

# NREST

Maximilian P.L. Haslbeck

February 6, 2026

## Abstract

This entry introduces NREST, a nondeterministic result monad with time, which was spun off a development of a framework for verifying functional correctness and worst-case complexity of algorithms refined down to LLVM, due to Haslbeck and Lammich.

## Contents

<b>1</b>	<b>Auxiliaries</b>	<b>2</b>
1.1	Auxiliaries for option . . . . .	2
1.2	Auxiliaries for enat . . . . .	3
1.3	Auxiliary (for Sup and Inf) . . . . .	3
<b>2</b>	<b>NREST</b>	<b>4</b>
<b>3</b>	<b>pointwise reasoning</b>	<b>7</b>
<b>4</b>	<b>Monad Operators</b>	<b>9</b>
<b>5</b>	<b>Monad Rules</b>	<b>10</b>
<b>6</b>	<b>Monotonicity lemmas</b>	<b>10</b>
6.1	Derived Program Constructs . . . . .	10
6.1.1	Assertion . . . . .	10
6.2	SELECT . . . . .	11
<b>7</b>	<b>RECT</b>	<b>12</b>
<b>8</b>	<b>Generalized Weakest Precondition</b>	<b>14</b>
8.1	mm . . . . .	14
8.2	mii . . . . .	15
8.3	lst - latest starting time . . . . .	16
8.4	pointwise reasoning about lst via nres3 . . . . .	17
8.5	rules for lst . . . . .	17

<b>9</b>	<b>consequence rules</b>	<b>18</b>
<b>10</b>	<b>Experimental Hoare reasoning</b>	<b>19</b>
<b>11</b>	<b>VCG</b>	<b>19</b>
11.1	Progress rules . . . . .	20
<b>12</b>	<b>rules for whileT</b>	<b>21</b>
<b>13</b>	<b>some Monadic Refinement Automation</b>	<b>22</b>
13.1	Data Refinement . . . . .	22
13.1.1	Examples . . . . .	22
13.2	WHILET refine . . . . .	26
13.3	ASSERT . . . . .	27
13.4	VCG splitter . . . . .	28
13.5	mm3 and emb . . . . .	28
13.6	Setup Labeled VCG . . . . .	29
13.7	Examples, labeled vcg . . . . .	35
13.8	progress solver . . . . .	36
13.9	more stuff involving mm3 and emb . . . . .	36
13.10	VCG for monadic programs . . . . .	37
13.10.1	old . . . . .	37
13.10.2	new setup . . . . .	37
13.11	setup for <i>refine-vcg</i> . . . . .	38
13.12	Relators . . . . .	38
13.13	Auxilliary Lemmas . . . . .	39
13.14	Definition . . . . .	39
13.15	Proof Rules . . . . .	40
13.16	Nres-Fold with Interruption ( <i>nfoldli</i> ) . . . . .	41

**theory**

*NREST-Auxiliaries*

**imports**

*HOL-Library.Extended-Nat Automatic-Refinement.Automatic-Refinement*

**begin**

**unbundle** *lattice-syntax*

## 1 Auxiliaries

### 1.1 Auxiliaries for option

**lemma** *less-eq-option-None-is-None'*:  $x \leq \text{None} \longleftrightarrow x = \text{None}$

*<proof>*

**lemma** *everywhereNone*:  $(\forall x \in X. x = \text{None}) \longleftrightarrow X = \{\}$   $\vee$   $X = \{\text{None}\}$

*<proof>*

## 1.2 Auxiliaries for enat

**lemma** *enat-minus-mono*:  $a' \geq b \implies a' \geq a \implies a' - b \geq (a::\text{enat}) - b$   
 ⟨proof⟩

**lemma** *enat-plus-minus-aux1*:  $a + b \leq a' \implies \neg a' < a \implies b \leq a' - (a::\text{enat})$   
 ⟨proof⟩

**lemma** *enat-plus-minus-aux2*:  $\neg a < a' \implies b \leq a - a' \implies a' + b \leq (a::\text{enat})$   
 ⟨proof⟩

**lemma** *enat-minus-inf-conv[simp]*:  $a - \text{enat } n = \infty \longleftrightarrow a = \infty$  ⟨proof⟩

**lemma** *enat-minus-fin-conv[simp]*:  $a - \text{enat } n = \text{enat } m \longleftrightarrow (\exists k. a = \text{enat } k \wedge m = k - n)$   
 ⟨proof⟩

**lemma** *helper*:  $x2 \leq x2a \implies \neg x2 < a \implies \neg x2a < a \implies x2 - (a::\text{enat}) \leq x2a - a$   
 ⟨proof⟩

**lemma** *helper2*:  $x2b \leq x2 \implies \neg x2a < x2 \implies \neg x2a < x2b \implies x2a - (x2::\text{enat}) \leq x2a - x2b$   
 ⟨proof⟩

**lemma** *Sup-finite-enat*:  $\text{Sup } X = \text{Some } (\text{enat } a) \implies \text{Some } (\text{enat } a) \in X$   
 ⟨proof⟩

**lemma** *Sup-enat-less2*:  $\text{Sup } X = \infty \implies \exists x \in X. \text{enat } t < x$   
 ⟨proof⟩

**lemma** *enat-upper[simp]*:  $t \leq \text{Some } (\infty::\text{enat})$   
 ⟨proof⟩

## 1.3 Auxiliary (for Sup and Inf)

**lemma** *aux11*:  $f'X = \{y\} \longleftrightarrow (X \neq \{\}) \wedge (\forall x \in X. f x = y)$  ⟨proof⟩

**lemma** *aux2*:  $(\lambda f. f x) ' \{[x \mapsto t1] \mid x t1. M x = \text{Some } t1\} = \{\text{None}\} \longleftrightarrow (M x = \text{None} \wedge M \neq \text{Map.empty})$   
 ⟨proof⟩

**lemma** *aux3*:  $(\lambda f. f x) ' \{[x \mapsto t1] \mid x t1. M x = \text{Some } t1\} = \{\text{Some } t1 \mid t1. M x = \text{Some } t1\} \cup (\{\text{None} \mid y. y \neq x \wedge M y \neq \text{None}\})$   
 ⟨proof⟩

**lemma** *Sup-pointwise-eq-fun*:  $(\bigsqcup f \in \{[x \mapsto t1] \mid x t1. M x = \text{Some } t1\}. f x) = M x$   
 ⟨proof⟩

**lemma** *SUP-eq-None-iff*:  $(\bigsqcup f \in X. f x) = \text{None} \longleftrightarrow X = \{\} \vee (\forall f \in X. f x =$

None)  
⟨proof⟩

**lemma** *SUP-eq-Some-iff*:

$(\bigsqcup f \in X. f x) = \text{Some } t \iff (\exists f \in X. f x \neq \text{None}) \wedge (t = \text{Sup } \{t' \mid f t'. f \in X \wedge f x = \text{Some } t'\})$   
⟨proof⟩

**lemma** *Sup-enat-less*:  $X \neq \{\}$   $\implies \text{enat } t \leq \text{Sup } X \iff (\exists x \in X. \text{enat } t \leq x)$   
⟨proof⟩

**lemma** *fixes Q P shows*

$\text{Inf } \{ P x \leq Q x \mid x. \text{True} \} \iff P \leq Q$   
⟨proof⟩

**end**

**theory** *NREST*

**imports** *HOL-Library.Extended-Nat Refine-Monadic.RefineG-Domain Refine-Monadic.Refine-Misc*

*HOL-Library.Monad-Syntax NREST-Auxiliaries*

**begin**

## 2 NREST

**datatype** *'a nrest* = *FAILi* | *REST 'a*  $\Rightarrow$  *enat option*

**instantiation** *nrest* :: (*type*) *complete-lattice*

**begin**

**fun** *less-eq-nrest* **where**

$- \leq \text{FAILi} \iff \text{True} \mid$   
 $(\text{REST } a) \leq (\text{REST } b) \iff a \leq b \mid$   
 $\text{FAILi} \leq (\text{REST } -) \iff \text{False}$

**fun** *less-nrest* **where**

$\text{FAILi} < - \iff \text{False} \mid$   
 $(\text{REST } -) < \text{FAILi} \iff \text{True} \mid$   
 $(\text{REST } a) < (\text{REST } b) \iff a < b$

**fun** *sup-nrest* **where**

$\text{sup } - \text{ FAILi} = \text{FAILi} \mid$   
 $\text{sup } \text{FAILi } - = \text{FAILi} \mid$   
 $\text{sup } (\text{REST } a) (\text{REST } b) = \text{REST } (\lambda x. \text{max } (a x) (b x))$

**fun** *inf-nrest* **where**

$\text{inf } x \text{ FAIL}i = x \mid$   
 $\text{inf } \text{FAIL}i \ x = x \mid$   
 $\text{inf } (\text{REST } a) (\text{REST } b) = \text{REST } (\lambda x. \text{min } (a \ x) (b \ x))$

**lemma**  $\text{min } (\text{None}) (\text{Some } (1::\text{enat})) = \text{None}$   $\langle \text{proof} \rangle$

**lemma**  $\text{max } (\text{None}) (\text{Some } (1::\text{enat})) = \text{Some } 1$   $\langle \text{proof} \rangle$

**definition**  $\text{Sup } X \equiv \text{if } \text{FAIL}i \in X \text{ then } \text{FAIL}i \text{ else } \text{REST } (\text{Sup } \{f . \text{REST } f \in X\})$

**definition**  $\text{Inf } X \equiv \text{if } \exists f. \text{REST } f \in X \text{ then } \text{REST } (\text{Inf } \{f . \text{REST } f \in X\}) \text{ else } \text{FAIL}i$

**definition**  $\text{bot} \equiv \text{REST } (\text{Map.empty})$

**definition**  $\text{top} \equiv \text{FAIL}i$

**instance**

$\langle \text{proof} \rangle$

**end**

**definition**  $\text{RETURN}T \ x \equiv \text{REST } (\lambda e. \text{if } e=x \text{ then } \text{Some } 0 \text{ else } \text{None})$

**abbreviation**  $\text{FAIL}T \equiv \text{top}::'a \ \text{nrest}$

**abbreviation**  $\text{SUCCEED}T \equiv \text{bot}::'a \ \text{nrest}$

**abbreviation**  $\text{SPECT where } \text{SPECT} \equiv \text{REST}$

**lemma**  $\text{RETURN}T\text{-alt}: \text{RETURN}T \ x = \text{REST } [x \mapsto 0]$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{nrest-inequalities}[\text{simp}]$ :

$\text{FAIL}T \neq \text{REST } X$

$\text{FAIL}T \neq \text{SUCCEED}T$

$\text{FAIL}T \neq \text{RETURN}T \ x$

$\text{SUCCEED}T \neq \text{FAIL}T$

$\text{SUCCEED}T \neq \text{RETURN}T \ x$

$\text{REST } X \neq \text{FAIL}T$

$\text{RETURN}T \ x \neq \text{FAIL}T$

$\text{RETURN}T \ x \neq \text{SUCCEED}T$

$\langle \text{proof} \rangle$

**lemma**  $\text{nrest-more-simps}[\text{simp}]$ :

$\text{SUCCEED}T = \text{REST } X \longleftrightarrow X = \text{Map.empty}$

$\text{REST } X = \text{SUCCEED}T \longleftrightarrow X = \text{Map.empty}$

$\text{REST } X = \text{RETURN}T \ x \longleftrightarrow X = [x \mapsto 0]$

$\text{REST } X = \text{REST } Y \longleftrightarrow X = Y$

$\text{RETURN}T \ x = \text{REST } X \longleftrightarrow X = [x \mapsto 0]$

$\text{RETURN}T \ x = \text{RETURN}T \ y \longleftrightarrow x = y$

$\langle \text{proof} \rangle$

**lemma**  $\text{nres-simp-internals}$ :

$\text{REST } \text{Map.empty} = \text{SUCCEED}T$

$FAILi = FAILT$   
 $\langle proof \rangle$

**lemma** *nres-order-simps*[simp]:  
 $\neg FAILT \leq REST M$   
 $REST M \leq REST M' \longleftrightarrow (M \leq M')$   
 $\langle proof \rangle$

**lemma** *nres-top-unique*[simp]:  $FAILT \leq S' \longleftrightarrow S' = FAILT$   
 $\langle proof \rangle$

**lemma** *FAILT-cases*[simp]:  $(case\ FAILT\ of\ FAILi \Rightarrow P \mid REST\ x \Rightarrow Q\ x) = P$   
 $\langle proof \rangle$

**lemma** *nrest-Sup-FAILT*:  
 $Sup\ X = FAILT \longleftrightarrow FAILT \in X$   
 $FAILT = Sup\ X \longleftrightarrow FAILT \in X$   
 $\langle proof \rangle$

**lemma** *nrest-Sup-SPECT-D*:  $Sup\ X = SPECT\ m \implies m\ x = Sup\ \{f\ x \mid f.\ REST\ f \in X\}$   
 $\langle proof \rangle$

**declare** *nres-simp-internals*(2)[simp]

**lemma** *nrest-noREST-FAILT*[simp]:  $(\forall x2.\ m \neq REST\ x2) \longleftrightarrow m = FAILT$   
 $\langle proof \rangle$

**lemma** *no-FAILTE*:  
**assumes**  $g\ xa \neq FAILT$   
**obtains**  $X$  **where**  $g\ xa = REST\ X$   
 $\langle proof \rangle$

