**  $\vdash l \Rightarrow \tau$

**shows** *atom*  $x \# \tau$

*<proof>*

**lemma** *subst-infer-v*:

**fixes**  $v::v$  **and**  $v'::v$

**assumes**  $\Theta ; \mathcal{B} ; \Gamma' @ ((x, b_1, c0[z0::=[x]^v]_{cv}) \#_{\Gamma} \Gamma) \vdash v' \Rightarrow \tau_2$  **and**

$\Theta ; \mathcal{B} ; \Gamma \vdash v \Rightarrow \tau_1$  **and**

$\Theta ; \mathcal{B} ; \Gamma \vdash \tau_1 \lesssim (\{ z0 : b_1 \mid c0 \})$  **and** *atom*  $z0 \# (x, v)$

**shows**  $\Theta ; \mathcal{B} ; (\Gamma'[x::=v]_{\Gamma v}) @ \Gamma \vdash v'[x::=v]_{vv} \Rightarrow \tau_2[x::=v]_{\tau v}$

*<proof>*

**lemma** *subst-infer-check-v*:

**fixes**  $v::v$  **and**  $v'::v$

**assumes**  $\Theta ; \mathcal{B} ; \Gamma \vdash v \Rightarrow \tau_1$  **and**

*check-v*  $\Theta \mathcal{B} (\Gamma' @ ((x, b_1, c0[z0::=[x]^v]_{cv}) \#_{\Gamma} \Gamma)) \vdash v' \tau_2$  **and**

$\Theta ; \mathcal{B} ; \Gamma \vdash \tau_1 \lesssim (\{ z0 : b_1 \mid c0 \})$  **and** *atom*  $z0 \# (x, v)$

**shows** *check-v*  $\Theta \mathcal{B} ((\Gamma'[x::=v]_{\Gamma v}) @ \Gamma) \vdash (v'[x::=v]_{vv}) (\tau_2[x::=v]_{\tau v})$

*<proof>*

**lemma** *type-veq-subst[simp]*:

**assumes** *atom*  $z \# (x, v)$

**shows**  $\{ z : b \mid CE\text{-val} (V\text{-var } z) == CE\text{-val } v' \{ [x::=v]_{\tau v} \} = \{ z : b \mid CE\text{-val} (V\text{-var } z) == CE\text{-val } v'[x::=v]_{vv} \}$

*<proof>*

**lemma** *subst-infer-v-form*:

**fixes**  $v::v$  **and**  $v'::v$  **and**  $\Gamma::\Gamma$

**assumes**  $\Theta ; \mathcal{B} ; \Gamma \vdash v \Rightarrow \tau_1$  **and**

$\Theta ; \mathcal{B} ; \Gamma' @ ((x, b_1, c0[z0::=[x]^v]_{cv}) \#_{\Gamma} \Gamma) \vdash v' \Rightarrow \tau_2$  **and** *b=* *b-of*  $\tau_2$

$\Theta ; \mathcal{B} ; \Gamma \vdash \tau_1 \lesssim (\{ z0 : b_1 \mid c0 \})$  **and** *atom*  $z0 \# (x, v)$  **and** *atom*  $z0 \# (x, v, v', \Gamma' @ ((x, b_1, c0[z0::=[x]^v]_{cv}) \#_{\Gamma} \Gamma))$

)

**shows**  $\Theta ; \mathcal{B} ; \Gamma'[x::=v]_{\Gamma v} @ \Gamma \vdash v'[x::=v]_{vv} \Rightarrow \{ z0' : b \mid CE\text{-val} (V\text{-var } z0') == CE\text{-val}$

$v'[x::=v]_{vv} \} \rangle$   
 $\langle \text{proof} \rangle$

## 14.7 Expressions

For operator, fst and snd cases, we use elimination to get one or more values, apply the substitution lemma for values. The types always have the same form and are equal under substitution. For function application, the subst value is a subtype of the value which is a subtype of the argument. The return of the function is the same under substitution.

Observe a similar pattern for each case

**lemma** *subst-infer-e*:

**fixes**  $v::v$  **and**  $e::e$  **and**  $\Gamma::\Gamma$

**assumes**

$\Theta ; \Phi ; \mathcal{B} ; G ; \Delta \vdash e \Rightarrow \tau_2$  **and**  $G = (\Gamma' @ ((x, b_1, \text{subst-cv } c0 \ z0 \ (V\text{-var } x)) \#_{\Gamma} \Gamma))$

$\Theta ; \mathcal{B} ; \Gamma \vdash v \Rightarrow \tau_1$  **and**

$\Theta ; \mathcal{B} ; \Gamma \vdash \tau_1 \lesssim \{ \{ z0 : b_1 \mid c0 \} \}$  **and** *atom*  $z0 \# (x, v)$

**shows**  $\Theta ; \Phi ; \mathcal{B} ; ((\Gamma'[x::=v]_{\Gamma v}) @ \Gamma) ; (\Delta[x::=v]_{\Delta v}) \vdash (\text{subst-ev } e \ x \ v) \Rightarrow \tau_2[x::=v]_{\tau v}$

$\langle \text{proof} \rangle$

**lemma** *infer-e-uniqueness*:

**assumes**  $\Theta ; \Phi ; \mathcal{B} ; GNil ; \Delta \vdash e_1 \Rightarrow \tau_1$  **and**  $\Theta ; \Phi ; \mathcal{B} ; GNil ; \Delta \vdash e_1 \Rightarrow \tau_2$

**shows**  $\tau_1 = \tau_2$

$\langle \text{proof} \rangle$

## 14.8 Statements

**lemma** *subst-infer-check-v1*:

**fixes**  $v::v$  **and**  $v'::v$  **and**  $\Gamma::\Gamma$

**assumes**  $\Gamma = \Gamma_1 @ ((x, b_1, c0 [z0::=[x]^v]_{cv}) \#_{\Gamma} \Gamma_2)$  **and**

$\Theta ; \mathcal{B} ; \Gamma_2 \vdash v \Rightarrow \tau_1$  **and**

$\Theta ; \mathcal{B} ; \Gamma \vdash v' \Leftarrow \tau_2$  **and**

$\Theta ; \mathcal{B} ; \Gamma_2 \vdash \tau_1 \lesssim \{ \{ z0 : b_1 \mid c0 \} \}$  **and** *atom*  $z0 \# (x, v)$

**shows**  $\Theta ; \mathcal{B} ; \Gamma[x::=v]_{\Gamma v} \vdash v'[x::=v]_{vv} \Leftarrow \tau_2[x::=v]_{\tau v}$

$\langle \text{proof} \rangle$

**lemma** *infer-v-c-valid*:

**assumes**  $\Theta ; \mathcal{B} ; \Gamma \vdash v \Rightarrow \tau$  **and**  $\Theta ; \mathcal{B} ; \Gamma \vdash \tau \lesssim \{ \{ z : b \mid c \} \}$

**shows**  $\langle \Theta ; \mathcal{B} ; \Gamma \models c[z::=v]_{cv} \rangle$

$\langle \text{proof} \rangle$

Substitution Lemma for Statements

**lemma** *subst-infer-check-s*:

**fixes**  $v::v$  **and**  $s::s$  **and**  $cs::\text{branch-s}$  **and**  $x::x$  **and**  $c::c$  **and**  $b::b$  **and**

$\Gamma_1::\Gamma$  **and**  $\Gamma_2::\Gamma$  **and**  $css::\text{branch-list}$

**assumes**  $\Theta ; \mathcal{B} ; \Gamma_1 \vdash v \Rightarrow \tau$  **and**  $\Theta ; \mathcal{B} ; \Gamma_1 \vdash \tau \lesssim \{ \{ z : b \mid c \} \}$  **and**

*atom*  $z \# (x, v)$

**shows**  $\Theta ; \Phi ; \mathcal{B} ; \Gamma ; \Delta \vdash s \Leftarrow \tau' \implies$

$\Gamma = (\Gamma_2 @ ((x, b, c[z::=[x]^v]_{cv}) \#_{\Gamma} \Gamma_1)) \implies$

$\Theta ; \Phi ; \mathcal{B} ; \Gamma[x::=v]_{\Gamma v} ; \Delta[x::=v]_{\Delta v} \vdash s[x::=v]_{sv} \Leftarrow \tau'[x::=v]_{\tau v}$