**lemma** *case-prod-refine*:  
**fixes**  $P\ Q :: 'a \Rightarrow 'b \Rightarrow 'c\ nrest$   
**assumes**  $\bigwedge a\ b.\ P\ a\ b \leq Q\ a\ b$   
**shows**  $(case\ x\ of\ (a,b) \Rightarrow P\ a\ b) \leq (case\ x\ of\ (a,b) \Rightarrow Q\ a\ b)$   
 $\langle proof \rangle$

**lemma** *case-option-refine*:  
**fixes**  $P\ Q :: 'a \Rightarrow 'b \Rightarrow 'c\ nrest$   
**assumes**  
     $PN \leq QN$   
     $\bigwedge a.\ PS\ a \leq QS\ a$   
**shows**  $(case\ x\ of\ None \Rightarrow PN \mid Some\ a \Rightarrow PS\ a) \leq (case\ x\ of\ None \Rightarrow QN \mid Some\ a \Rightarrow QS\ a)$   
 $\langle proof \rangle$

**lemma** *SPECT-Map-empty*[simp]:  $SPECT\ Map.empty \leq a$

*<proof>*

**lemma** *FAILT-SUP*:  $(FAILT \in X) \implies Sup X = FAILT$   
*<proof>*

### 3 pointwise reasoning

**named-theorems** *refine-pw-simps*  
*<ML>*

**definition** *nofailT* :: 'a nrest  $\Rightarrow$  bool **where** *nofailT* S  $\equiv S \neq FAILT$

**definition** *le-or-fail* :: 'a nrest  $\Rightarrow$  'a nrest  $\Rightarrow$  bool (**infix**  $\langle \leq_n \rangle$  50) **where**  
 $m \leq_n m' \equiv \text{nofailT } m \longrightarrow m \leq m'$

**lemma** *nofailT-simps[simp]*:  
*nofailT* FAILT  $\longleftrightarrow$  False  
*nofailT* (REST X)  $\longleftrightarrow$  True  
*nofailT* (RETURNT x)  $\longleftrightarrow$  True  
*nofailT* SUCCEEDT  $\longleftrightarrow$  True  
*<proof>*

**definition** *inresT* :: 'a nrest  $\Rightarrow$  'a  $\Rightarrow$  nat  $\Rightarrow$  bool **where**  
 $\text{inresT } S x t \equiv (\text{case } S \text{ of } FAILi \Rightarrow \text{True} \mid \text{REST } X \Rightarrow (\exists t'. X x = \text{Some } t' \wedge \text{enat } t \leq t'))$

**lemma** *inresT-alt*:  $\text{inresT } S x t \longleftrightarrow \text{REST } ([x \rightarrow \text{enat } t]) \leq S$   
*<proof>*

**lemma** *inresT-mono*:  $\text{inresT } S x t \implies t' \leq t \implies \text{inresT } S x t'$   
*<proof>*

**lemma** *inresT-RETURNT[simp]*:  $\text{inresT } (\text{RETURNT } x) y t \longleftrightarrow t = 0 \wedge y = x$   
*<proof>*

**lemma** *inresT-FAILT[simp]*:  $\text{inresT } FAILT r t$   
*<proof>*

**lemma** *fail-inresT[refine-pw-simps]*:  $\neg \text{nofailT } M \implies \text{inresT } M x t$   
*<proof>*

**lemma** *pw-inresT-Sup[refine-pw-simps]*:  $\text{inresT } (\text{Sup } X) r t \longleftrightarrow (\exists M \in X. \exists t' \geq t. \text{inresT } M r t')$   
*<proof>*

**lemma** *inresT-REST[simp]*:  
 $\text{inresT } (\text{REST } X) x t \longleftrightarrow (\exists t' \geq t. X x = \text{Some } t')$   
*<proof>*

**lemma** *pw-Sup-nofail*[*refine-pw-simps*]:  $\text{nofailT } (\text{Sup } X) \longleftrightarrow (\forall x \in X. \text{nofailT } x)$   
 ⟨*proof*⟩

**lemma** *inres-simps*[*simp*]:  
 $\text{inresT } \text{FAILT} = (\lambda - . \text{True})$   
 $\text{inresT } \text{SUCCEEDT} = (\lambda - . \text{False})$   
 ⟨*proof*⟩

**lemma** *pw-le-iff*:  
 $S \leq S' \longleftrightarrow (\text{nofailT } S' \longrightarrow (\text{nofailT } S \wedge (\forall x t. \text{inresT } S x t \longrightarrow \text{inresT } S' x t)))$   
 ⟨*proof*⟩

**lemma** *RETURN-le-RETURN-iff*[*simp*]:  $\text{RETURNT } x \leq \text{RETURNT } y \longleftrightarrow x=y$   
 ⟨*proof*⟩

**lemma**  $S \leq S' \implies \text{inresT } S x t \implies \text{inresT } S' x t$   
 ⟨*proof*⟩

**lemma** *pw-eq-iff*:  
 $S=S' \longleftrightarrow (\text{nofailT } S = \text{nofailT } S' \wedge (\forall x t. \text{inresT } S x t \longleftrightarrow \text{inresT } S' x t))$   
 ⟨*proof*⟩

**lemma** *pw-flat-ge-iff*:  $\text{flat-ge } S S' \longleftrightarrow$   
 $(\text{nofailT } S) \longrightarrow \text{nofailT } S' \wedge (\forall x t. \text{inresT } S x t \longleftrightarrow \text{inresT } S' x t)$   
 ⟨*proof*⟩

**lemma** *pw-eqI*:  
**assumes**  $\text{nofailT } S = \text{nofailT } S'$   
**assumes**  $\bigwedge x t. \text{inresT } S x t \longleftrightarrow \text{inresT } S' x t$   
**shows**  $S=S'$   
 ⟨*proof*⟩

**definition** *consume*  $M t \equiv \text{case } M \text{ of}$   
 $\text{FAIL}i \Rightarrow \text{FAILT} \mid$   
 $\text{REST } X \Rightarrow \text{REST } (\text{map-option } ((+) t) o (X))$

**definition** *SPEC*  $P t = \text{REST } (\lambda v. \text{if } P v \text{ then } \text{Some } (t v) \text{ else } \text{None})$

**lemma** *consume-mono*:  
**assumes**  $t \leq t' \ M \leq M'$   
**shows**  $\text{consume } M t \leq \text{consume } M' t'$   
 ⟨*proof*⟩

**lemma** *nofailT-SPEC*[*refine-pw-simps*]:  $\text{nofailT } (\text{SPEC } a b)$   
 ⟨*proof*⟩

**lemma** *inresT-SPEC*[*refine-pw-simps*]:  $\text{inresT } (\text{SPEC } a b) = (\lambda x t. a x \wedge b x \geq t)$   
 ⟨*proof*⟩

## 4 Monad Operators

**definition**  $bindT :: 'b \text{ nrest} \Rightarrow ('b \Rightarrow 'a \text{ nrest}) \Rightarrow 'a \text{ nrest}$  **where**

$bindT \ M \ f \equiv \text{case } M \ \text{of}$   
 $FAILi \Rightarrow FAILT \ |$   
 $REST \ X \Rightarrow \text{Sup} \{ (\text{case } f \ x \ \text{of } FAILi \Rightarrow FAILT$   
 $\quad | \ REST \ m2 \Rightarrow REST \ (\text{map-option } ((+) \ t1) \ o \ (m2)) \ )$   
 $\quad | \ x \ t1. \ X \ x = \text{Some } t1 \}$

**lemma**  $bindT\text{-alt}$ :  $bindT \ M \ f = (\text{case } M \ \text{of}$   
 $FAILi \Rightarrow FAILT \ |$   
 $REST \ X \Rightarrow \text{Sup} \{ \text{consume } (f \ x) \ t1 \ | \ x \ t1. \ X \ x = \text{Some } t1 \})$   
 $\langle \text{proof} \rangle$

**lemma**  $bindT \ (REST \ X) \ f =$   
 $(\bigsqcup x \in \text{dom } X. \text{consume } (f \ x) \ (\text{the } (X \ x)))$   
 $\langle \text{proof} \rangle$

**adhoc-overloading**

$\text{Monad-Syntax.bind} \equiv \text{NREST.bindT}$

**lemma**  $bindT\text{-FAIL[simp]}$ :  $bindT \ FAILT \ g = FAILT$   
 $\langle \text{proof} \rangle$

**lemma**  $bindT \ SUCCEEDT \ f = SUCCEEDT$   
 $\langle \text{proof} \rangle$

**lemma**  $m \ r = \text{Some } \infty \implies \text{inresT } (REST \ m) \ r \ t$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{pw-inresT-bindT-aux}$ :  $\text{inresT } (bindT \ m \ f) \ r \ t \longleftrightarrow$   
 $(\text{nofailT } m \longrightarrow (\exists r' \ t' \ t''. \text{inresT } m \ r' \ t' \wedge \text{inresT } (f \ r') \ r \ t'' \wedge t \leq t' + t''))$   
 $(\text{is } ?l \longleftrightarrow ?r)$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{pw-inresT-bindT[refine-pw-simps]}$ :  $\text{inresT } (bindT \ m \ f) \ r \ t \longleftrightarrow$   
 $(\text{nofailT } m \longrightarrow (\exists r' \ t' \ t''. \text{inresT } m \ r' \ t' \wedge \text{inresT } (f \ r') \ r \ t'' \wedge t = t' + t''))$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{pw-bindT-nofailT[refine-pw-simps]}$ :  $\text{nofailT } (bindT \ M \ f) \longleftrightarrow (\text{nofailT } M$   
 $\wedge (\forall x \ t. \text{inresT } M \ x \ t \longrightarrow \text{nofailT } (f \ x)))$   $(\text{is } ?l \longleftrightarrow ?r)$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{nat-plus-0-is-id[simp]}$ :  $((+) \ (0::\text{enat})) = \text{id}$   $\langle \text{proof} \rangle$

**declare**  $\text{map-option.id[simp]}$

## 5 Monad Rules

**lemma** *nres-bind-left-identity[simp]*:  $\text{bindT } (\text{RETURNNT } x) f = f x$   
 ⟨proof⟩

**lemma** *nres-bind-right-identity[simp]*:  $\text{bindT } M \text{ RETURNNT} = M$   
 ⟨proof⟩

**lemma** *nres-bind-assoc[simp]*:  $\text{bindT } (\text{bindT } M (\lambda x. f x)) g = \text{bindT } M (\%x. \text{bindT } (f x) g)$   
 ⟨proof⟩

## 6 Monotonicity lemmas

**lemma** *bindT-mono*:  
 $m \leq m' \implies (\bigwedge x. (\exists t. \text{inresT } m x t) \implies \text{nofailT } m' \implies f x \leq f' x)$   
 $\implies \text{bindT } m f \leq \text{bindT } m' f'$   
 ⟨proof⟩

**lemma** *bindT-mono'[refine-mono]*:  
 $m \leq m' \implies (\bigwedge x. f x \leq f' x)$   
 $\implies \text{bindT } m f \leq \text{bindT } m' f'$   
 ⟨proof⟩

**lemma** *bindT-flat-mono[refine-mono]*:  
 $\llbracket \text{flat-ge } M M'; \bigwedge x. \text{flat-ge } (f x) (f' x) \rrbracket \implies \text{flat-ge } (\text{bindT } M f) (\text{bindT } M' f')$   
 ⟨proof⟩

### 6.1 Derived Program Constructs

#### 6.1.1 Assertion

**definition** *iASSERT*  $\text{ret } \Phi \equiv \text{if } \Phi \text{ then } \text{ret } () \text{ else } \text{top}$

**definition** *ASSERT where ASSERT*  $\equiv \text{iASSERT RETURNNT}$

**lemma** *ASSERT-True[simp]*:  $\text{ASSERT True} = \text{RETURNNT } ()$   
 ⟨proof⟩

**lemma** *ASSERT-False[simp]*:  $\text{ASSERT False} = \text{FAILT}$   
 ⟨proof⟩

**lemma** *bind-ASSERT-eq-if*:  $\text{do } \{ \text{ASSERT } \Phi; m \} = (\text{if } \Phi \text{ then } m \text{ else } \text{FAILT})$   
 ⟨proof⟩

**lemma** *pw-ASSERT[refine-pw-simps]*:  
 $\text{nofailT } (\text{ASSERT } \Phi) \longleftrightarrow \Phi$   
 $\text{inresT } (\text{ASSERT } \Phi) x 0$   
 ⟨proof⟩

**lemma** *ASSERT-refine*:  $(Q \implies P) \implies \text{ASSERT } P \leq \text{ASSERT } Q$   
 ⟨proof⟩

**lemma** *ASSERT-leI*:  $\Phi \implies (\Phi \implies M \leq M') \implies \text{ASSERT } \Phi \ggg (\lambda-. M) \leq M'$   
 ⟨proof⟩

**lemma** *le-ASSERTI*:  $(\Phi \implies M \leq M') \implies M \leq \text{ASSERT } \Phi \ggg (\lambda-. M')$   
 ⟨proof⟩

**lemma** *inresT-ASSERT*:  $\text{inresT } (\text{ASSERT } Q \ggg (\lambda-. M)) \ x \ ta = (Q \longrightarrow \text{inresT } M \ x \ ta)$   
 ⟨proof⟩

**lemma** *nofailT-ASSERT-bind*:  $\text{nofailT } (\text{ASSERT } P \ggg (\lambda-. M)) \longleftrightarrow (P \wedge \text{nofailT } M)$   
 ⟨proof⟩

## 6.2 SELECT

**definition** *emb' where*  $\bigwedge Q \ T. \text{emb}' Q (T::'a \Rightarrow \text{enat}) = (\lambda x. \text{if } Q \ x \ \text{then } \text{Some } (T \ x) \ \text{else } \text{None})$

**abbreviation**  $\text{emb } Q \ t \equiv \text{emb}' Q (\lambda-. t)$

**lemma** *emb-eq-Some-conv*:  $\bigwedge T. \text{emb}' Q \ T \ x = \text{Some } t' \longleftrightarrow (t' = T \ x \wedge Q \ x)$   
 ⟨proof⟩

**lemma** *emb-le-Some-conv*:  $\bigwedge T. \text{Some } t' \leq \text{emb}' Q \ T \ x \longleftrightarrow (t' \leq T \ x \wedge Q \ x)$   
 ⟨proof⟩

**lemma** *SPEC-REST-emb'-conv*:  $\text{SPEC } P \ t = \text{REST } (\text{emb}' P \ t)$   
 ⟨proof⟩

**lemma** *SPECT-ub*:  $T \leq T' \implies \text{SPECT } (\text{emb}' M' T) \leq \text{SPECT } (\text{emb}' M' T')$   
 ⟨proof⟩

Select some value with given property, or *None* if there is none.

**definition** *SELECT* ::  $('a \Rightarrow \text{bool}) \Rightarrow \text{enat} \Rightarrow 'a \ \text{option} \ \text{nrest}$

**where**  $\text{SELECT } P \ tf \equiv \text{if } \exists x. P \ x \ \text{then } \text{REST } (\text{emb } (\lambda r. \text{case } r \ \text{of } \text{Some } p \Rightarrow P \ p \mid \text{None} \Rightarrow \text{False}) \ tf)$   
            $\text{else } \text{REST } [\text{None} \mapsto tf]$

**lemma** *inresT-SELECT-Some*:  $\text{inresT } (\text{SELECT } Q \ tt) \ (\text{Some } x) \ t' \longleftrightarrow (Q \ x \wedge (t' \leq tt))$   
 ⟨proof⟩

**lemma** *inresT-SELECT-None*:  $\text{inresT } (\text{SELECT } Q \ tt) \ \text{None } t' \longleftrightarrow (\neg(\exists x. Q \ x) \wedge (t' \leq tt))$   
 ⟨proof⟩

**lemma** *inresT-SELECT*[*refine-pw-simps*]:  
 $inresT (SELECT Q tt) x t' \longleftrightarrow ((case\ x\ of\ None \Rightarrow \neg(\exists x. Q\ x) \mid Some\ x \Rightarrow Q\ x) \wedge (t' \leq tt))$   
 ⟨*proof*⟩

**lemma** *nofailT-SELECT*[*refine-pw-simps*]: *nofailT* (*SELECT* *Q* *tt*)  
 ⟨*proof*⟩

**lemma** *s1*: *SELECT* *P* *T*  $\leq$  (*SELECT* *P* *T'*)  $\longleftrightarrow$  *T*  $\leq$  *T'*  
 ⟨*proof*⟩

**lemma** *s2*: *SELECT* *P* *T*  $\leq$  (*SELECT* *P'* *T*)  $\longleftrightarrow$  (  
 (*Ex* *P'*  $\longrightarrow$  *Ex* *P*)  $\wedge$  ( $\forall x. P\ x \longrightarrow P'\ x$ )  
 ⟨*proof*⟩

**lemma** *SELECT-refine*:  
 assumes  $\bigwedge x'. P'\ x' \Longrightarrow \exists x. P\ x$   
 assumes  $\bigwedge x. P\ x \Longrightarrow P'\ x$   
 assumes *T*  $\leq$  *T'*  
 shows *SELECT* *P* *T*  $\leq$  (*SELECT* *P'* *T'*)  
 ⟨*proof*⟩

## 7 RECT

**definition** *mono2* *B*  $\equiv$  *flatf-mono-ge* *B*  $\wedge$  *mono* *B*

**lemma** *trimonoD-flatf-ge*: *mono2* *B*  $\Longrightarrow$  *flatf-mono-ge* *B*  
 ⟨*proof*⟩

**lemma** *trimonoD-mono*: *mono2* *B*  $\Longrightarrow$  *mono* *B*  
 ⟨*proof*⟩

**definition** *RECT* *B* *x* =  
 (*if* *mono2* *B* *then* (*gfp* *B* *x*) *else* (*top*::'*a*::*complete-lattice*))

**lemma** *RECT-flat-gfp-def*: *RECT* *B* *x* =  
 (*if* *mono2* *B* *then* (*flatf-gfp* *B* *x*) *else* (*top*::'*a*::*complete-lattice*))  
 ⟨*proof*⟩

**lemma** *RECT-unfold*:  $\llbracket mono2\ B \rrbracket \Longrightarrow RECT\ B = B\ (RECT\ B)$   
 ⟨*proof*⟩

**definition** *whileT* :: ('*a*  $\Rightarrow$  *bool*)  $\Rightarrow$  ('*a*  $\Rightarrow$  '*a* *nrest*)  $\Rightarrow$  '*a*  $\Rightarrow$  '*a* *nrest* **where**  
*whileT* *b* *c* = *RECT* ( $\lambda whileT\ s. (if\ b\ s\ then\ bindT\ (c\ s)\ whileT\ else\ RETURN\ s)$ )

**definition**  $whileIET :: ('a \Rightarrow bool) \Rightarrow ('a \Rightarrow nat) \Rightarrow ('a \Rightarrow bool) \Rightarrow ('a \Rightarrow 'a nrest) \Rightarrow 'a \Rightarrow 'a nrest$  **where**

$\bigwedge E c. whileIET I E b c = whileT b c$

**definition**  $whileTI :: ('a \Rightarrow enat option) \Rightarrow (('a \times 'a) set) \Rightarrow ('a \Rightarrow bool) \Rightarrow ('a \Rightarrow 'a nrest) \Rightarrow 'a \Rightarrow 'a nrest$  **where**

$whileTI I R b c s = whileT b c s$

**lemma**  $trimonoI[refine-mono]$ :

$\llbracket flatf-mono-ge B; mono B \rrbracket \Longrightarrow mono2 B$   
 $\langle proof \rangle$

**lemma**  $mono-fun-transform[refine-mono]$ :  $(\bigwedge f g x. (\bigwedge x. f x \leq g x) \Longrightarrow B f x \leq B g x) \Longrightarrow mono B$

$\langle proof \rangle$

**lemma**  $whileT-unfold$ :  $whileT b c = (\lambda s. (if b s then bindT (c s) (whileT b c) else RETURN s))$

$\langle proof \rangle$

**lemma**  $RECT-mono[refine-mono]$ :

**assumes**  $[simp]$ :  $mono2 B'$   
**assumes**  $LE$ :  $\bigwedge F x. (B' F x) \leq (B F x)$   
**shows**  $(RECT B' x) \leq (RECT B x)$   
 $\langle proof \rangle$

**lemma**  $whileT-mono$ :

**assumes**  $\bigwedge x. b x \Longrightarrow c x \leq c' x$   
**shows**  $(whileT b c x) \leq (whileT b c' x)$   
 $\langle proof \rangle$

**lemma**  $wf-fp-induct$ :

**assumes**  $fp$ :  $\bigwedge x. f x = B (f) x$   
**assumes**  $wf$ :  $wf R$   
**assumes**  $\bigwedge x D. \llbracket \bigwedge y. (y, x) \in R \Longrightarrow P y (D y) \rrbracket \Longrightarrow P x (B D x)$   
**shows**  $P x (f x)$   
 $\langle proof \rangle$

**lemma**  $RECT-wf-induct-aux$ :

**assumes**  $wf$ :  $wf R$   
**assumes**  $mono$ :  $mono2 B$   
**assumes**  $(\bigwedge x D. (\bigwedge y. (y, x) \in R \Longrightarrow P y (D y)) \Longrightarrow P x (B D x))$   
**shows**  $P x (RECT B x)$   
 $\langle proof \rangle$

**theorem**  $RECT-wf-induct[consumes 1]$ :

**assumes**  $RECT B x = r$

**assumes**  $wf\ R$   
**and**  $mono2\ B$   
**and**  $\bigwedge x\ D\ r. (\bigwedge y\ r. (y, x) \in R \implies D\ y = r \implies P\ y\ r) \implies B\ D\ x = r \implies P\ x\ r$   
**shows**  $P\ x\ r$   
  
 $\langle proof \rangle$

**definition** *monadic-WHILEIT*  $I\ b\ f\ s \equiv do\ \{$   
 $\quad RECT\ (\lambda D\ s. do\ \{$   
 $\quad\quad ASSERT\ (I\ s);$   
 $\quad\quad bv \leftarrow b\ s;$   
 $\quad\quad if\ bv\ then\ do\ \{$   
 $\quad\quad\quad s \leftarrow f\ s;$   
 $\quad\quad\quad D\ s$   
 $\quad\quad\quad \} else\ do\ \{RETURN\ s\}$   
 $\quad\quad \} \} s$   
 $\}$

## 8 Generalized Weakest Precondition

### 8.1 mm

**definition**  $mm :: ( 'a \Rightarrow enat\ option) \Rightarrow ( 'a \Rightarrow enat\ option) \Rightarrow ( 'a \Rightarrow enat\ option) \mathbf{where}$

$mm\ R\ m = (\lambda x. (case\ m\ x\ of\ None \Rightarrow Some\ \infty$   
 $\quad\quad\quad | Some\ mt \Rightarrow$   
 $\quad\quad\quad (case\ R\ x\ of\ None \Rightarrow None\ | Some\ rt \Rightarrow (if\ rt < mt$   
 $then\ None\ else\ Some\ (rt - mt))))))$

**lemma** *mm-mono*:  $Q1\ x \leq Q2\ x \implies mm\ Q1\ M\ x \leq mm\ Q2\ M\ x$   
 $\langle proof \rangle$

**lemma** *mm-antimono*:  $M1\ x \geq M2\ x \implies mm\ Q\ M1\ x \leq mm\ Q\ M2\ x$   
 $\langle proof \rangle$

**lemma** *mm-continous*:  $mm\ (\lambda x. Inf\ \{u. \exists y. u = f\ y\ x\})\ m\ x = Inf\ \{u. \exists y. u = mm\ (f\ y)\ m\ x\}$   
 $\langle proof \rangle$

**definition**  $mm2 :: ( enat\ option) \Rightarrow ( enat\ option) \Rightarrow ( enat\ option) \mathbf{where}$   
 $mm2\ r\ m = (case\ m\ of\ None \Rightarrow Some\ \infty$   
 $\quad\quad\quad | Some\ mt \Rightarrow$   
 $\quad\quad\quad (case\ r\ of\ None \Rightarrow None\ | Some\ rt \Rightarrow (if\ rt < mt\ then$   
 $None\ else\ Some\ (rt - mt))))$

**lemma** *mm-alt*:  $mm\ R\ m\ x = mm2\ (R\ x)\ (m\ x)\ \langle proof \rangle$

**lemma** *mm2-None[simp]*:  $mm2\ q\ None = Some\ \infty\ \langle proof \rangle$

**lemma** *mm2-Some0[simp]*:  $mm2\ q\ (Some\ 0) = q\ \langle proof \rangle$

**lemma** *mm2-antimono*:  $x \leq y \implies mm2\ q\ x \geq mm2\ q\ y$   
 $\langle proof \rangle$

**lemma** *mm2-contiuous2*:  
**assumes**  $\forall x \in X. t \leq mm2\ q\ x$  **shows**  $t \leq mm2\ q\ (Sup\ X)$   
 $\langle proof \rangle$

**lemma** *ft*:  $(a::enat) - b = \infty \implies a = \infty$   
 $\langle proof \rangle$

**lemma** *mm-inf1*:  $mm\ R\ m\ x = Some\ \infty \implies m\ x = None \vee R\ x = Some\ \infty$   
 $\langle proof \rangle$

**lemma** *mm-inf2*:  $m\ x = None \implies mm\ R\ m\ x = Some\ \infty$   
 $\langle proof \rangle$

**lemma** *mm-inf3*:  $R\ x = Some\ \infty \implies mm\ R\ m\ x = Some\ \infty$   
 $\langle proof \rangle$

**lemma** *mm-inf*:  $mm\ R\ m\ x = Some\ \infty \longleftrightarrow m\ x = None \vee R\ x = Some\ \infty$   
 $\langle proof \rangle$

**lemma** *InfQ-E*:  $Inf\ Q = Some\ t \implies None \notin Q$   
 $\langle proof \rangle$

**lemma** *InfQ-iff*:  $(\exists t' \geq enat\ t. Inf\ Q = Some\ t') \longleftrightarrow None \notin Q \wedge Inf\ (Option.these\ Q) \geq t$   
 $\langle proof \rangle$

**lemma** *mm2-fst-None[simp]*:  $mm2\ None\ q = (case\ q\ of\ None \Rightarrow Some\ \infty\ | - \Rightarrow None)$   
 $\langle proof \rangle$

**lemma** *mm2-auxXX1*:  $Some\ t \leq mm2\ (Q\ x)\ (Some\ t') \implies Some\ t' \leq mm2\ (Q\ x)\ (Some\ t)$   
 $\langle proof \rangle$