**and**

$\Theta ; \Phi ; \mathcal{B} ; \Gamma ; \Delta ; tid ; cons ; const ; v' \vdash cs \Leftarrow \tau' \implies$   
 $\Gamma = (\Gamma_2 @ ((x, b, c[z::=[x]^v]_{cv}) \#_{\Gamma} \Gamma_1)) \implies$   
 $\Theta ; \Phi ; \mathcal{B} ; \Gamma[x::=v]_{\Gamma v} ; \Delta[x::=v]_{\Delta v} ;$   
 $tid ; cons ; const ; v'[x::=v]_{v v} \vdash cs[x::=v]_{sv} \Leftarrow \tau'[x::=v]_{\tau v}$

**and**

$\Theta ; \Phi ; \mathcal{B} ; \Gamma ; \Delta ; tid ; dclist ; v' \vdash css \Leftarrow \tau' \implies$   
 $\Gamma = (\Gamma_2 @ ((x, b, c[z::=[x]^v]_{cv}) \#_{\Gamma} \Gamma_1)) \implies$   
 $\Theta ; \Phi ; \mathcal{B} ; \Gamma[x::=v]_{\Gamma v} ; \Delta[x::=v]_{\Delta v} ; tid ; dclist ; v'[x::=v]_{v v} \vdash$   
 $subst-branchlv\ css\ x\ v \Leftarrow \tau'[x::=v]_{\tau v}$

$\langle proof \rangle$

**lemma** *subst-check-check-s*:

**fixes**  $v::v$  **and**  $s::s$  **and**  $cs::branch-s$  **and**  $x::x$  **and**  $c::c$  **and**  $b::b$  **and**  $\Gamma_1::\Gamma$  **and**  $\Gamma_2::\Gamma$   
**assumes**  $\Theta ; \mathcal{B} ; \Gamma_1 \vdash v \Leftarrow \{ z : b \mid c \}$  **and**  $atom\ z \# (x, v)$   
**and**  $check-s\ \Theta\ \Phi\ \mathcal{B}\ \Gamma\ \Delta\ s\ \tau'$  **and**  $\Gamma = (\Gamma_2 @ ((x, b, c[z::=[x]^v]_{cv}) \#_{\Gamma} \Gamma_1))$   
**shows**  $check-s\ \Theta\ \Phi\ \mathcal{B}\ (subst-gv\ \Gamma\ x\ v)\ \Delta[x::=v]_{\Delta v}\ (s[x::=v]_{sv})\ (subst-tv\ \tau'\ x\ v)$

$\langle proof \rangle$

If a statement checks against a type  $\tau$  then it checks against a supertype of  $\tau$

**lemma** *check-s-supertype*:

**fixes**  $v::v$  **and**  $s::s$  **and**  $cs::branch-s$  **and**  $x::x$  **and**  $c::c$  **and**  $b::b$  **and**  $\Gamma::\Gamma$  **and**  $\Gamma'::\Gamma$  **and**  $css::branch-list$   
**shows**  $check-s\ \Theta\ \Phi\ \mathcal{B}\ G\ \Delta\ s\ t1 \implies \Theta ; \mathcal{B} ; G \vdash t1 \lesssim t2 \implies check-s\ \Theta\ \Phi\ \mathcal{B}\ G\ \Delta\ s\ t2$  **and**  
 $check-branch-s\ \Theta\ \Phi\ \mathcal{B}\ G\ \Delta\ tid\ cons\ const\ v\ cs\ t1 \implies \Theta ; \mathcal{B} ; G \vdash t1 \lesssim t2 \implies check-branch-s\ \Theta\ \Phi$   
 $\mathcal{B}\ G\ \Delta\ tid\ cons\ const\ v\ cs\ t2$  **and**  
 $check-branch-list\ \Theta\ \Phi\ \mathcal{B}\ G\ \Delta\ tid\ dclist\ v\ css\ t1 \implies \Theta ; \mathcal{B} ; G \vdash t1 \lesssim t2 \implies check-branch-list\ \Theta\ \Phi$   
 $\mathcal{B}\ G\ \Delta\ tid\ dclist\ v\ css\ t2$

$\langle proof \rangle$

**lemma** *subtype-let*:

**fixes**  $s'::s$  **and**  $cs::branch-s$  **and**  $css::branch-list$  **and**  $v::v$   
**shows**  $\Theta ; \Phi ; \mathcal{B} ; GNil ; \Delta \vdash AS-let\ x\ e_1\ s \Leftarrow \tau \implies \Theta ; \Phi ; \mathcal{B} ; GNil ; \Delta \vdash e_1 \Rightarrow \tau_1 \implies$   
 $\Theta ; \Phi ; \mathcal{B} ; GNil ; \Delta \vdash e_2 \Rightarrow \tau_2 \implies \Theta ; \mathcal{B} ; GNil \vdash \tau_2 \lesssim \tau_1 \implies \Theta ; \Phi ; \mathcal{B} ; GNil ; \Delta \vdash AS-let$   
 $x\ e_2\ s \Leftarrow \tau$  **and**  
 $check-branch-s\ \Theta\ \Phi\ \{||\}\ GNil\ \Delta\ tid\ dc\ const\ v\ cs\ \tau \implies True$  **and**  
 $check-branch-list\ \Theta\ \Phi\ \{||\}\ \Gamma\ \Delta\ tid\ dclist\ v\ css\ \tau \implies True$

$\langle proof \rangle$

**end**

## Chapter 15

# Basic Type Variable Substitution Lemmas

Lemmas that show that types are preserved, in some way, under basic type variable substitution

**lemma** *subst-vv-subst-bb-commute:*

**fixes**  $bv::bv$  **and**  $b::b$  **and**  $x::x$  **and**  $v::v$

**assumes**  $atom\ bv \ \# \ v$

**shows**  $(v'[x::=v]_{vv})[bv::=b]_{vb} = (v'[bv::=b]_{vb})[x::=v]_{vv}$

$\langle proof \rangle$

**lemma** *subst-cev-subst-bb-commute:*

**fixes**  $bv::bv$  **and**  $b::b$  **and**  $x::x$  **and**  $v::v$

**assumes**  $atom\ bv \ \# \ v$

**shows**  $(ce[x::=v]_v)[bv::=b]_{ceb} = (ce[bv::=b]_{ceb})[x::=v]_v$

$\langle proof \rangle$

**lemma** *subst-cv-subst-bb-commute:*

**fixes**  $bv::bv$  **and**  $b::b$  **and**  $x::x$  **and**  $v::v$

**assumes**  $atom\ bv \ \# \ v$

**shows**  $c[x::=v]_{cv}[bv::=b]_{cb} = (c[bv::=b]_{cb})[x::=v]_{cv}$

$\langle proof \rangle$

**lemma** *subst-b-c-of:*

$(c\text{-of } \tau\ z)[bv::=b]_{cb} = c\text{-of } (\tau[bv::=b]_{\tau b})\ z$

$\langle proof \rangle$

**lemma** *subst-b-b-of:*

$(b\text{-of } \tau)[bv::=b]_{bb} = b\text{-of } (\tau[bv::=b]_{\tau b})$

$\langle proof \rangle$

**lemma** *subst-b-if:*

$\{ z : b\text{-of } \tau[bv::=b]_{\tau b} \mid CE\text{-val } (v[bv::=b]_{vb}) == CE\text{-val } (V\text{-lit } ll) \ IMP\ c\text{-of } \tau[bv::=b]_{\tau b}\ z \} = \{ z : b\text{-of } \tau \mid CE\text{-val } (v) == CE\text{-val } (V\text{-lit } ll) \ IMP\ c\text{-of } \tau\ z \} [bv::=b]_{\tau b}$

$\langle proof \rangle$

**lemma** *subst-b-top-eq:*

$\{ z : B\text{-unit} \mid TRUE \} [bv::=b]_{\tau b} = \{ z : B\text{-unit} \mid TRUE \}$  **and**  $\{ z : B\text{-bool} \mid TRUE \} [bv::=b]_{\tau b} =$