## 8.2 mii

**definition** *mii* ::  $('a \Rightarrow enat\ option) \Rightarrow 'a\ nrest \Rightarrow 'a \Rightarrow enat\ option$  **where**  
 $mii\ Q\ f\ M\ x = (case\ M\ of\ FAIL\ i \Rightarrow None$

|  $REST\ Mf \Rightarrow (mm\ Qf\ Mf)\ x$ )

**lemma** *mii-alt*:  $mii\ Qf\ M\ x = (case\ M\ of\ FAILi \Rightarrow None$   
|  $REST\ Mf \Rightarrow (mm2\ (Qf\ x)\ (Mf\ x))$ ) )  
⟨proof⟩

**lemma** *mii-continuous*:  $mii\ (\lambda x. Inf\ \{f\ y\ x\ |y. True\})\ m\ x = Inf\ \{mii\ (\%x. f\ y\ x)$   
 $m\ x\ |y. True\}$   
⟨proof⟩

**lemma** *mii-continuous2*:  $(mii\ Q\ (Sup\ \{F\ x\ t1\ |x\ t1. P\ x\ t1\})\ x \geq t) = (\forall y\ t1. P$   
 $y\ t1 \longrightarrow mii\ Q\ (F\ y\ t1)\ x \geq t)$   
⟨proof⟩

**lemma** *mii-inf*:  $mii\ Qf\ M\ x = Some\ \infty \longleftrightarrow (\exists\ Mf. M = SPECT\ Mf \wedge (Mf\ x =$   
 $None \vee Qf\ x = Some\ \infty))$   
⟨proof⟩

**lemma** *miiFailt*:  $mii\ Q\ FAILT\ x = None$   
⟨proof⟩

### 8.3 lst - latest starting time

**definition** *lst* :: 'a nrest  $\Rightarrow$  ('a  $\Rightarrow$  enat option)  $\Rightarrow$  enat option  
**where**  $lst\ M\ Qf = Inf\ \{mii\ Qf\ M\ x\ |x. True\}$

**lemma** *T-alt-def*:  $lst\ M\ Qf = Inf\ ( (mii\ Qf\ M)\ 'UNIV )$   
⟨proof⟩

**lemma** *T-pw*:  $lst\ M\ Q \geq t \longleftrightarrow (\forall x. mii\ Q\ M\ x \geq t)$   
⟨proof⟩

**lemma** *T-specifies-I*:  
**assumes**  $lst\ m\ Q \geq Some\ 0$  **shows**  $m \leq SPECT\ Q$   
⟨proof⟩

**lemma** *T-specifies-rev*:  
**assumes**  $m \leq SPECT\ Q$  **shows**  $lst\ m\ Q \geq Some\ 0$   
⟨proof⟩

**lemma** *T-specifies*:  $lst\ m\ Q \geq Some\ 0 = (m \leq SPECT\ Q)$   
⟨proof⟩

**lemma** *pointwise-lesseq*:  
**fixes**  $x :: 'a::order$   
**shows**  $(\forall t. x \geq t \longrightarrow x' \geq t) \Longrightarrow x \leq x'$   
⟨proof⟩

## 8.4 pointwise reasoning about lst via nres3

**definition** *nres3* where  $nres3\ Q\ M\ x\ t \longleftrightarrow mii\ Q\ M\ x \geq t$

**lemma** *pw-T-le*:

**assumes**  $\bigwedge t. (\forall x. nres3\ Q\ M\ x\ t) \implies (\forall x. nres3\ Q'\ M'\ x\ t)$   
**shows**  $lst\ M\ Q \leq lst\ M'\ Q'$   
 $\langle proof \rangle$

**lemma** **assumes**  $\bigwedge t. (\forall x. nres3\ Q\ M\ x\ t) = (\forall x. nres3\ Q'\ M'\ x\ t)$   
**shows** *pw-T-eq-iff*:  $lst\ M\ Q = lst\ M'\ Q'$   
 $\langle proof \rangle$

**lemma** **assumes**  $\bigwedge t. (\forall x. nres3\ Q\ M\ x\ t) \implies (\forall x. nres3\ Q'\ M'\ x\ t)$   
 $\bigwedge t. (\forall x. nres3\ Q'\ M'\ x\ t) \implies (\forall x. nres3\ Q\ M\ x\ t)$   
**shows** *pw-T-eqI*:  $lst\ M\ Q = lst\ M'\ Q'$   
 $\langle proof \rangle$

**lemma** *lem*:  $\forall t1. M\ y = Some\ t1 \longrightarrow t \leq mii\ Q\ (SPECT\ (map-option\ ((+)\ t1)\ o\ x2))\ x \implies f\ y = SPECT\ x2 \implies t \leq mii\ (\lambda y. mii\ Q\ (f\ y)\ x)\ (SPECT\ M)\ y$   
 $\langle proof \rangle$

**lemma** *diff-diff-add-enat*:  $a - (b+c) = a - b - (c::enat)$   
 $\langle proof \rangle$

**lemma** *lem2*:  $t \leq mii\ (\lambda y. mii\ Q\ (f\ y)\ x)\ (SPECT\ M)\ y \implies M\ y = Some\ t1 \implies f\ y = SPECT\ fF \implies t \leq mii\ Q\ (SPECT\ (map-option\ ((+)\ t1)\ o\ fF))\ x$   
 $\langle proof \rangle$

**lemma** *fixes*  $m :: 'b\ nrest$

**shows** *mii-bindT*:  $(t \leq mii\ Q\ (bindT\ m\ f)\ x) \longleftrightarrow (\forall y. t \leq mii\ (\lambda y. mii\ Q\ (f\ y)\ x)\ m\ y)$   
 $\langle proof \rangle$

**lemma** *nres3-bindT*:  $(\forall x. nres3\ Q\ (bindT\ m\ f)\ x\ t) = (\forall y. nres3\ (\lambda y. lst\ (f\ y)\ Q)\ m\ y\ t)$   
 $\langle proof \rangle$

## 8.5 rules for lst

**lemma** *T-bindT*:  $lst\ (bindT\ M\ f)\ Q = lst\ M\ (\lambda y. lst\ (f\ y)\ Q)$   
 $\langle proof \rangle$

**lemma** *T-REST*:  $lst\ (REST\ [x \mapsto t])\ Q = mm2\ (Q\ x)\ (Some\ t)$   
 $\langle proof \rangle$

**lemma** *T-RETURN*:  $lst (RETURN x) Q = Q x$   
 ⟨proof⟩

**lemma** *T-SELECT*:

**assumes**

$\forall x. \neg P x \implies Some\ tt \leq lst (SPECT [None \mapsto tf]) Q$

**and**  $p: (\bigwedge x. P x \implies Some\ tt \leq lst (SPECT [Some\ x \mapsto tf]) Q)$

**shows**  $Some\ tt \leq lst (SELECT P tf) Q$

⟨proof⟩

## 9 consequence rules

**lemma** *aux1*:  $Some\ t \leq mm2\ Q (Some\ t') \iff Some\ (t+t') \leq Q$   
 ⟨proof⟩

**lemma** *aux1a*:  $(\forall x\ t''. Q' x = Some\ t'' \implies (Q x) \geq Some\ (t + t'))$   
 $= (\forall x. mm2 (Q x) (Q' x) \geq Some\ t)$   
 ⟨proof⟩

**lemma** *T-conseq4*:

**assumes**

$lst\ f\ Q' \geq Some\ t'$

$\bigwedge x\ t''. M. Q' x = Some\ t'' \implies (Q x) \geq Some\ ((t - t') + t'')$

**shows**  $lst\ f\ Q \geq Some\ t$

⟨proof⟩

**lemma** *T-conseq6*:

**assumes**

$lst\ f\ Q' \geq Some\ t$

$\bigwedge x\ t''. M. f = SPECT\ M \implies M\ x \neq None \implies Q' x = Some\ t'' \implies (Q x) \geq Some\ (t'')$

**shows**  $lst\ f\ Q \geq Some\ t$

⟨proof⟩

**lemma** *T-conseq6'*:

**assumes**

$lst\ f\ Q' \geq Some\ t$

$\bigwedge x\ t''. M. f = SPECT\ M \implies M\ x \neq None \implies (Q x) \geq Q' x$

**shows**  $lst\ f\ Q \geq Some\ t$

⟨proof⟩

**lemma** *T-conseq5*:

**assumes**

$lst\ f\ Q' \geq Some\ t'$

$\bigwedge x\ t''. M. f = SPECT\ M \implies M\ x \neq None \implies Q' x = Some\ t'' \implies (Q x) \geq Some\ ((t - t') + t'')$

**shows**  $lst\ f\ Q \geq Some\ t$   
 $\langle proof \rangle$

**lemma** *T-conseq3*:

**assumes**

$lst\ f\ Q' \geq Some\ t'$

$\bigwedge x. mm2\ (Q\ x)\ (Q'\ x) \geq Some\ (t - t')$

**shows**  $lst\ f\ Q \geq Some\ t$

$\langle proof \rangle$

## 10 Experimental Hoare reasoning

**named-theorems** *vcg-rules*

**method** *vcg* **uses**  $rls = ((rule\ rls\ vcg\ rules[THEN\ T-conseq4] \mid clarsimp\ simp:\ emb\ eq\ Some\ conv\ emb\ le\ Some\ conv\ T\ bindT\ T\ RETURNT)+)$

**experiment**

**begin**

**definition** *P* **where**  $P\ f\ g = bindT\ f\ (\lambda x. bindT\ g\ (\lambda y. RETURNT\ (x+(y::nat))))$

**lemma** **assumes**

$f\ spec[vcg\ rules]: lst\ f\ (emb'\ (\lambda x. x > 2)\ (enat\ o\ ((*)\ 2)) ) \geq Some\ 0$

**and**

$g\ spec[vcg\ rules]: lst\ g\ (emb'\ (\lambda x. x > 2)\ (enat)) \geq Some\ 0$

**shows**  $lst\ (P\ f\ g)\ (emb'\ (\lambda x. x > 5)\ (enat\ o\ (*)\ 3)) \geq Some\ 0$

$\langle proof \rangle$

**end**

## 11 VCG

**named-theorems** *vcg-simp-rules*

**lemmas**  $[vcg\ simp\ rules] = T\ RETURNT$

**lemma** *TbindT-I*:  $Some\ t \leq lst\ M\ (\lambda y. lst\ (f\ y)\ Q) \implies Some\ t \leq lst\ (M\ \gg\ f)$

$Q$

$\langle proof \rangle$

**method** *vcg'* **uses**  $rls = ((rule\ rls\ TbindT\ I\ vcg\ rules[THEN\ T-conseq6] \mid clarsimp\ split:\ if\ splits\ simp:\ vcg\ simp\ rules)+)$

**lemma** *mm2-refl*:  $A < \infty \implies mm2\ (Some\ A)\ (Some\ A) = Some\ 0$

$\langle proof \rangle$

**definition** *mm3* **where**

$mm3\ t\ A = (case\ A\ of\ None \Rightarrow None \mid Some\ t' \Rightarrow if\ t' \leq t\ then\ Some\ (enat\ (t-t'))\ else\ None)$

**lemma** *mm3-same[simp]*:  $mm3\ t0\ (Some\ t0) = Some\ 0\ \langle proof \rangle$

**lemma** *mm3-Some-conv*:  $(mm3\ t0\ A = Some\ t) = (\exists\ t'.\ A = Some\ t' \wedge t0 \geq t' \wedge t=t0-t')\ \langle proof \rangle$

**lemma** *mm3-None[simp]*:  $mm3\ t0\ None = None\ \langle proof \rangle$

**lemma** *T-FAILT[simp]*:  $lst\ FAILT\ Q = None\ \langle proof \rangle$

**definition** *progress*  $m \equiv \forall\ s'\ M.\ m = SPECT\ M \longrightarrow M\ s' \neq None \longrightarrow M\ s' > Some\ 0$

**lemma** *progressD*:  $progress\ m \Longrightarrow m=SPECT\ M \Longrightarrow M\ s' \neq None \Longrightarrow M\ s' > Some\ 0\ \langle proof \rangle$

**lemma** *progress-FAILT[simp]*:  $progress\ FAILT\ \langle proof \rangle$

## 11.1 Progress rules

**named-theorems** *progress-rules*

**lemma** *progress-SELECT-iff*:  $progress\ (SELECT\ P\ t) \longleftrightarrow t > 0\ \langle proof \rangle$

**lemmas** [*progress-rules*] = *progress-SELECT-iff*[*THEN iffD2*]

**lemma** *progress-REST-iff*:  $progress\ (REST\ [x \mapsto t]) \longleftrightarrow t > 0\ \langle proof \rangle$

**lemmas** [*progress-rules*] = *progress-REST-iff*[*THEN iffD2*]

**lemma** *progress-ASSERT-bind*[*progress-rules*]:  $\llbracket \Phi \Longrightarrow progress\ (f\ ()) \rrbracket \Longrightarrow progress\ (ASSERT\ \Phi \ggg f)\ \langle proof \rangle$

**lemma** *progress-SPECT-emb*[*progress-rules*]:  $t > 0 \Longrightarrow progress\ (SPECT\ (emb\ P\ t))\ \langle proof \rangle$

**lemma** *Sup-Some*:  $Sup\ (S::enat\ option\ set) = Some\ e \Longrightarrow \exists\ x \in S.\ (\exists\ i.\ x = Some\ i)\ \langle proof \rangle$

**lemma** *progress-bind*[*progress-rules*]: **assumes**  $progress\ m \vee (\forall\ x.\ progress\ (f\ x))$   
**shows**  $progress\ (m \ggg f)\ \langle proof \rangle$

**lemma** *mm2SomeleSome-conv*:  $mm2 (Qf) (Some\ t) \geq Some\ 0 \iff Qf \geq Some\ t$   
 ⟨proof⟩

## 12 rules for whileT

**lemma**

**assumes** *whileT*  $b\ c\ s = r$   
**assumes** *IS*[*vcg-rules*]:  $\bigwedge s\ t'.\ I\ s = Some\ t' \implies b\ s$   
 $\implies\ lst\ (c\ s)\ (\lambda s'.\ if\ (s',s) \in R\ then\ I\ s'\ else\ None) \geq Some\ t'$

**assumes**  $I\ s = Some\ t$   
**assumes** *wf*:  $wf\ R$   
**shows** *whileT-rule''*:  $lst\ r\ (\lambda x.\ if\ b\ x\ then\ None\ else\ I\ x) \geq Some\ t$   
 ⟨proof⟩

**lemma**

**fixes**  $I :: 'a \Rightarrow nat\ option$   
**assumes** *whileT*  $b\ c\ s0 = r$   
**assumes** *progress*:  $\bigwedge s.\ progress\ (c\ s)$   
**assumes** *IS*[*vcg-rules*]:  $\bigwedge s\ t\ t'.\ I\ s = Some\ t \implies b\ s \implies$   
 $lst\ (c\ s)\ (\lambda s'.\ mm3\ t\ (I\ s')) \geq Some\ 0$

**assumes** [*simp*]:  $I\ s0 = Some\ t0$

**shows** *whileT-rule'''*:  $lst\ r\ (\lambda x.\ if\ b\ x\ then\ None\ else\ mm3\ t0\ (I\ x)) \geq Some\ 0$   
 ⟨proof⟩

**lemma** *whileIET-rule*[*THEN T-conseq6, vcg-rules*]:

**fixes**  $E$   
**assumes**  
 ( $\bigwedge s\ t\ t'.$   
 ( $if\ I\ s\ then\ Some\ (E\ s)\ else\ None) = Some\ t \implies$   
 $b\ s \implies Some\ 0 \leq lst\ (C\ s)\ (\lambda s'.\ mm3\ t\ (if\ I\ s'\ then\ Some\ (E\ s')\ else\ None))$ )  
 $\bigwedge s.\ progress\ (C\ s)$   
 $I\ s0$

**shows**  $Some\ 0 \leq lst\ (whileIET\ I\ E\ b\ C\ s0)\ (\lambda x.\ if\ b\ x\ then\ None\ else\ mm3\ (E\ s0)\ (if\ I\ x\ then\ Some\ (E\ x)\ else\ None))$   
 ⟨proof⟩

**lemma** *transf*:

**assumes**  $I\ s \implies b\ s \implies Some\ 0 \leq lst\ (C\ s)\ (\lambda s'.\ mm3\ (E\ s)\ (if\ I\ s'\ then\ Some\ (E\ s')\ else\ None))$

**shows**  
 ( $if\ I\ s\ then\ Some\ (E\ s)\ else\ None) = Some\ t \implies$   
 $b\ s \implies Some\ 0 \leq lst\ (C\ s)\ (\lambda s'.\ mm3\ t\ (if\ I\ s'\ then\ Some\ (E\ s')\ else\ None))$   
 ⟨proof⟩

**lemma** *whileIET-rule'*:

**fixes**  $E$

**assumes**

$(\bigwedge s t'. I s \implies b s \implies \text{Some } 0 \leq \text{lst } (C s) (\lambda s'. \text{mm3 } (E s) (\text{if } I s' \text{ then } \text{Some } (E s') \text{ else None})))$

$\bigwedge s. \text{progress } (C s)$

$I s0$

**shows**  $\text{Some } 0 \leq \text{lst } (\text{whileIET } I E b C s0) (\lambda x. \text{if } b x \text{ then None else mm3 } (E s0) (\text{if } I x \text{ then Some } (E x) \text{ else None}))$

*<proof>*

## 13 some Monadic Refinement Automation

*<ML>*

**end**

**theory** *DataRefinement*

**imports** *NREST*

**begin**

### 13.1 Data Refinement

**lemma**  $\{(1,2),(2,4)\} \text{''}\{1,2\}=\{2,4\} \text{''}$  *<proof>*

**definition** *conc-fun* ( $\Downarrow$ ) **where**

$\text{conc-fun } R m \equiv \text{case } m \text{ of } \text{FAIL}i \Rightarrow \text{FAILT} \mid \text{REST } X \Rightarrow \text{REST } (\lambda c. \text{Sup } \{X a \mid a. (c,a) \in R\})$

**definition** *abs-fun* ( $\Uparrow$ ) **where**

$\text{abs-fun } R m \equiv \text{case } m \text{ of } \text{FAIL}i \Rightarrow \text{FAILT}$   
 $\mid \text{REST } X \Rightarrow \text{if } \text{dom } X \subseteq \text{Domain } R \text{ then } \text{REST } (\lambda a. \text{Sup } \{X c \mid c. (c,a) \in R\})$   
 $\text{else } \text{FAILT}$

**lemma**

$\text{conc-fun-FAIL}[\text{simp}]: \Downarrow R \text{ FAILT} = \text{FAILT}$  **and**

$\text{conc-fun-RES}: \Downarrow R (\text{REST } X) = \text{REST } (\lambda c. \text{Sup } \{X a \mid a. (c,a) \in R\})$

*<proof>*

**lemma** *nrest-Rel-mono*:  $A \leq B \implies \Downarrow R A \leq \Downarrow R B$

*<proof>*

#### 13.1.1 Examples

**definition** *Rset* **where**  $Rset = \{ (c,a) \mid c a. \text{set } c = a \}$

**lemma**  $\text{RETURN}T [1,2,3] \leq \Downarrow Rset (\text{RETURN}T \{1,2,3\})$

*<proof>*

**lemma**  $RETURNNT [1,2,3] \leq \Downarrow Rset (RETURNNT \{1,2,3\})$   
 ⟨proof⟩

**definition** *Reven* **where**  $Reven = \{ (c,a) \mid c \text{ a. even } c = a \}$

**lemma**  $RETURNNT 3 \leq \Downarrow Reven (RETURNNT False)$   
 ⟨proof⟩

**lemma**  $m \leq \Downarrow Id m$   
 ⟨proof⟩

**lemma**  $m \leq \Downarrow \{\} m \longleftrightarrow (m = FAILT \vee m = SPECT bot)$   
 ⟨proof⟩

**lemma** *abs-fun-simps[simp]*:  
 $\Uparrow R FAILT = FAILT$   
 $dom X \subseteq Domain R \implies \Uparrow R (REST X) = REST (\lambda a. Sup \{X c \mid c. (c,a) \in R\})$   
 $\neg(dom X \subseteq Domain R) \implies \Uparrow R (REST X) = FAILT$   
 ⟨proof⟩

**context** *fixes R assumes SV: single-valued R begin*

**lemma**  
**fixes**  $m :: 'b \Rightarrow enat \text{ option}$   
**shows**  
 $Sup\text{-sv}: (c, a') \in R \implies Sup \{m a \mid a. (c,a) \in R\} = m a'$   
 ⟨proof⟩

**lemma** *indomD*:  $M c = Some y \implies dom M \subseteq Domain R \implies (\exists a. (c,a) \in R)$   
 ⟨proof⟩

**lemma** *conc-abs-swap*:  $m' \leq \Downarrow R m \longleftrightarrow \Uparrow R m' \leq m$   
 ⟨proof⟩

**lemma**  
**fixes**  $m :: 'b \Rightarrow enat \text{ option}$   
**shows**  
 $Inf\text{-sv}: (c, a') \in R \implies Inf \{m a \mid a. (c,a) \in R\} = m a'$   
 ⟨proof⟩

**lemma** *ac-galois: galois-connection*  $(\Uparrow R) (\Downarrow R)$   
 ⟨proof⟩

**lemma**  
**fixes**  $m :: 'b \Rightarrow enat \text{ option}$   
**shows** *Sup-some-svD*:  $Sup \{m a \mid a. (c, a) \in R\} = Some t' \implies (\exists a. (c,a) \in R \wedge$

$m\ a = \text{Some } t'$   
 $\langle \text{proof} \rangle$

**end**

**lemma** *pw-abs-nofail*[*refine-pw-simps*]:  
 $\text{nofailT } (\uparrow R\ M) \longleftrightarrow (\text{nofailT } M \wedge (\forall x. (\exists t. \text{inresT } M\ x\ t) \longrightarrow x \in \text{Domain } R))$   
**(is**  $?l \longleftrightarrow ?r$   
 $\langle \text{proof} \rangle$

**lemma** *pw-conc-nofail*[*refine-pw-simps*]:  
 $\text{nofailT } (\Downarrow R\ S) = \text{nofailT } S$   
 $\langle \text{proof} \rangle$

**lemma** *single-valued*  $A \implies \text{single-valued } B \implies \text{single-valued } (A\ O\ B)$   
 $\langle \text{proof} \rangle$

**lemma** *Sup-enatoption-less2*:  $\text{Sup } X = \text{Some } \infty \implies (\exists x. \text{Some } x \in X \wedge \text{enat } t < x)$   
 $\langle \text{proof} \rangle$

**lemma** *pw-conc-inres*[*refine-pw-simps*]:  
 $\text{inresT } (\Downarrow R\ S')\ s\ t = (\text{nofailT } S' \longrightarrow ((\exists s'. (s, s') \in R \wedge \text{inresT } S'\ s'\ t)))$  **(is**  $?l \longleftrightarrow ?r$   
 $\langle \text{proof} \rangle$

**lemma** *sv-the*:  $\text{single-valued } R \implies (c, a) \in R \implies (\text{THE } a. (c, a) \in R) = a$   
 $\langle \text{proof} \rangle$

**lemma**  
*conc-fun-RES-sv*:  
**assumes**  $SV$ : *single-valued*  $R$   
**shows**  $\Downarrow R\ (\text{REST } X) = \text{REST } (\lambda c. \text{if } c \in \text{Domain } R \text{ then } X\ (\text{THE } a. (c, a) \in R)$   
*else None* )  
 $\langle \text{proof} \rangle$

**lemma** *conc-fun-mono*[*simp, intro!*]:  
**shows**  $\text{mono } (\Downarrow R)$   
 $\langle \text{proof} \rangle$

**lemma** *conc-fun-R-mono*:  
**assumes**  $R \subseteq R'$   
**shows**  $\Downarrow R\ M \leq \Downarrow R'\ M$   
 $\langle \text{proof} \rangle$

**lemma** *SupSup-2*:  $Sup \{m \ a \ |a. (c, a) \in R \ O \ S\} = Sup \{m \ a \ |a \ b. (b,a) \in S \wedge (c,b) \in R \}$   
 ⟨*proof*⟩

**lemma**

**fixes**  $m :: 'a \Rightarrow enat \ option$

**shows** *SupSup*:  $Sup \{Sup \{m \ aa \ |aa. P \ a \ aa \ c\} \ |a. Q \ a \ c\} = Sup \{m \ aa \ |a \ aa. P \ a \ aa \ c \wedge Q \ a \ c\}$   
 ⟨*proof*⟩

**lemma**

**fixes**  $m :: 'a \Rightarrow enat \ option$

**shows**

*SupSup-1*:  $Sup \{Sup \{m \ aa \ |aa. (a, aa) \in S\} \ |a. (c, a) \in R\} = Sup \{m \ aa \ |a \ aa. (a,aa) \in S \wedge (c,a) \in R\}$   
 ⟨*proof*⟩

**lemma** *conc-fun-chain*:

**shows**  $\Downarrow R (\Downarrow S \ M) = \Downarrow (R \ O \ S) \ M$

⟨*proof*⟩

**lemma** *conc-fun-chain-sv*:

**assumes** *SVR*: *single-valued R* **and** *SVS*: *single-valued S*

**shows**  $\Downarrow R (\Downarrow S \ M) = \Downarrow (R \ O \ S) \ M$

⟨*proof*⟩

**lemma** *conc-Id[simp]*:  $\Downarrow Id = id$

⟨*proof*⟩

**lemma** *conc-fun-fail-iff[simp]*:

$\Downarrow R \ S = FAILT \longleftrightarrow S = FAILT$

$FAILT = \Downarrow R \ S \longleftrightarrow S = FAILT$

⟨*proof*⟩

**lemma** *conc-trans[trans]*:

**assumes**  $A: C \leq \Downarrow R \ B$  **and**  $B: B \leq \Downarrow R' \ A$

**shows**  $C \leq \Downarrow R (\Downarrow R' \ A)$

⟨*proof*⟩

**lemma** *conc-trans-sv*:

**assumes** *SV*: *single-valued R* *single-valued R'*

**and**  $A: C \leq \Downarrow R \ B$  **and**  $B: B \leq \Downarrow R' \ A$

**shows**  $C \leq \Downarrow R (\Downarrow R' \ A)$

⟨*proof*⟩

WARNING: The order of the single statements is important here!