$\{\{ z : B\text{-bool} \mid \text{TRUE} \}\}$  **and**  $\{\{ z : B\text{-id tid} \mid \text{TRUE} \}\} = \{\{ z : B\text{-id tid} \mid \text{TRUE} \}\}[bv::=b]_{\tau b}$   
 $\langle \text{proof} \rangle$

**lemmas** *subst-b-eq* = *subst-b-c-of subst-b-b-of subst-b-if subst-b-top-eq*

**lemma** *subst-cx-subst-bb-commute*[*simp*]:

**fixes**  $bv::bv$  **and**  $b::b$  **and**  $x::x$  **and**  $v'::c$

**shows**  $(v'[x::=V\text{-var } y]_{cv})[bv::=b]_{cb} = (v'[bv::=b]_{cb})[x::=V\text{-var } y]_{cv}$

$\langle \text{proof} \rangle$

**lemma** *subst-b-infer-b*:

**fixes**  $l::l$  **and**  $b::b$

**assumes**  $\vdash l \Rightarrow \tau$  **and**  $\Theta$  ;  $\{\{\}\} \vdash_{wf} b$  **and**  $B = \{|bv|\}$

**shows**  $\vdash l \Rightarrow (\tau[bv::=b]_{\tau b})$

$\langle \text{proof} \rangle$

**lemma** *subst-b-subtype*:

**fixes**  $s::s$  **and**  $b'::b$

**assumes**  $\Theta$  ;  $\{|bv|\}$  ;  $\Gamma \vdash \tau 1 \lesssim \tau 2$  **and**  $\Theta$  ;  $\{\{\}\} \vdash_{wf} b'$  **and**  $B = \{|bv|\}$

**shows**  $\Theta$  ;  $\{\{\}\}$  ;  $\Gamma[bv::=b]_{\Gamma b} \vdash \tau 1[bv::=b]_{\tau b} \lesssim \tau 2[bv::=b]_{\tau b}$

$\langle \text{proof} \rangle$

**lemma** *b-of-subst-bv*:

$(b\text{-of } \tau)[x::=v]_{bb} = b\text{-of } (\tau[x::=v]_{\tau b})$

$\langle \text{proof} \rangle$

**lemma** *subst-b-infer-v*:

**fixes**  $v::v$  **and**  $b::b$

**assumes**  $\Theta$  ;  $B$  ;  $G \vdash v \Rightarrow \tau$  **and**  $\Theta$  ;  $\{\{\}\} \vdash_{wf} b$  **and**  $B = \{|bv|\}$

**shows**  $\Theta$  ;  $\{\{\}\}$  ;  $G[bv::=b]_{\Gamma b} \vdash v[bv::=b]_{vb} \Rightarrow (\tau[bv::=b]_{\tau b})$

$\langle \text{proof} \rangle$

**lemma** *subst-b-check-v*:

**fixes**  $v::v$  **and**  $b::b$

**assumes**  $\Theta$  ;  $B$  ;  $G \vdash v \Leftarrow \tau$  **and**  $\Theta$  ;  $\{\{\}\} \vdash_{wf} b$  **and**  $B = \{|bv|\}$

**shows**  $\Theta$  ;  $\{\{\}\}$  ;  $G[bv::=b]_{\Gamma b} \vdash v[bv::=b]_{vb} \Leftarrow (\tau[bv::=b]_{\tau b})$

$\langle \text{proof} \rangle$

**lemma** *subst-vv-subst-vb-switch*:

**shows**  $(v'[bv::=b]_{vb})[x::=v[bv::=b]_{vb}]_{vv} = v'[x::=v]_{vv}[bv::=b]_{vb}$

$\langle \text{proof} \rangle$

**lemma** *subst-cev-subst-vb-switch*:

**shows**  $(ce[bv::=b]_{ceb})[x::=v[bv::=b]_{vb}]_{cev} = (ce[x::=v]_{cev})[bv::=b]_{ceb}$

$\langle \text{proof} \rangle$

**lemma** *subst-cv-subst-vb-switch*:

**shows**  $(c[bv::=b]_{cb})[x::=v[bv::=b]_{vb}]_{cv} = c[x::=v]_{cv}[bv::=b]_{cb}$

$\langle \text{proof} \rangle$

**lemma** *subst-tv-subst-vb-switch*:

**shows**  $(\tau[bv::=b]_{\tau b})[x::=v[bv::=b]_{vb}]_{\tau v} = \tau[x::=v]_{\tau v}[bv::=b]_{\tau b}$

*<proof>*

**lemma** *subst-tb-triple:*

**assumes** *atom bv # τ'*

**shows**  $\tau'[bv'::=b'[bv::=b]_{bb}]_{\tau b}[x'::=v'[bv::=b]_{vb}]_{\tau v} = \tau'[bv'::=b']_{\tau b}[x'::=v']_{\tau v}[bv::=b]_{\tau b}$

*<proof>*

**lemma** *subst-b-infer-e:*

**fixes** *s::s and b::b*

**assumes**  $\Theta ; \Phi ; B ; G ; D \vdash e \Rightarrow \tau$  **and**  $\Theta ; \{|\}\vdash_{wf} b$  **and**  $B = \{|bv|\}$

**shows**  $\Theta ; \Phi ; \{|\}\vdash G[bv::=b]_{\Gamma b} ; D[bv::=b]_{\Delta b} \vdash (e[bv::=b]_{eb}) \Rightarrow (\tau[bv::=b]_{\tau b})$

*<proof>*

This is needed for preservation. When we apply a function "f [b] v" we need to substitute into the body of the function f replacing type-variable with b

**lemma** *subst-b-c-of-forget:*

**assumes** *atom bv # const*

**shows**  $(c\text{-of } const\ x)[bv::=b]_{cb} = c\text{-of } const\ x$

*<proof>*

**lemma** *subst-b-check-s:*

**fixes** *s::s and b::b and cs::branch-s and css::branch-list and v::v and τ::τ*

**assumes**  $\Theta ; \{|\}\vdash_{wf} b$  **and**  $B = \{|bv|\}$

**shows**  $\Theta ; \Phi ; B ; G ; D \vdash s \Leftarrow \tau \Longrightarrow \Theta ; \Phi ; \{|\}\vdash G[bv::=b]_{\Gamma b} ; D[bv::=b]_{\Delta b} \vdash (s[bv::=b]_{sb}) \Leftarrow (\tau[bv::=b]_{\tau b})$  **and**

$\Theta ; \Phi ; B ; G ; D ; tid ; cons ; const ; v \vdash cs \Leftarrow \tau \Longrightarrow \Theta ; \Phi ; \{|\}\vdash G[bv::=b]_{\Gamma b} ; D[bv::=b]_{\Delta b} ; tid ; cons ; const ; v[bv::=b]_{vb} \vdash (subst\ branchb\ cs\ bv\ b) \Leftarrow (\tau[bv::=b]_{\tau b})$  **and**

$\Theta ; \Phi ; B ; G ; D ; tid ; dclist ; v \vdash css \Leftarrow \tau \Longrightarrow \Theta ; \Phi ; \{|\}\vdash G[bv::=b]_{\Gamma b} ; D[bv::=b]_{\Delta b} ; tid ; dclist ; v[bv::=b]_{vb} \vdash (subst\ branchlb\ css\ bv\ b) \Leftarrow (\tau[bv::=b]_{\tau b})$

*<proof>*

**end**

**method** *supp-calc = (metis (mono-tags, opaque-lifting) pure-supp c.supp e.supp v.supp supp-l-empty opp.supp sup-bot.right-neutral supp-at-base)*

**declare** *infer-e.intros[simp]*

**declare** *infer-e.intros[intro]*

# Chapter 16

## Safety

Lemmas about the operational semantics leading up to progress and preservation and then safety.