**lemma** *conc-trans-additional*[trans]:

**assumes** *single-valued R*

**shows**

$$\begin{aligned} \bigwedge A B C. A \leq \Downarrow R B &\implies B \leq C \implies A \leq \Downarrow R C \\ \bigwedge A B C. A \leq \Downarrow Id B &\implies B \leq \Downarrow R C \implies A \leq \Downarrow R C \\ \bigwedge A B C. A \leq \Downarrow R B &\implies B \leq \Downarrow Id C \implies A \leq \Downarrow R C \end{aligned}$$

$$\begin{aligned} \bigwedge A B C. A \leq \Downarrow Id B &\implies B \leq \Downarrow Id C \implies A \leq C \\ \bigwedge A B C. A \leq \Downarrow Id B &\implies B \leq C \implies A \leq C \\ \bigwedge A B C. A \leq B &\implies B \leq \Downarrow Id C \implies A \leq C \end{aligned}$$

*<proof>*

**lemma** *RETURN-refine*:

**assumes**  $(x, x') \in R$

**shows**  $RETURN x \leq \Downarrow R (RETURN x')$

*<proof>*

**lemma** *bindT-refine'*:

**fixes**  $R' :: ('a \times 'b)$  set **and**  $R :: ('c \times 'd)$  set

**assumes**  $R1: M \leq \Downarrow R' M'$

**assumes**  $R2: \bigwedge x x' t. \llbracket (x, x') \in R'; \text{inresT } M x t; \text{inresT } M' x' t; \text{nofailT } M; \text{nofailT } M' \rrbracket$

$\implies f x \leq \Downarrow R (f' x')$

**shows**  $\text{bindT } M (\lambda x. f x) \leq \Downarrow R (\text{bindT } M' (\lambda x'. f' x'))$

*<proof>*

**lemma** *bindT-refine*:

**fixes**  $R' :: ('a \times 'b)$  set **and**  $R :: ('c \times 'd)$  set

**assumes**  $R1: M \leq \Downarrow R' M'$

**assumes**  $R2: \bigwedge x x'. \llbracket (x, x') \in R' \rrbracket$

$\implies f x \leq \Downarrow R (f' x')$

**shows**  $\text{bindT } M (\lambda x. f x) \leq \Downarrow R (\text{bind } M' (\lambda x'. f' x'))$

*<proof>*

## 13.2 WHILET refine

**lemma** *RECT-refine*:

**assumes**  $M: \text{mono2 } \text{body}$

**assumes**  $R0: (x, x') \in R$

**assumes**  $RS: \bigwedge f f' x x'. \llbracket \bigwedge x x'. (x, x') \in R \implies f x \leq \Downarrow S (f' x'); (x, x') \in R \rrbracket$   
 $\implies \text{body } f x \leq \Downarrow S (\text{body}' f' x')$

**shows**  $RECT (\lambda f x. \text{body } f x) x \leq \Downarrow S (RECT (\lambda f' x'. \text{body}' f' x') x')$

*<proof>*

**lemma** *WHILET-refine*:

**assumes**  $R0: (x, x') \in R$

**assumes**  $COND-REF: \bigwedge x x'. \llbracket (x, x') \in R \rrbracket \implies b x = b' x'$

**assumes**  $STEP-REF:$

$\bigwedge x x'. \llbracket (x, x') \in R; b x; b' x' \rrbracket \implies f x \leq \Downarrow R (f' x')$

**shows**  $\text{whileT } b \ f \ x \leq \Downarrow R \ (\text{whileT } b' \ f' \ x')$   
 $\langle \text{proof} \rangle$

**lemma** *SPECT-refines-conc-fun'*:  
**assumes**  $a: \bigwedge m \ c. \ M = \text{SPECT } m$   
 $\implies c \in \text{dom } n \implies (\exists a. (c,a) \in R \wedge n \ c \leq m \ a)$   
**shows**  $\text{SPECT } n \leq \Downarrow R \ M$   
 $\langle \text{proof} \rangle$

**lemma** *SPECT-refines-conc-fun*:  
**assumes**  $a: \bigwedge m \ c. (\exists a. (c,a) \in R \wedge n \ c \leq m \ a)$   
**shows**  $\text{SPECT } n \leq \Downarrow R \ (\text{SPECT } m)$   
 $\langle \text{proof} \rangle$

**lemma** *conc-fun-br*:  $\Downarrow (br \ \alpha \ I1) \ (\text{SPECT } (emb \ I2 \ t))$   
 $= (\text{SPECT } (emb \ (\lambda x. I1 \ x \wedge I2 \ (\alpha \ x)) \ t))$   
 $\langle \text{proof} \rangle$

**end**  
**theory** *RefineMonadicVCG*  
**imports** *NREST DataRefinement*  
*Case-Labeling.Case-Labeling*  
**begin**

— Might look similar to *repeat-new* from *HOL–Eisbach.Eisbach*, however the placement of the ? is different, in particular that means this method is allowed to failed  
**method** *repeat-all-new* **methods**  $m = (m; \text{repeat-all-new } \langle m \rangle) ?$

**lemma** *R-intro*:  $A \leq \Downarrow Id \ B \implies A \leq B \ \langle \text{proof} \rangle$

### 13.3 ASSERT

**lemma** *le-R-ASSERTI*:  $(\Phi \implies M \leq \Downarrow R \ M') \implies M \leq \Downarrow R \ (\text{ASSERT } \Phi \gg (\lambda-. M'))$   
 $\langle \text{proof} \rangle$

**lemma** *T-ASSERT[vcg-simp-rules]*:  $\text{Some } t \leq \text{lst } (\text{ASSERT } \Phi) \ Q \longleftrightarrow \text{Some } t \leq Q \ () \wedge \Phi$   
 $\langle \text{proof} \rangle$

**lemma** *T-ASSERT-I*:  $(\Phi \implies \text{Some } t \leq Q \ ()) \implies \Phi \implies \text{Some } t \leq \text{lst } (\text{ASSERT } \Phi) \ Q$   
 $\langle \text{proof} \rangle$

**lemma** *T-RESTemb-iff*:  $\text{Some } t' \leq \text{lst } (\text{REST } (emb' \ P \ t)) \ Q \longleftrightarrow (\forall x. P \ x \longrightarrow \text{Some } (t' + t \ x) \leq Q \ x)$   
 $\langle \text{proof} \rangle$

**lemma** *T-RESTemb*:  $(\bigwedge x. P x \implies \text{Some } (t' + t x) \leq Q x)$   
 $\implies \text{Some } t' \leq \text{lst } (\text{REST } (\text{emb}' P t)) Q$   
 $\langle \text{proof} \rangle$

**lemma** *T-SPEC*:  $(\bigwedge x. P x \implies \text{Some } (t' + t x) \leq Q x)$   
 $\implies \text{Some } t' \leq \text{lst } (\text{SPEC } P t) Q$   
 $\langle \text{proof} \rangle$

**lemma** *T-SPECT-I*:  $(\text{Some } (t' + t) \leq Q x)$   
 $\implies \text{Some } t' \leq \text{lst } (\text{SPECT } [x \mapsto t]) Q$   
 $\langle \text{proof} \rangle$

**lemma** *mm2-map-option*:  
**assumes**  $\text{Some } (t'+t) \leq \text{mm2 } (Q x) (x2 x)$   
**shows**  $\text{Some } t' \leq \text{mm2 } (Q x) (\text{map-option } ((+) t) (x2 x))$   
 $\langle \text{proof} \rangle$

**lemma** *T-consume*:  $(\text{Some } (t' + t) \leq \text{lst } M Q)$   
 $\implies \text{Some } t' \leq \text{lst } (\text{consume } M t) Q$   
 $\langle \text{proof} \rangle$

**definition** *valid t Q M* =  $(\text{Some } t \leq \text{lst } M Q)$

### 13.4 VCG splitter

$\langle ML \rangle$

### 13.5 mm3 and emb

**lemma** *Some-eq-mm3-Some-conv[vcg-simp-rules]*:  $\text{Some } t = \text{mm3 } t' (\text{Some } t'')$   
 $\longleftrightarrow (t'' \leq t' \wedge t = \text{enat } (t' - t''))$   
 $\langle \text{proof} \rangle$

**lemma** *Some-eq-mm3-Some-conv'*:  $\text{mm3 } t' (\text{Some } t'') = \text{Some } t \longleftrightarrow (t'' \leq t' \wedge t = \text{enat } (t' - t''))$   
 $\langle \text{proof} \rangle$

**lemma** *Some-le-emb'-conv[vcg-simp-rules]*:  $\text{Some } t \leq \text{emb}' Q \text{ft } x \longleftrightarrow Q x \wedge t \leq \text{ft } x$   
 $\langle \text{proof} \rangle$

**lemma** *Some-eq-emb'-conv[vcg-simp-rules]*:  $\text{emb}' Q \text{tf } s = \text{Some } t \longleftrightarrow (Q s \wedge t = \text{tf } s)$   
 $\langle \text{proof} \rangle$

## 13.6 Setup Labeled VCG

**context**

**begin**

**interpretation** *Labeling-Syntax*  $\langle \text{proof} \rangle$

**lemma** *LCondRule*:

**fixes**  $IC CT$  **defines**  $CT' \equiv ("cond", IC, []) \# CT$

**assumes**  $b \implies C \langle \text{Suc } IC, ("the", IC, []) \# ("cond", IC, []) \# CT, OC1: \text{valid } t Q c1 \rangle$

**and**  $\sim b \implies C \langle \text{Suc } OC1, ("els", \text{Suc } OC1, []) \# ("cond", IC, []) \# CT, OC: \text{valid } t Q c2 \rangle$

**shows**  $C \langle IC, CT, OC: \text{valid } t Q \text{ (if } b \text{ then } c1 \text{ else } c2) \rangle$

$\langle \text{proof} \rangle$

**lemma** *LouterCondRule*:

**fixes**  $IC CT$  **defines**  $CT' \equiv ("cond2", IC, []) \# CT$

**assumes**  $b \implies C \langle \text{Suc } IC, ("the", IC, []) \# ("cond2", IC, []) \# CT, OC1: t \leq A \rangle$

**and**  $\sim b \implies C \langle \text{Suc } OC1, ("els", \text{Suc } OC1, []) \# ("cond2", IC, []) \# CT, OC: t \leq B \rangle$

**shows**  $C \langle IC, CT, OC: t \leq \text{(if } b \text{ then } A \text{ else } B) \rangle$

$\langle \text{proof} \rangle$

**lemma** *While*:

**assumes**  $I s0 (\bigwedge s. I s \implies b s \implies \text{Some } 0 \leq \text{lst } (C s) (\lambda s'. \text{mm3 } (E s) \text{ (if } I s' \text{ then } \text{Some } (E s') \text{ else } \text{None})))$

$(\bigwedge s. \text{progress } (C s))$

$(\bigwedge x. \neg b x \implies I x \implies (E x) \leq (E s0) \implies \text{Some } (t + \text{enat } ((E s0) - E x)) \leq Q x)$

**shows**  $\text{Some } t \leq \text{lst } (\text{whileIET } I E b C s0) Q$

$\langle \text{proof} \rangle$

**definition** *monadic-WHILE*  $b f s \equiv \text{do } \{$

$\text{RECT } (\lambda D s. \text{do } \{$

$bv \leftarrow b s;$

$\text{if } bv \text{ then do } \{$

$s \leftarrow f s;$

$D s$

$\} \text{ else do } \{ \text{RETURN } s \}$

$\} \} s$

$\}$

**lemma** *monadic-WHILE-mono*:

**assumes**  $\bigwedge x. bm x \leq bm' x$

**and**  $\bigwedge x t. \text{nofailT } (bm' x) \implies \text{inresT } (bm x) \text{ True } t \implies c x \leq c' x$

**shows**  $(\text{monadic-WHILE } bm c x) \leq (\text{monadic-WHILE } bm' c' x)$

$\langle \text{proof} \rangle$

**lemma**  $z: \text{inresT } A x t \implies A \leq B \implies \text{inresT } B x t$

*<proof>*

**lemma** *monadic-WHILE-mono'*:

**assumes**

$\bigwedge x. bm\ x \leq bm'\ x$

**and**  $\bigwedge x\ t. nofailT\ (bm'\ x) \implies inresT\ (bm'\ x)\ True\ t \implies c\ x \leq c'\ x$

**shows**  $(monadic-WHILE\ bm\ c\ x) \leq (monadic-WHILE\ bm'\ c'\ x)$

*<proof>*

**lemma** *monadic-WHILE-refine*:

**assumes**

$(x, x') \in R$

$\bigwedge x\ x'. (x, x') \in R \implies bm\ x \leq \Downarrow Id\ (bm'\ x')$

**and**  $\bigwedge x\ x'\ t. (x, x') \in R \implies nofailT\ (bm'\ x') \implies inresT\ (bm'\ x')\ True\ t \implies c\ x \leq \Downarrow R\ (c'\ x')$

**shows**  $(monadic-WHILE\ bm\ c\ x) \leq \Downarrow R\ (monadic-WHILE\ bm'\ c'\ x')$

*<proof>*

**lemma** *monadic-WHILE-aux*:  $monadic-WHILE\ b\ f\ s = monadic-WHILEIT\ (\lambda. True)\ b\ f\ s$

*<proof>*

**lemma**  $lst\ (c\ x)\ Q = Some\ t \implies Some\ t \leq lst\ (c\ x)\ Q'$

*<proof>*

**lemma** *TbindT-I2*:  $tt \leq lst\ M\ (\lambda y. lst\ (f\ y)\ Q) \implies tt \leq lst\ (M \ggg f)\ Q$

*<proof>*

**lemma** *T-conseq7*:

**assumes**

$lst\ f\ Q' \geq tt$

$\bigwedge x\ t''\ M. f = SPECT\ M \implies M\ x \neq None \implies Q'\ x = Some\ t'' \implies (Q\ x) \geq Some\ (t'')$

**shows**  $lst\ f\ Q \geq tt$

*<proof>*

**lemma** *monadic-WHILE-ruleaaa''*:

**assumes**  $monadic-WHILE\ bm\ c\ s = r$

**assumes**  $IS[vcg-rules]: \bigwedge s.$

$lst\ (bm\ s) (\lambda b. \text{if } b \text{ then } lst\ (c\ s) (\lambda s'. \text{if } (s',s) \in R \text{ then } I\ s' \text{ else } None) \text{ else } Q\ s) \geq I\ s$

**assumes**  $wf: wf\ R$

**shows**  $lst\ r\ Q \geq I\ s$

*<proof>*

**lemma** *monadic-WHILE-rule''*:

**assumes**  $monadic-WHILE\ bm\ c\ s = r$

**assumes**  $IS[vcg-rules]: \bigwedge s\ t'. I\ s = Some\ t'$

$\implies lst\ (bm\ s) (\lambda b. \text{if } b \text{ then } lst\ (c\ s) (\lambda s'. \text{if } (s',s) \in R \text{ then } I\ s' \text{ else } None) \text{ else } Q\ s) \geq I\ s$

$None)else Q s) \geq Some t'$

**assumes**  $I s = Some t$   
**assumes**  $wf: wf R$   
**shows**  $lst r Q \geq Some t$   
 $\langle proof \rangle$

**lemma** *whileT-rule''*:

**fixes**  $I :: 'a \Rightarrow nat option$   
**assumes**  $whileT b c s0 = r$   
**assumes**  $progress: \bigwedge s. progress (c s)$   
**assumes**  $IS[vcg-rules]: \bigwedge s t t'. I s = Some t \implies b s \implies$   
 $lst (c s) (\lambda s'. mm3 t (I s')) \geq Some 0$

**assumes**  $[simp]: I s0 = Some t0$

**shows**  $lst r (\lambda x. if b x then None else mm3 t0 (I x)) \geq Some 0$   
 $\langle proof \rangle$

**fun** *Someplus where*

$Someplus None - = None$   
 $| Someplus - None = None$   
 $| Someplus (Some a) (Some b) = Some (a+b)$

**lemma**  $l: \sim (a::enat) < b \longleftrightarrow a \geq b \langle proof \rangle$

**lemma**  $kk: a \geq b \implies (b::enat) + (a - b) = a$   
 $\langle proof \rangle$

**lemma** *Tea*:  $Someplus A B = Some t \longleftrightarrow (\exists a b. A = Some a \wedge B = Some b \wedge t = a + b)$   
 $\langle proof \rangle$

**lemma** *TTT-Some-nofailT*:  $lst c Q = Some l \implies c \neq FAILT$   
 $\langle proof \rangle$

**lemma** *GRR*: **assumes**  $lst (SPECT Mf) Q = Some l$   
**shows**  $Mf x = None \vee (Q x \neq None \wedge (Q x) \geq Mf x)$   
 $\langle proof \rangle$

**lemma** *Someplus-None*:  $Someplus A B = None \longleftrightarrow (A = None \vee B = None)$   
 $\langle proof \rangle$

**lemma** *Somemm3*:  $A \geq B \implies mm3 A (Some B) = Some (A - B)$   
 $\langle proof \rangle$

**lemma** *newWhile-rule*: **assumes** *monadic-WHILE*  $bm c s0 = r$   
**and step**:  $\bigwedge s. I s \implies$   
 $Some 0 \leq lst (bm s) (\lambda b. if b$

then  $lst (c s) (\lambda s'. (if I s' \wedge (E s' \leq E s) then Some (enat (E s - E s')) else None))$   
 else  $mm2 (Q s) (Someplus (Some t) (mm3 (E s0) (Some (E s))))$  )

**and**  $progress: \bigwedge s. progress (c s)$

**and**  $I0: I s0$   
**shows**  $Some t \leq lst r Q$   
 $\langle proof \rangle$

**definition** *monadic-WHILEIE* **where**  
 $monadic-WHILEIE I E bm c s = monadic-WHILE bm c s$

**definition**  $G b d = (if b then Some d else None)$

**definition**  $H Qs t Es0 Es = mm2 Qs (Someplus (Some t) (mm3 (Es0) (Some (Es))))$

**lemma** *neueWhile-rule'*:

**fixes**  $s0 :: 'a$  **and**  $I :: 'a \Rightarrow bool$  **and**  $E :: 'a \Rightarrow nat$   
**assumes**  
 $step: (\bigwedge s. I s \Longrightarrow Some 0 \leq lst (bm s) (\lambda b. if b then lst (c s) (\lambda s'. if I s' \wedge E s' \leq E s then Some (enat (E s - E s')) else None) else mm2 (Q s) (Someplus (Some t) (mm3 (E s0) (Some (E s))))))$   
**and**  $progress: \bigwedge s. progress (c s)$   
**and**  $i: I s0$   
**shows**  $Some t \leq lst (monadic-WHILEIE I E bm c s0) Q$   
 $\langle proof \rangle$

**lemma** *neueWhile-rule''*:

**fixes**  $s0 :: 'a$  **and**  $I :: 'a \Rightarrow bool$  **and**  $E :: 'a \Rightarrow nat$   
**assumes**  
 $step: (\bigwedge s. I s \Longrightarrow Some 0 \leq lst (bm s) (\lambda b. if b then lst (c s) (\lambda s'. G (I s' \wedge E s' \leq E s) (enat (E s - E s')) else H (Q s) t (E s0) (E s))))$   
**and**  $progress: \bigwedge s. progress (c s)$   
**and**  $i: I s0$   
**shows**  $Some t \leq lst (monadic-WHILEIE I E bm c s0) Q$   
 $\langle proof \rangle$

**lemma** *LmonWhileRule*:

**fixes**  $IC CT$   
**assumes**  $V\langle\langle "precondition", IC, [] \rangle, \langle "monwhile", IC, [] \rangle \# CT: I s0\rangle$   
**and**  $\bigwedge s. I s \Longrightarrow C\langle Suc IC, \langle "invariant", Suc IC, [] \rangle \# \langle "monwhile", IC, [] \rangle \# CT, OC: valid 0 (\lambda b. if b then lst (C s) (\lambda s'. if I s' \wedge E s' \leq E s then Some (enat (E s - E s')) else None) else mm2 (Q s) (Someplus (Some t) (mm3 (E s0) (Some (E s)))))) (bm s)\rangle$   
**and**  $\bigwedge s. V\langle\langle "progress", IC, [] \rangle, \langle "monwhile", IC, [] \rangle \# CT: progress (C s)\rangle$   
**shows**  $C\langle IC, CT, OC: valid t Q (monadic-WHILEIE I E bm C s0)\rangle$   
 $\langle proof \rangle$

**lemma** *LWhileRule*:

**fixes** *IC CT*  
**assumes**  $V\langle\langle\text{"precondition"}, IC, []\rangle, \langle\text{"while"}, IC, []\rangle \# CT: I s0\rangle$   
**and**  $\bigwedge s. I s \implies b s \implies C\langle\text{Suc } IC, \langle\text{"invariant"}, \text{Suc } IC, []\rangle \# \langle\text{"while"}, IC, []\rangle \# CT, OC1: \text{valid } 0 (\lambda s'. \text{mm3 } (E s) (\text{if } I s' \text{ then Some } (E s') \text{ else None})) (C s)\rangle$   
**and**  $\bigwedge s. V\langle\langle\text{"progress"}, IC, []\rangle, \langle\text{"while"}, IC, []\rangle \# CT: \text{progress } (C s)\rangle$   
**and**  $\bigwedge x. \neg b x \implies I x \implies (E x) \leq (E s0) \implies C\langle\text{Suc } OC1, \langle\text{"postcondition"}, IC, []\rangle \# \langle\text{"while"}, IC, []\rangle \# CT, OC: \text{Some } (t + \text{enat } ((E s0) - E x)) \leq Q x\rangle$   
**shows**  $C\langle IC, CT, OC: \text{valid } t Q (\text{whileIET } I E b C s0)\rangle$   
 $\langle\text{proof}\rangle$

**lemma** *validD*:  $\text{valid } t Q M \implies \text{Some } t \leq \text{lst } M Q \langle\text{proof}\rangle$

**lemma** *LABELs-to-concl*:

$C\langle IC, CT, OC: \text{True}\rangle \implies C\langle IC, CT, OC: P\rangle \implies P$   
 $V\langle x, ct: \text{True}\rangle \implies V\langle x, ct: P\rangle \implies P$   
 $\langle\text{proof}\rangle$

**lemma** *LASSERTRule*:

**assumes**  $V\langle\langle\text{"ASSERT"}, IC, []\rangle, CT: \Phi\rangle$   
 $C\langle\text{Suc } IC, CT, OC: \text{valid } t Q (\text{RETURN } ())\rangle$   
**shows**  $C\langle IC, CT, OC: \text{valid } t Q (\text{ASSERT } \Phi)\rangle$   
 $\langle\text{proof}\rangle$

**lemma** *LbindTRule*:

**assumes**  $C\langle IC, CT, OC: \text{valid } t (\lambda y. \text{lst } (f y) Q) m\rangle$   
**shows**  $C\langle IC, CT, OC: \text{valid } t Q (\text{bindT } m f)\rangle$   
 $\langle\text{proof}\rangle$

**lemma** *LRETURNTRule*:

**assumes**  $C\langle IC, CT, OC: \text{Some } t \leq Q x\rangle$   
**shows**  $C\langle IC, CT, OC: \text{valid } t Q (\text{RETURN } x)\rangle$   
 $\langle\text{proof}\rangle$

**lemma** *LSELECTRule*:

**fixes** *IC CT* **defines**  $CT' \equiv \langle\text{"cond"}, IC, []\rangle \# CT$   
**assumes**  $\bigwedge x. P x \implies C\langle\text{Suc } IC, \langle\text{"Some"}, IC, []\rangle \# \langle\text{"SELECT"}, IC, []\rangle \# CT, OC1: \text{valid } (t+T) Q (\text{RETURN } (\text{Some } x))\rangle$   
**and**  $\forall x. \neg P x \implies C\langle\text{Suc } OC1, \langle\text{"None"}, \text{Suc } OC1, []\rangle \# \langle\text{"SELECT"}, IC, []\rangle \# CT, OC: \text{valid } (t+T) Q (\text{RETURN } \text{None})\rangle$   
**shows**  $C\langle IC, CT, OC: \text{valid } t Q (\text{SELECT } P T)\rangle$   
 $\langle\text{proof}\rangle$

**lemma** *LRESTembRule*:

**assumes**  $\bigwedge x. P x \implies C\langle IC, CT, OC: \text{Some } (t + T x) \leq Q x\rangle$   
**shows**  $C\langle IC, CT, OC: \text{valid } t Q (\text{REST } (\text{emb } P T))\rangle$   
 $\langle\text{proof}\rangle$

**lemma** *LRESTsingleRule*:

**assumes**  $C\langle IC, CT, OC: \text{Some } (t + t') \leq Q \ x \rangle$   
**shows**  $C\langle IC, CT, OC: \text{valid } t \ Q \ (\text{REST } [x \mapsto t']) \rangle$   
*<proof>*

**lemma** *LTTTinRule*:

**assumes**  $C\langle IC, CT, OC: \text{valid } t \ Q \ M \rangle$   
**shows**  $C\langle IC, CT, OC: \text{Some } t \leq \text{lst } M \ Q \rangle$   
*<proof>*

**lemma** *LfinaltimeRule*:

**assumes**  $V\langle ("time", IC, []), CT: t \leq t' \rangle$   
**shows**  $C\langle IC, CT, IC: \text{Some } t \leq \text{Some } t' \rangle$   
*<proof>*

**lemma** *LfinalfuncRule*:

**assumes**  $V\langle ("func", IC, []), CT: \text{False} \rangle$   
**shows**  $C\langle IC, CT, IC: \text{Some } t \leq \text{None} \rangle$   
*<proof>*

**lemma** *LembRule*:

**assumes**  $V\langle ("time", IC, []), CT: t \leq T \ x \rangle$   
**and**  $V\langle ("func", IC, []), CT: P \ x \rangle$   
**shows**  $C\langle IC, CT, IC: \text{Some } t \leq \text{emb}' P \ T \ x \rangle$   
*<proof>*

**lemma** *Lmm3Rule*:

**assumes**  $V\langle ("time", IC, []), CT: Va' \leq Va \wedge t \leq \text{enat } (Va - Va') \rangle$   
**and**  $V\langle ("func", IC, []), CT: b \rangle$   
**shows**  $C\langle IC, CT, IC: \text{Some } t \leq \text{mm}3 \ Va \ (\text{if } b \ \text{then } \text{Some } Va' \ \text{else } \text{None}) \rangle$   
*<proof>*

**lemma** *LinjectRule*:

**assumes**  $\text{Some } t \leq \text{lst } A \ Q \implies \text{Some } t \leq \text{lst } B \ Q$   
 $C\langle IC, CT, OC: \text{valid } t \ Q \ A \rangle$   
**shows**  $C\langle IC, CT, OC: \text{valid } t \ Q \ B \rangle$   
*<proof>*