### 16.1 Store Lemmas

**abbreviation** *delta-ext* ( $\langle - \sqsubseteq - \rangle$ ) **where**  
*delta-ext*  $\Delta \Delta' \equiv (\text{setD } \Delta \subseteq \text{setD } \Delta')$

**nominal-function** *dc-of*  $:: \text{branch-s} \Rightarrow \text{string}$  **where**  
*dc-of* (*AS-branch* *dc*  $-$ ) = *dc*  
*\langle proof \rangle*

**nominal-termination** (*eqvt*) *\langle proof \rangle*

**lemma** *delta-sim-fresh*:  
**assumes**  $\Theta \vdash \delta \sim \Delta$  **and** *atom*  $u \# \delta$   
**shows** *atom*  $u \# \Delta$   
*\langle proof \rangle*

**lemma** *delta-sim-v*:  
**fixes**  $\Delta :: \Delta$   
**assumes**  $\Theta \vdash \delta \sim \Delta$  **and**  $(u, v) \in \text{set } \delta$  **and**  $(u, \tau) \in \text{setD } \Delta$  **and**  $\Theta ; \{\|\}; \text{GNil} \vdash_{wf} \Delta$   
**shows**  $\Theta ; \{\|\}; \text{GNil} \vdash v \Leftarrow \tau$   
*\langle proof \rangle*

**lemma** *delta-sim-delta-lookup*:  
**assumes**  $\Theta \vdash \delta \sim \Delta$  **and**  $(u, \{z : b \mid c\}) \in \text{setD } \Delta$   
**shows**  $\exists v. (u, v) \in \text{set } \delta$   
*\langle proof \rangle*

**lemma** *update-d-stable*:  
*fst* ' *set*  $\delta = \text{fst}$  ' *set* (*update-d*  $\delta$   $u$   $v$ )  
*\langle proof \rangle*

**lemma** *update-d-sim*:  
**fixes**  $\Delta :: \Delta$   
**assumes**  $\Theta \vdash \delta \sim \Delta$  **and**  $\Theta ; \{\|\}; \text{GNil} \vdash v \Leftarrow \tau$  **and**  $(u, \tau) \in \text{setD } \Delta$  **and**  $\Theta ; \{\|\}; \text{GNil} \vdash_{wf} \Delta$

**shows**  $\Theta \vdash (\text{update-d } \delta \ u \ v) \sim \Delta$   
 ⟨proof⟩

## 16.2 Preservation

Types are preserved under reduction step. Broken down into lemmas about different operations

### 16.2.1 Function Application

**lemma** *check-s-x-fresh*:

**fixes**  $x::x$  **and**  $s::s$

**assumes**  $\Theta ; \Phi ; B ; GNil ; D \vdash s \Leftarrow \tau$

**shows**  $\text{atom } x \# s \wedge \text{atom } x \# \tau \wedge \text{atom } x \# D$

⟨proof⟩

**lemma** *check-funtyp-subst-b*:

**fixes**  $b'::b$

**assumes**  $\text{check-funtyp } \Theta \ \Phi \ \{\{bv\}\} \ (AF\text{-fun-ty} \ x \ b \ c \ \tau \ s)$  **and**  $\langle \Theta ; \{\{\}\} \vdash_{wf} \ b' \rangle$

**shows**  $\text{check-funtyp } \Theta \ \Phi \ \{\{\}\} \ (AF\text{-fun-ty} \ x \ b[bv::=b]_{bb} \ (c[bv::=b]_{cb}) \ \tau[bv::=b]_{\tau b} \ s[bv::=b]_{sb})$

⟨proof⟩

**lemma** *funtyp-simple-check*:

**fixes**  $s::s$  **and**  $\Delta::\Delta$  **and**  $\tau::\tau$  **and**  $v::v$

**assumes**  $\text{check-funtyp } \Theta \ \Phi \ (\{\{\}\}::bv \ fset) \ (AF\text{-fun-ty} \ x \ b \ c \ \tau \ s)$  **and**

$\Theta ; \{\{\}\} ; GNil \vdash v \Leftarrow \{ \! \! \! x : b \mid c \}$

**shows**  $\Theta ; \Phi ; \{\{\}\} ; GNil ; DNil \vdash s[x::=v]_{sv} \Leftarrow \tau[x::=v]_{\tau v}$

⟨proof⟩

**lemma** *funtypq-simple-check*:

**fixes**  $s::s$  **and**  $\Delta::\Delta$  **and**  $\tau::\tau$  **and**  $v::v$

**assumes**  $\text{check-funtypq } \Theta \ \Phi \ (AF\text{-fun-ty-none } (AF\text{-fun-ty } \ x \ b \ c \ t \ s))$  **and**

$\Theta ; \{\{\}\} ; GNil \vdash v \Leftarrow \{ \! \! \! x : b \mid c \}$

**shows**  $\Theta ; \Phi ; \{\{\}\} ; GNil ; DNil \vdash s[x::=v]_{sv} \Leftarrow t[x::=v]_{\tau v}$

⟨proof⟩

**lemma** *funtyp-poly-eq-iff-equalities*:

**assumes**  $[[\text{atom } bv]] \text{lst. } AF\text{-fun-ty} \ x' \ b'' \ c' \ t' \ s' = [[\text{atom } bv]] \text{lst. } AF\text{-fun-ty} \ x \ b \ c \ t \ s$

**shows**  $\{ \! \! \! x' : b'[bv::=b]_{bb} \mid c'[bv::=b]_{cb} \} = \{ \! \! \! x : b[bv::=b]_{bb} \mid c[bv::=b]_{cb} \} \wedge$

$s'[bv::=b]_{sb}[x'::=v]_{s'v} = s[bv::=b]_{sb}[x::=v]_{sv} \wedge t'[bv::=b]_{\tau b}[x'::=v]_{\tau v} = t[bv::=b]_{\tau b}[x::=v]_{\tau v}$

⟨proof⟩

**lemma** *funtypq-poly-check*:

**fixes**  $s::s$  **and**  $\Delta::\Delta$  **and**  $\tau::\tau$  **and**  $v::v$  **and**  $b'::b$

**assumes**  $\text{check-funtypq } \Theta \ \Phi \ (AF\text{-fun-ty-some } bv \ (AF\text{-fun-ty } \ x \ b \ c \ t \ s))$  **and**

$\Theta ; \{\{\}\} ; GNil \vdash v \Leftarrow \{ \! \! \! x : b[bv::=b]_{bb} \mid c[bv::=b]_{cb} \}$  **and**

$\Theta ; \{\{\}\} \vdash_{wf} \ b'$

**shows**  $\Theta ; \Phi ; \{\{\}\} ; GNil ; DNil \vdash s[bv::=b]_{sb}[x::=v]_{sv} \Leftarrow t[bv::=b]_{\tau b}[x::=v]_{\tau v}$

⟨proof⟩

**lemma** *fundef-simple-check*:

**fixes**  $s::s$  **and**  $\Delta::\Delta$  **and**  $\tau::\tau$  **and**  $v::v$

**assumes** *check-fundef*  $\Theta \Phi$  (*AF-fundef*  $f$  (*AF-fun-typ-none* (*AF-fun-typ*  $x b c t s$ ))) **and**  
 $\Theta ; \{\|\}; GNil \vdash v \Leftarrow \{ x : b \mid c \}$  **and**  $\Theta ; \{\|\}; GNil \vdash_{wf} \Delta$   
**shows**  $\Theta ; \Phi ; \{\|\}; GNil ; \Delta \vdash s[x::=v]_{sv} \Leftarrow t[x::=v]_{\tau v}$   
 $\langle proof \rangle$

**lemma** *fundef-poly-check*:

**fixes**  $s::s$  **and**  $\Delta::\Delta$  **and**  $\tau::\tau$  **and**  $v::v$  **and**  $b'::b$   
**assumes** *check-fundef*  $\Theta \Phi$  (*AF-fundef*  $f$  (*AF-fun-typ-some*  $bv$  (*AF-fun-typ*  $x b c t s$ ))) **and**  
 $\Theta ; \{\|\}; GNil \vdash v \Leftarrow \{ x : b[bv::=b]_{bb} \mid c[bv::=b]_{cb} \}$  **and**  $\Theta ; \{\|\}; GNil \vdash_{wf} \Delta$  **and**  $\Theta ; \{\|\}$   
 $\vdash_{wf} b'$   
**shows**  $\Theta ; \Phi ; \{\|\}; GNil ; \Delta \vdash s[bv::=b]_{sb}[x::=v]_{sv} \Leftarrow t[bv::=b]_{\tau b}[x::=v]_{\tau v}$   
 $\langle proof \rangle$

**lemma** *preservation-app*:

**assumes**  
*Some* (*AF-fundef*  $f$  (*AF-fun-typ-none* (*AF-fun-typ*  $x1 b1 c1 \tau1' s1'$ ))) = *lookup-fun*  $\Phi f$  **and** ( $\forall fd \in set$   
 $\Phi$ . *check-fundef*  $\Theta \Phi fd$ )  
**shows**  $\Theta ; \Phi ; B ; G ; \Delta \vdash ss \Leftarrow \tau \implies B = \{\|\} \implies G = GNil \implies ss = LET x = (AE-app f v)$   
 $IN s \implies$   
 $\Theta ; \Phi ; \{\|\}; GNil ; \Delta \vdash LET x : (\tau1'[x1::=v]_{\tau v}) = (s1'[x1::=v]_{sv}) IN s \Leftarrow \tau$  **and**  
*check-branch-s*  $\Theta \Phi \mathcal{B} GNil \Delta tid dc const v cs \tau \implies True$  **and**  
*check-branch-list*  $\Theta \Phi \mathcal{B} \Gamma \Delta tid dclist v css \tau \implies True$   
 $\langle proof \rangle$

**lemma** *fresh-subst-v-subst-b*:

**fixes**  $x2::x$  **and**  $tm::a::\{has-subst-v, has-subst-b\}$  **and**  $x::x$   
**assumes**  $supp tm \subseteq \{ atom bv, atom x \}$  **and**  $atom x2 \# v$   
**shows**  $atom x2 \# tm[bv::=b]_b[x::=v]_v$   
 $\langle proof \rangle$

**lemma** *preservation-poly-app*:

**assumes**  
*Some* (*AF-fundef*  $f$  (*AF-fun-typ-some*  $bv1$  (*AF-fun-typ*  $x1 b1 c1 \tau1' s1'$ ))) = *lookup-fun*  $\Phi f$  **and**  
( $\forall fd \in set$   $\Phi$ . *check-fundef*  $\Theta \Phi fd$ )  
**shows**  $\Theta ; \Phi ; B ; G ; \Delta \vdash ss \Leftarrow \tau \implies B = \{\|\} \implies G = GNil \implies ss = LET x = (AE-appP f b'$   
 $v) IN s \implies \Theta ; \{\|\} \vdash_{wf} b' \implies$   
 $\Theta ; \Phi ; \{\|\}; GNil ; \Delta \vdash LET x : (\tau1'[bv1::=b]_{\tau b}[x1::=v]_{\tau v}) = (s1'[bv1::=b]_{sb}[x1::=v]_{sv}) IN$   
 $s \Leftarrow \tau$  **and**  
*check-branch-s*  $\Theta \Phi \mathcal{B} GNil \Delta tid dc const v cs \tau \implies True$  **and**  
*check-branch-list*  $\Theta \Phi \mathcal{B} \Gamma \Delta tid dclist v css \tau \implies True$   
 $\langle proof \rangle$

**lemma** *check-s-plus*:

**assumes**  $\Theta ; \Phi ; \{\|\}; GNil ; \Delta \vdash LET x = (AE-op Plus (V-lit (L-num n1)) (V-lit (L-num n2))) IN$   
 $s' \Leftarrow \tau$   
**shows**  $\Theta ; \Phi ; \{\|\}; GNil ; \Delta \vdash LET x = (AE-val (V-lit (L-num (n1+n2)))) IN s' \Leftarrow \tau$   
 $\langle proof \rangle$

**lemma** *check-s-leq*:

**assumes**  $\Theta ; \Phi ; \{\|\}; GNil ; \Delta \vdash LET x = (AE-op LEq (V-lit (L-num n1)) (V-lit (L-num n2)))$   
 $IN s' \Leftarrow \tau$   
**shows**  $\Theta ; \Phi ; \{\|\}; GNil ; \Delta \vdash LET x = (AE-val (V-lit (if (n1 \leq n2) then L-true else L-false))) IN$

$s' \Leftarrow \tau$   
 $\langle \text{proof} \rangle$

**lemma** *check-s-eq*:

**assumes**  $\Theta; \Phi; \{\|\}; GNil; \Delta \vdash LET\ x = (AE\text{-op}\ Eq\ (V\text{-lit}\ (n1))\ (V\text{-lit}\ (n2)))\ IN\ s' \Leftarrow \tau$   
**shows**  $\Theta; \Phi; \{\|\}; GNil; \Delta \vdash LET\ x = (AE\text{-val}\ (V\text{-lit}\ (if\ (n1 = n2)\ then\ L\text{-true}\ else\ L\text{-false})))\ IN\ s' \Leftarrow \tau$   
 $\langle \text{proof} \rangle$

## 16.2.2 Operators

**lemma** *preservation-plus*:

**assumes**  $\Theta; \Phi; \Delta \vdash \langle \delta, LET\ x = (AE\text{-op}\ Plus\ (V\text{-lit}\ (L\text{-num}\ n1))\ (V\text{-lit}\ (L\text{-num}\ n2)))\ IN\ s' \rangle \Leftarrow \tau$   
**shows**  $\Theta; \Phi; \Delta \vdash \langle \delta, LET\ x = (AE\text{-val}\ (V\text{-lit}\ (L\text{-num}\ (n1+n2))))\ IN\ s' \rangle \Leftarrow \tau$   
 $\langle \text{proof} \rangle$

**lemma** *preservation-leq*:

**assumes**  $\Theta; \Phi; \Delta \vdash \langle \delta, AS\text{-let}\ x\ (AE\text{-op}\ LEq\ (V\text{-lit}\ (L\text{-num}\ n1))\ (V\text{-lit}\ (L\text{-num}\ n2)))\ s' \rangle \Leftarrow \tau$   
**shows**  $\Theta; \Phi; \Delta \vdash \langle \delta, AS\text{-let}\ x\ (AE\text{-val}\ (V\text{-lit}\ (((if\ (n1 \leq n2)\ then\ L\text{-true}\ else\ L\text{-false})))))\ s' \rangle \Leftarrow \tau$   
 $\langle \text{proof} \rangle$

**lemma** *preservation-eq*:

**assumes**  $\Theta; \Phi; \Delta \vdash \langle \delta, AS\text{-let}\ x\ (AE\text{-op}\ Eq\ (V\text{-lit}\ (n1))\ (V\text{-lit}\ (n2)))\ s' \rangle \Leftarrow \tau$   
**shows**  $\Theta; \Phi; \Delta \vdash \langle \delta, AS\text{-let}\ x\ (AE\text{-val}\ (V\text{-lit}\ (((if\ (n1 = n2)\ then\ L\text{-true}\ else\ L\text{-false})))))\ s' \rangle \Leftarrow \tau$   
 $\langle \text{proof} \rangle$

## 16.2.3 Let Statements

**lemma** *subst-s-abs-let*:

**fixes**  $s::s$  **and**  $sa::s$  **and**  $v':v$   
**assumes**  $[[atom\ x]]\ lst.\ s = [[atom\ xa]]\ lst.\ sa$  **and**  $atom\ xa \# v \wedge atom\ x \# v$   
**shows**  $s[x::=v]_{sv} = sa[xa::=v]_{sv}$   
 $\langle \text{proof} \rangle$

**lemma** *check-let-val*:

**fixes**  $v::v$  **and**  $s::s$   
**shows**  $\Theta; \Phi; B; G; \Delta \vdash ss \Leftarrow \tau \implies B = \{\|\} \implies G = GNil \implies$   
 $ss = AS\text{-let}\ x\ (AE\text{-val}\ v)\ s \vee ss = AS\text{-let2}\ x\ t\ (AS\text{-val}\ v)\ s \implies \Theta; \Phi; \{\|\}; GNil; \Delta \vdash (s[x::=v]_{sv})$   
 $\Leftarrow \tau$  **and**  
*check-branch-s*  $\Theta\ \Phi\ \mathcal{B}\ GNil\ \Delta\ tid\ dc\ const\ v\ cs\ \tau \implies True$  **and**  
*check-branch-list*  $\Theta\ \Phi\ \mathcal{B}\ \Gamma\ \Delta\ tid\ dclist\ v\ css\ \tau \implies True$   
 $\langle \text{proof} \rangle$

**lemma** *preservation-let-val*:

**assumes**  $\Theta; \Phi; \Delta \vdash \langle \delta, AS\text{-let}\ x\ (AE\text{-val}\ v)\ s \rangle \Leftarrow \tau \vee \Theta; \Phi; \Delta \vdash \langle \delta, AS\text{-let2}\ x\ t\ (AS\text{-val}\ v)\ s \rangle \Leftarrow \tau$  **(is**  $?A \vee ?B$ **)**  
**shows**  $\exists \Delta'. \Theta; \Phi; \Delta' \vdash \langle \delta, s[x::=v]_{sv} \rangle \Leftarrow \tau \wedge \Delta \sqsubseteq \Delta'$   
 $\langle \text{proof} \rangle$

**lemma** *check-s-fst-snd*:

**assumes**  $fst\text{-snd} = AE\text{-fst} \wedge v=v1 \vee fst\text{-snd} = AE\text{-snd} \wedge v=v2$   
**and**  $\Theta; \Phi; \{\|\}; GNil; \Delta \vdash AS\text{-let}\ x\ (fst\text{-snd}\ (V\text{-pair}\ v1\ v2))\ s' \Leftarrow \tau$

**shows**  $\Theta; \Phi; \{\|\}; GNil; \Delta \vdash AS\text{-let } x (AE\text{-val } v) s' \Leftarrow \tau$   
 <proof>

**lemma** *preservation-fst-snd*:

**assumes**  $\Theta; \Phi; \Delta \vdash \langle \delta, LET\ x = (fst\text{-snd } (V\text{-pair } v1\ v2))\ IN\ s' \rangle \Leftarrow \tau$  **and**

$fst\text{-snd} = AE\text{-fst} \wedge v=v1 \vee fst\text{-snd} = AE\text{-snd} \wedge v=v2$

**shows**  $\exists \Delta'. \Theta; \Phi; \Delta \vdash \langle \delta, LET\ x = (AE\text{-val } v)\ IN\ s' \rangle \Leftarrow \tau \wedge \Delta \sqsubseteq \Delta'$

<proof>

**inductive-cases** *check-branch-s-elim2[elim!]*:

$\Theta; \Phi; \mathcal{B}; \Gamma; \Delta; tid; cons; const; v \vdash cs \Leftarrow \tau$

**lemmas** *freshers = freshers atom-dom.simps toSet.simps fresh-def x-not-in-b-set*

**declare** *freshers [simp]*

**lemma** *subtype-eq-if*:

**fixes**  $t::\tau$  **and**  $va::v$

**assumes**  $\Theta; \mathcal{B}; \Gamma \vdash_{wf} \{z : b\text{-of } t \mid c\text{-of } t\ z\}$  **and**  $\Theta; \mathcal{B}; \Gamma \vdash_{wf} \{z : b\text{-of } t \mid c\ IMP\ c\text{-of } t\ z\}$

**shows**  $\Theta; \mathcal{B}; \Gamma \vdash \{z : b\text{-of } t \mid c\text{-of } t\ z\} \lesssim \{z : b\text{-of } t \mid c\ IMP\ c\text{-of } t\ z\}$

<proof>

**lemma** *subtype-eq-if- $\tau$* :

**fixes**  $t::\tau$  **and**  $va::v$

**assumes**  $\Theta; \mathcal{B}; \Gamma \vdash_{wf} t$  **and**  $\Theta; \mathcal{B}; \Gamma \vdash_{wf} \{z : b\text{-of } t \mid c\ IMP\ c\text{-of } t\ z\}$  **and**  $atom\ z \# t$

**shows**  $\Theta; \mathcal{B}; \Gamma \vdash t \lesssim \{z : b\text{-of } t \mid c\ IMP\ c\text{-of } t\ z\}$

<proof>

**lemma** *valid-conj*:

**assumes**  $\Theta; \mathcal{B}; \Gamma \models c1$  **and**  $\Theta; \mathcal{B}; \Gamma \models c2$

**shows**  $\Theta; \mathcal{B}; \Gamma \models c1\ AND\ c2$

<proof>

## 16.2.4 Other Statements

**lemma** *check-if*:

**fixes**  $s'::s$  **and**  $cs::branch\text{-}s$  **and**  $css::branch\text{-}list$  **and**  $v::v$

**shows**  $\Theta; \Phi; \mathcal{B}; G; \Delta \vdash s' \Leftarrow \tau \implies s' = IF\ (V\text{-lit } ll)\ THEN\ s1\ ELSE\ s2 \implies$

$\Theta; \{\|\}; GNil \vdash_{wf} \tau \implies G = GNil \implies B = \{\|\} \implies ll = L\text{-true} \wedge s = s1 \vee ll = L\text{-false} \wedge s = s2 \implies$

$\Theta; \Phi; \{\|\}; GNil; \Delta \vdash s \Leftarrow \tau$  **and**

*check-branch-s*  $\Theta\ \Phi\ \{\|\}\ GNil\ \Delta\ tid\ dc\ const\ v\ cs\ \tau \implies True$  **and**

*check-branch-list*  $\Theta\ \Phi\ \{\|\}\ \Gamma\ \Delta\ tid\ dclist\ v\ css\ \tau \implies True$

<proof>

**lemma** *preservation-if*:

**assumes**  $\Theta; \Phi; \Delta \vdash \langle \delta, IF\ (V\text{-lit } ll)\ THEN\ s1\ ELSE\ s2 \rangle \Leftarrow \tau$  **and**

$ll = L\text{-true} \wedge s = s1 \vee ll = L\text{-false} \wedge s = s2$

**shows**  $\Theta; \Phi; \Delta \vdash \langle \delta, s \rangle \Leftarrow \tau \wedge setD\ \Delta \subseteq setD\ \Delta$

<proof>

**lemma** *wfT-conj*:

**assumes**  $\Theta; \mathcal{B}; GNil \vdash_{wf} \{z : b \mid c1\}$  **and**  $\Theta; \mathcal{B}; GNil \vdash_{wf} \{z : b \mid c2\}$

**shows**  $\Theta; \mathcal{B}; GNil \vdash_{wf} \{z : b \mid c1\ AND\ c2\}$

*<proof>*

**lemma** *subtype-conj*:

**assumes**  $\Theta ; \mathcal{B} ; GNil \vdash t \lesssim \{ z : b \mid c1 \}$  **and**  $\Theta ; \mathcal{B} ; GNil \vdash t \lesssim \{ z : b \mid c2 \}$   
**shows**  $\Theta ; \mathcal{B} ; GNil \vdash \{ z : b \mid c\text{-of } t \ z \} \lesssim \{ z : b \mid c1 \text{ AND } c2 \}$

*<proof>*

**lemma** *infer-v-conj*:

**assumes**  $\Theta ; \mathcal{B} ; GNil \vdash v \Leftarrow \{ z : b \mid c1 \}$  **and**  $\Theta ; \mathcal{B} ; GNil \vdash v \Leftarrow \{ z : b \mid c2 \}$   
**shows**  $\Theta ; \mathcal{B} ; GNil \vdash v \Leftarrow \{ z : b \mid c1 \text{ AND } c2 \}$

*<proof>*

**lemma** *wfG-conj*:

**fixes**  $c1::c$   
**assumes**  $\Theta ; \mathcal{B} \vdash_{wf} (x, b, c1 \text{ AND } c2) \#_{\Gamma} \Gamma$   
**shows**  $\Theta ; \mathcal{B} \vdash_{wf} (x, b, c1) \#_{\Gamma} \Gamma$

*<proof>*

**lemma** *check-match*:

**fixes**  $s'::s$  **and**  $s::s$  **and**  $css::branch\text{-list}$  **and**  $cs::branch\text{-s}$   
**shows**  $\Theta ; \Phi ; \mathcal{B} ; \Gamma ; \Delta \vdash s \Leftarrow \tau \implies True$  **and**  
 $\Theta ; \Phi ; \mathcal{B} ; G ; \Delta ; tid ; dc ; const ; vcons \vdash cs \Leftarrow \tau \implies$   
 $vcons = V\text{-cons } tid \ dc \ v \implies B = \{\|\}$   $\implies G = GNil \implies cs = (dc \ x' \Rightarrow s') \implies$   
 $\Theta ; \{\|\} ; GNil \vdash v \Leftarrow const \implies$   
 $\Theta ; \Phi ; \{\|\} ; GNil ; \Delta \vdash s'[x'::=v]_{sv} \Leftarrow \tau$  **and**  
 $\Theta ; \Phi ; \mathcal{B} ; G ; \Delta ; tid ; dclist ; vcons \vdash css \Leftarrow \tau \implies distinct (map \text{fst } dclist) \implies$   
 $vcons = V\text{-cons } tid \ dc \ v \implies B = \{\|\} \implies (dc, const) \in set \ dclist \implies G = GNil \implies$   
 $Some (AS\text{-branch } dc \ x' \ s') = lookup\text{-branch } dc \ css \implies \Theta ; \{\|\} ; GNil \vdash v \Leftarrow const \implies$   
 $\Theta ; \Phi ; \{\|\} ; GNil ; \Delta \vdash s'[x'::=v]_{sv} \Leftarrow \tau$

*<proof>*

Lemmas for while reduction. Making these separate lemmas allows flexibility in wiring them into the main proof and robustness if we change it

**lemma** *check-unit*:

**fixes**  $\tau::\tau$  **and**  $\Phi::\Phi$  **and**  $\Delta::\Delta$  **and**  $G::\Gamma$   
**assumes**  $\Theta ; \{\|\} ; GNil \vdash \{ z : B\text{-unit} \mid TRUE \} \lesssim \tau'$  **and**  $\Theta ; \{\|\} ; GNil \vdash_{wf} \Delta$  **and**  $\Theta \vdash_{wf} \Phi$   
**and**  $\Theta ; \{\|\} \vdash_{wf} G$   
**shows**  $\Theta ; \Phi ; \{\|\} ; G ; \Delta \vdash [[L\text{-unit}]^v]^s \Leftarrow \tau'$

*<proof>*

**lemma** *preservation-var*:

**shows**  $\Theta ; \Phi ; \{\|\} ; GNil ; \Delta \vdash VAR \ u : \tau' = v \ IN \ s \Leftarrow \tau \implies \Theta \vdash \delta \sim \Delta \implies atom \ u \ \# \ \delta \implies atom \ u \ \# \ \Delta \implies$   
 $\Theta ; \Phi ; \{\|\} ; GNil ; (u, \tau') \#_{\Delta} \Delta \vdash s \Leftarrow \tau \wedge \Theta \vdash (u, v) \# \delta \sim (u, \tau') \#_{\Delta} \Delta$   
**and**  
*check-branch-s*  $\Theta \ \Phi \ \{\|\} \ GNil \ \Delta \ tid \ dc \ const \ v \ cs \ \tau \implies True$  **and**  
*check-branch-list*  $\Theta \ \Phi \ \{\|\} \ \Gamma \ \Delta \ tid \ dclist \ v \ css \ \tau \implies True$

*<proof>*

**lemma** *check-while*:

**shows**  $\Theta ; \Phi ; \{\|\} ; GNil ; \Delta \vdash WHILE \ s1 \ DO \ \{ \ s2 \} \Leftarrow \tau \implies atom \ x \ \# \ (s1, s2) \implies atom \ z' \ \# \ x \implies$   
 $\Theta ; \Phi ; \{\|\} ; GNil ; \Delta \vdash LET \ x : (\{ z' : B\text{-bool} \mid TRUE \}) = s1 \ IN \ (IF \ (V\text{-var } x) \ THEN \ (s2 \ ; \ ;$

(*WHILE*  $s_1$  *DO*  $\{s_2\}$ )  
*ELSE* ( $[V\text{-lit } L\text{-unit}]^s$ )  $\Leftarrow \tau$  **and**  
*check-branch-s*  $\Theta \Phi \{\|\}\ GNil \Delta \text{ tid } dc \text{ const } v \text{ cs } \tau \Longrightarrow \text{True}$  **and**  
*check-branch-list*  $\Theta \Phi \{\|\}\ \Gamma \Delta \text{ tid } dclist \text{ v } \text{css } \tau \Longrightarrow \text{True}$   
 $\langle \text{proof} \rangle$

**lemma** *check-s-narrow*:

**fixes**  $s::s$  **and**  $x::x$

**assumes** *atom*  $x \# (\Theta, \Phi, \mathcal{B}, \Gamma, \Delta, c, \tau, s)$  **and**  $\Theta ; \Phi ; \mathcal{B} ; (x, B\text{-bool}, c) \#_{\Gamma} \Gamma ; \Delta \vdash s \Leftarrow \tau$  **and**  
 $\Theta ; \mathcal{B} ; \Gamma \models c$

**shows**  $\Theta ; \Phi ; \mathcal{B} ; \Gamma ; \Delta \vdash s \Leftarrow \tau$

$\langle \text{proof} \rangle$

**lemma** *check-assert-s*:

**fixes**  $s::s$  **and**  $x::x$

**assumes**  $\Theta ; \Phi ; \{\|\}\ ; GNil ; \Delta \vdash AS\text{-assert } c \ s \Leftarrow \tau$

**shows**  $\Theta ; \Phi ; \{\|\}\ ; GNil ; \Delta \vdash s \Leftarrow \tau \wedge \Theta ; \{\|\}\ ; GNil \models c$

$\langle \text{proof} \rangle$

**lemma** *infer-v-pair2I*:

*atom*  $z \# (v_1, v_2) \Longrightarrow$

*atom*  $z \# (\Theta, \mathcal{B}, \Gamma) \Longrightarrow$

$\Theta ; \mathcal{B} ; \Gamma \vdash v_1 \Rightarrow t_1 \Longrightarrow$

$\Theta ; \mathcal{B} ; \Gamma \vdash v_2 \Rightarrow t_2 \Longrightarrow$

$b_1 = b\text{-of } t_1 \Longrightarrow b_2 = b\text{-of } t_2 \Longrightarrow$

$\Theta ; \mathcal{B} ; \Gamma \vdash [v_1, v_2]^v \Rightarrow \{z : [b_1, b_2]^b \mid [[z]^v]^{ce} == [[v_1, v_2]^v]^{ce}\}$

$\langle \text{proof} \rangle$

## 16.2.5 Main Lemma

**lemma** *preservation*:

**assumes**  $\Phi \vdash \langle \delta, s \rangle \longrightarrow \langle \delta', s' \rangle$  **and**  $\Theta ; \Phi ; \Delta \vdash \langle \delta, s \rangle \Leftarrow \tau$

**shows**  $\exists \Delta'. \Theta ; \Phi ; \Delta' \vdash \langle \delta', s' \rangle \Leftarrow \tau \wedge \Delta \sqsubseteq \Delta'$

$\langle \text{proof} \rangle$

**lemma** *preservation-many*:

**assumes**  $\Phi \vdash \langle \delta, s \rangle \longrightarrow^* \langle \delta', s' \rangle$

**shows**  $\Theta ; \Phi ; \Delta \vdash \langle \delta, s \rangle \Leftarrow \tau \Longrightarrow \exists \Delta'. \Theta ; \Phi ; \Delta' \vdash \langle \delta', s' \rangle \Leftarrow \tau \wedge \Delta \sqsubseteq \Delta'$

$\langle \text{proof} \rangle$

## 16.3 Progress

We prove that a well typed program is either a value or we can make a step

**lemma** *check-let-op-infer*:

**assumes**  $\Theta ; \Phi ; \{\|\}\ ; \Gamma ; \Delta \vdash LET \ x = (AE\text{-op } opp \ v_1 \ v_2) \ IN \ s \Leftarrow \tau$  **and**  $\text{supp} (LET \ x = (AE\text{-op } opp \ v_1 \ v_2) \ IN \ s) \subseteq \text{atom}'fst'setD \ \Delta$

**shows**  $\exists z \ b \ c. \Theta ; \Phi ; \{\|\}\ ; \Gamma ; \Delta \vdash (AE\text{-op } opp \ v_1 \ v_2) \Rightarrow \{z:b|c\}$

$\langle \text{proof} \rangle$

**lemma** *infer-pair*:

**assumes**  $\Theta ; B ; \Gamma \vdash v \Rightarrow \{z : B\text{-pair } b_1 \ b_2 \mid c\}$  **and**  $\text{supp } v = \{\}$

**obtains**  $v1$  **and**  $v2$  **where**  $v = V\text{-pair } v1 \ v2$   
 $\langle \text{proof} \rangle$

**lemma** *progress-fst*:

**assumes**  $\Theta; \Phi; \{\|\}; \Gamma; \Delta \vdash LET \ x = (AE\text{-fst } v) \ IN \ s \Leftarrow \tau$  **and**  $\Theta \vdash \delta \sim \Delta$  **and**  
 $supp \ (LET \ x = (AE\text{-fst } v) \ IN \ s) \subseteq atom\ 'fst'\ setD \ \Delta$   
**shows**  $\exists \delta' \ s'. \ \Phi \vdash \langle \delta, LET \ x = (AE\text{-fst } v) \ IN \ s \rangle \longrightarrow \langle \delta', s' \rangle$   
 $\langle \text{proof} \rangle$

**lemma** *progress-let*:

**assumes**  $\Theta; \Phi; \{\|\}; \Gamma; \Delta \vdash LET \ x = e \ IN \ s \Leftarrow \tau$  **and**  $\Theta \vdash \delta \sim \Delta$  **and**  
 $supp \ (LET \ x = e \ IN \ s) \subseteq atom \ 'fst' \ setD \ \Delta$  **and**  $sble \ \Theta \ \Gamma$   
**shows**  $\exists \delta' \ s'. \ \Phi \vdash \langle \delta, LET \ x = e \ IN \ s \rangle \longrightarrow \langle \delta', s' \rangle$   
 $\langle \text{proof} \rangle$

**lemma** *check-css-lookup-branch-exist*:

**fixes**  $s::s$  **and**  $cs::branch\text{-}s$  **and**  $css::branch\text{-}list$  **and**  $v::v$   
**shows**  
 $\Theta; \Phi; B; G; \Delta \vdash s \Leftarrow \tau \implies True$  **and**  
 $check\text{-}branch\text{-}s \ \Theta \ \Phi \ \{\|\} \ GNil \ \Delta \ tid \ dc \ const \ v \ cs \ \tau \implies True$  **and**  
 $\Theta; \Phi; B; \Gamma; \Delta; tid; dclist; v \vdash css \Leftarrow \tau \implies (dc, t) \in set \ dclist \implies$   
 $\exists x' \ s'. \ Some \ (AS\text{-branch } dc \ x' \ s') = lookup\text{-}branch \ dc \ css$   
 $\langle \text{proof} \rangle$

**lemma** *progress-aux*:

**shows**  $\Theta; \Phi; B; \Gamma; \Delta \vdash s \Leftarrow \tau \implies B = \{\|\} \implies sble \ \Theta \ \Gamma \implies supp \ s \subseteq atom \ 'fst' \ setD \ \Delta \implies$   
 $\Theta \vdash \delta \sim \Delta \implies$   
 $(\exists v. s = [v]^s) \vee (\exists \delta' \ s'. \ \Phi \vdash \langle \delta, s \rangle \longrightarrow \langle \delta', s' \rangle)$  **and**  
 $\Theta; \Phi; \{\|\}; \Gamma; \Delta; tid; dc; const; v2 \vdash cs \Leftarrow \tau \implies supp \ cs = \{\} \implies True$   
 $\Theta; \Phi; \{\|\}; \Gamma; \Delta; tid; dclist; v2 \vdash css \Leftarrow \tau \implies supp \ css = \{\} \implies True$   
 $\langle \text{proof} \rangle$

**lemma** *progress*:

**assumes**  $\Theta; \Phi; \Delta \vdash \langle \delta, s \rangle \Leftarrow \tau$   
**shows**  $(\exists v. s = [v]^s) \vee (\exists \delta' \ s'. \ \Phi \vdash \langle \delta, s \rangle \longrightarrow \langle \delta', s' \rangle)$   
 $\langle \text{proof} \rangle$

## 16.4 Safety

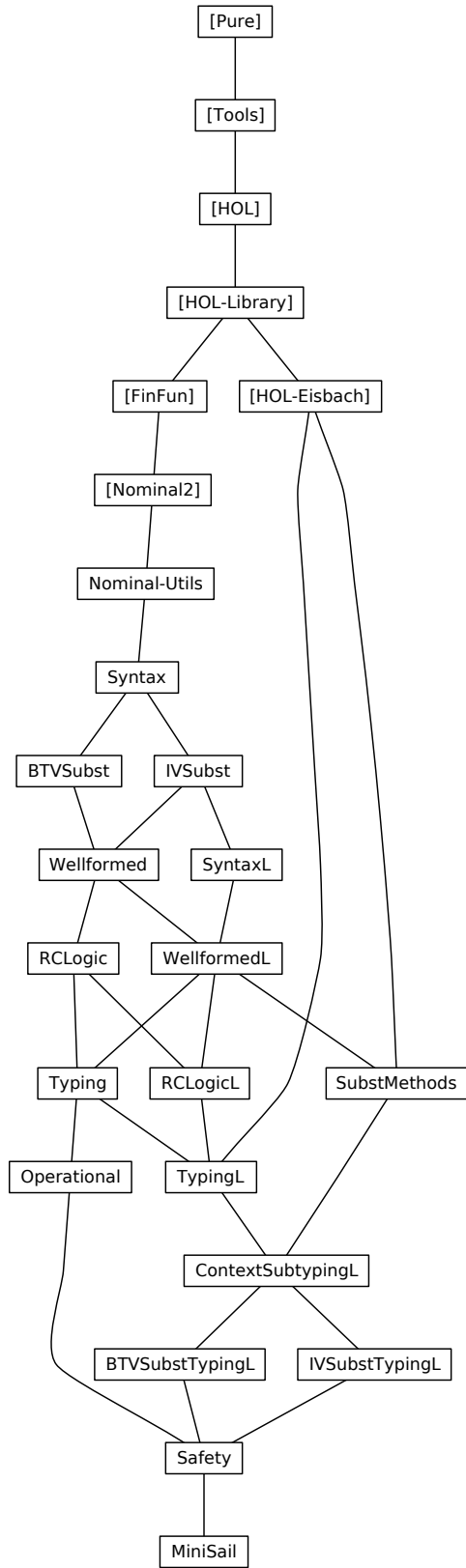
**lemma** *safety-stmt*:

**assumes**  $\Phi \vdash \langle \delta, s \rangle \longrightarrow^* \langle \delta', s' \rangle$  **and**  $\Theta; \Phi; \Delta \vdash \langle \delta, s \rangle \Leftarrow \tau$   
**shows**  $(\exists v. s' = [v]^s) \vee (\exists \delta'' \ s''. \ \Phi \vdash \langle \delta', s' \rangle \longrightarrow \langle \delta'', s'' \rangle)$   
 $\langle \text{proof} \rangle$

**lemma** *safety*:

**assumes**  $\vdash \langle PROG \ \Theta \ \Phi \ \mathcal{G} \ s \rangle \Leftarrow \tau$  **and**  $\Phi \vdash \langle \delta\text{-of } \mathcal{G}, s \rangle \longrightarrow^* \langle \delta', s' \rangle$   
**shows**  $(\exists v. s' = [v]^s) \vee (\exists \delta'' \ s''. \ \Phi \vdash \langle \delta', s' \rangle \longrightarrow \langle \delta'', s'' \rangle)$   
 $\langle \text{proof} \rangle$

**end**



# Bibliography

- [1] A. Armstrong, C. Pulte, S. Flur, I. Stark, N. Krishnaswami, P. Sewell, T. Bauereiss, B. Campbell, A. Reid, K. E. Gray, R. M. Norton, P. Mundkur, M. Wassell, and J. French. ISA semantics for ARMv8-A, RISC-V, and CHERI-MIPS. *Proceedings of the ACM on Programming Languages*, 3(POPL):1–31, 2019.
- [2] N. Vazou, E. L. Seidel, and S. Peyton-jones. Refinement Types For Haskell. *ICFP '14 Proceedings of the 19th ACM SIGPLAN international conference on Functional programming*, 2014.