**lemma** *Linject2Rule*:

**assumes**  $A = B$   
 $C\langle IC, CT, OC: \text{valid } t \ Q \ A \rangle$   
**shows**  $C\langle IC, CT, OC: \text{valid } t \ Q \ B \rangle$   
*<proof>*

**method** *labeled-VCG-init* = ((*rule T-specifies-I validD*)<sup>+</sup>), *rule Initial-Label*

**method** *labeled-VCG-step* **uses** *rules* = (*rule rules*[*symmetric*], *THEN Linject2Rule*)

*LTTTinRule LbindTRule*  
*LembRule Lmm3Rule*  
*LRETURNTRule LASSERTRule LCondRule LSELECTRule*  
*LRESTsingleRule LRESTembRule*  
*LouterCondRule*  
*LfinaltimeRule LfinalfuncRule*  
*LmonWhileRule LWhileRule) | vcg-split-case*

**method** *labeled-VCG uses rules = labeled-VCG-init, repeat-all-new*  $\langle$  *labeled-VCG-step rules: rules*  $\rangle$

**method** *casified-VCG uses rules = labeled-VCG rules: rules, casify*

### 13.7 Examples, labeled vcg

**lemma** *do* {  $x \leftarrow \text{SELECT } P \ T;$   
     *(case*  $x$  *of*  $\text{None} \Rightarrow \text{RETURNNT } (1::\text{nat}) \mid \text{Some } t \Rightarrow \text{RETURNNT } (2::\text{nat}))$   
   }  $\leq \text{SPECT } (\text{emb } Q \ T')$   
 $\langle$  *proof*  $\rangle$

**lemma**  
   **assumes**  $b \ c$   
   **shows** *do* {  $\text{ASSERT } b;$   
      $\text{ASSERT } c;$   
      $\text{RETURNNT } 1 \}$   $\leq \text{SPECT } (\text{emb } (\lambda x. x > (0::\text{nat})) \ 1)$   
 $\langle$  *proof*  $\rangle$

**lemma** *do* {  
   *(if*  $b$  *then*  $\text{RETURNNT } (1::\text{nat})$  *else*  $\text{RETURNNT } 2)$   
   }  $\leq \text{SPECT } (\text{emb } (\lambda x. x > 0) \ 1)$   
 $\langle$  *proof*  $\rangle$

**lemma**  
   **assumes**  $b$   
   **shows** *do* {  
      $\text{ASSERT } b;$   
     *(if*  $b$  *then*  $\text{RETURNNT } (1::\text{nat})$  *else*  $\text{RETURNNT } 2)$   
   }  $\leq \text{SPECT } (\text{emb } (\lambda x. x > 0) \ 1)$   
 $\langle$  *proof*  $\rangle$

**end**

**lemma** *SPECT-ub'*:  $T \leq T' \implies \text{SPECT } (\text{emb}' \ M' \ T) \leq \Downarrow \text{Id } (\text{SPECT } (\text{emb}' \ M' \ T'))$   
 $\langle$  *proof*  $\rangle$

**lemma** *REST-single-rule*[*vcg-simp-rules*]:  $\text{Some } t \leq \text{lst } (\text{REST } [x \mapsto t']) \ Q \longleftrightarrow \text{Some } (t+t') \leq (Q \ x)$   
 ⟨proof⟩

### 13.8 progress solver

**method** *progress* **methods** *solver* =  
 (rule *asm-rl*[of *progress* -],  
 (simp *split*: *prod.splits* | *intro* *allI* *impI* *conjI* | *determ* ⟨rule *progress-rules*⟩  
 | rule *disjI1*; *progress* ⟨*solver*⟩; *fail* | rule *disjI2*; *progress* ⟨*solver*⟩; *fail* |  
*solver*)+) []  
**method** *progress'* **methods** *solver* =  
 (rule *asm-rl*[of *progress* -], (*vcg-split-case* | *intro* *allI* *impI* *conjI* | *determ* ⟨rule  
*progress-rules*⟩  
 | rule *disjI1* *disjI2*; *progress'*⟨*solver*⟩ | (*solver*;fail))+) []

**lemma**  
**assumes**  $(\bigwedge s \ t. P \ s = \text{Some } t \implies \exists s'. \text{Some } t \leq Q \ s' \wedge (s, s') \in R)$   
**shows** *SPECT-refine*:  $\text{SPECT } P \leq \Downarrow R \ (\text{SPECT } Q)$   
 ⟨proof⟩

### 13.9 more stuff involving mm3 and emb

**lemma** *Some-le-mm3-Some-conv*[*vcg-simp-rules*]:  $\text{Some } t \leq \text{mm3 } t' \ (\text{Some } t'') \longleftrightarrow (t'' \leq t' \wedge t \leq \text{enat } (t' - t''))$   
 ⟨proof⟩

**lemma** *embtimeI*:  $T \leq T' \implies \text{emb } P \ T \leq \text{emb } P \ T'$  ⟨proof⟩

**lemma** *not-cons-is-Nil-conv*[*simp*]:  $(\forall y \ ys. l \neq y \ \# \ ys) \longleftrightarrow l = []$  ⟨proof⟩

**lemma** *mm3-Some0-eq*[*simp*]:  $\text{mm3 } t \ (\text{Some } 0) = \text{Some } t$   
 ⟨proof⟩

**lemma** *ran-emb'*:  $c \in \text{ran } (\text{emb}' \ Q \ t) \longleftrightarrow (\exists s'. \ Q \ s' \wedge t \ s' = c)$   
 ⟨proof⟩

**lemma** *ran-emb-conv*:  $\text{Ex } Q \implies \text{ran } (\text{emb } Q \ t) = \{t\}$   
 ⟨proof⟩

**lemma** *in-ran-emb-special-case*:  $c \in \text{ran } (\text{emb } Q \ t) \implies c \leq t$   
 ⟨proof⟩

**lemma** *dom-emb'-eq*[*simp*]:  $\text{dom } (\text{emb}' \ Q \ f) = \text{Collect } Q$   
 ⟨proof⟩

**lemma** *emb-le-emb*:  $\text{emb}' \ P \ T \leq \text{emb}' \ P' \ T' \longleftrightarrow (\forall x. P \ x \longrightarrow P' \ x \wedge T \ x \leq T' \ x)$

*<proof>*

## 13.10 VCG for monadic programs

### 13.10.1 old

**declare** *mm3-Some-conv* [*vcg-simp-rules*]

**lemma** *SS*[*vcg-simp-rules*]: *Some t = Some t'  $\longleftrightarrow$  t = t'* *<proof>*

**lemma** *SS'*: *(if b then Some t else None) = Some t'  $\longleftrightarrow$  (b  $\wedge$  t = t')* *<proof>*

**term** *(case s of (a,b)  $\Rightarrow$  M a b)*

**lemma** *case-T*[*vcg-rules*]: *( $\wedge$  a b. s = (a, b)  $\Longrightarrow$  t  $\leq$  lst Q (M a b))  $\Longrightarrow$  t  $\leq$  lst Q*

*(case s of (a,b)  $\Rightarrow$  M a b)*

*<proof>*

### 13.10.2 new setup

**named-theorems** *vcg-rules'*

**lemma** *if-T*[*vcg-rules'*]: *(b  $\Longrightarrow$  t  $\leq$  lst Ma Q)  $\Longrightarrow$  ( $\neg$ b  $\Longrightarrow$  t  $\leq$  lst Mb Q)  $\Longrightarrow$  t  $\leq$  lst (if b then Ma else Mb) Q*

*<proof>*

**lemma** *RETURN-T-I*[*vcg-rules'*]: *t  $\leq$  Q x  $\Longrightarrow$  t  $\leq$  lst (RETURN x) Q*

*<proof>*

**declare** *T-SPECT-I* [*vcg-rules'*]

**declare** *TbindT-I* [*vcg-rules'*]

**declare** *T-RESTEMb* [*vcg-rules'*]

**declare** *T-ASSERT-I* [*vcg-rules'*]

**declare** *While* [*vcg-rules'*]

**named-theorems** *vcg-simps'*

**declare** *option.case* [*vcg-simps'*]

**declare** *neueWhile-rule''* [*vcg-rules'*]

**method** *vcg'-step* **methods** *solver* **uses** *rules =*

*(intro rules vcg-rules' | vcg-split-case | (progress simp;fail) | (solver; fail))*

**method** *vcg'* **methods** *solver* **uses** *rules = repeat-all-new <vcg'-step solver rules: rules>*

**declare** *T-SELECT* [*vcg-rules'*]

— No proof

**lemma**  $\wedge$ *c. do* { *c  $\leftarrow$  RETURN None;*

*(case-option (RETURN (1::nat)) ( $\lambda$ p. RETURN (2::nat))) c*

*}  $\leq$  SPECT (emb ( $\lambda$ x. x>(0::nat)) 1)*

*<proof>*

**thm** *option.case*

### 13.11 setup for *refine-vcg*

**lemma** *If-refine[refine]*:  $b = b' \implies$   
 $(b \implies b' \implies S1 \leq \Downarrow R S1') \implies$   
 $(\neg b \implies \neg b' \implies S2 \leq \Downarrow R S2') \implies (if\ b\ then\ S1\ else\ S2) \leq \Downarrow R (if\ b'\ then$   
 $S1'\ else\ S2')$   
*<proof>*

**lemma** *Case-option-refine[refine]*:  $(x, x') \in \langle S \rangle option-rel \implies$   
 $(\bigwedge y\ y'. (y, y') \in S \implies S2\ y \leq \Downarrow R (S2'\ y')) \implies S1 \leq \Downarrow R S1'$   
 $\implies (case\ x\ of\ None \Rightarrow S1 \mid Some\ y \Rightarrow S2\ y) \leq \Downarrow R (case\ x'\ of\ None \Rightarrow S1' \mid$   
 $Some\ y' \Rightarrow S2'\ y')$   
*<proof>*

**lemma** *conc-fun-Id-refined[refine0]*:  $\bigwedge S. S \leq \Downarrow Id\ S$  *<proof>*

**lemma** *conc-fun-ASSERT-refine[refine0]*:  $\Phi \implies (\Phi \implies S \leq \Downarrow R S') \implies ASSERT$   
 $\Phi \gg= (\lambda-. S) \leq \Downarrow R S'$   
*<proof>*

**declare** *le-R-ASSERTI [refine0]*

**declare** *bindT-refine [refine]*

**declare** *WHILET-refine [refine]*

— Check name

**lemma** *SPECT-refine-vcg-cons[refine-vcg-cons]*:  $m \leq SPECT\ \Phi \implies (\bigwedge x. \Phi\ x \leq$   
 $\Psi\ x) \implies m \leq SPECT\ \Psi$   
*<proof>*

**end**

**theory** *SepLogic-Misc*

**imports** *DataRefinement*

**begin**

### 13.12 Relators

**definition** *nrest-rel where*

*nres-rel-def-internal*:  $nrest-rel\ R \equiv \{(c, a). c \leq \Downarrow R\ a\}$

**lemma** *nrest-rel-def*:  $\langle R \rangle nrest-rel \equiv \{(c, a). c \leq \Downarrow R\ a\}$   
*<proof>*

**lemma** *nrest-relD*:  $(c, a) \in \langle R \rangle nrest-rel \implies c \leq \Downarrow R\ a$  *<proof>*

**lemma** *nrest-relI*:  $c \leq \Downarrow R\ a \implies (c, a) \in \langle R \rangle nrest-rel$  *<proof>*

**lemma** *nrest-rel-comp*:  $\langle A \rangle nrest-rel\ O\ \langle B \rangle nrest-rel = \langle A\ O\ B \rangle nrest-rel$   
*<proof>*

**lemma** *pw-nrest-rel-iff*:  $(a,b) \in \langle A \rangle nrest\text{-rel} \iff nofailT (\Downarrow A b) \longrightarrow nofailT a \wedge (\forall x t. inresT a x t \longrightarrow inresT (\Downarrow A b) x t)$   
 $\langle proof \rangle$

**lemma** *param-FAIL*[*param*]:  $(FAILT, FAILT) \in \langle R \rangle nrest\text{-rel}$   
 $\langle proof \rangle$

**lemma** *param-RETURN*[*param*]:  
 $(RETURNNT, RETURNNT) \in R \rightarrow \langle R \rangle nrest\text{-rel}$   
 $\langle proof \rangle$

**lemma** *param-bind*[*param*]:  
 $(bindT, bindT) \in \langle Ra \rangle nrest\text{-rel} \rightarrow (Ra \rightarrow \langle Rb \rangle nrest\text{-rel}) \rightarrow \langle Rb \rangle nrest\text{-rel}$   
 $\langle proof \rangle$

**end**

**theory** *Refine-Foreach*

**imports** *NREST RefineMonadicVCG SepLogic-Misc*

**begin**

A common pattern for loop usage is iteration over the elements of a set. This theory provides the *FOREACH*-combinator, that iterates over each element of a set.

### 13.13 Auxilliary Lemmas

The following lemma is commonly used when reasoning about iterator invariants. It helps converting the set of elements that remain to be iterated over to the set of elements already iterated over.

**lemma** *it-step-insert-iff*:  
 $it \subseteq S \implies x \in it \implies S - (it - \{x\}) = insert\ x\ (S - it)$   $\langle proof \rangle$

### 13.14 Definition

Foreach-loops come in different versions, depending on whether they have an annotated invariant (I), a termination condition (C), and an order (O).

Note that asserting that the set is finite is not necessary to guarantee termination. However, we currently provide only iteration over finite sets, as this also matches the ICF concept of iterators.

**definition** *FOREACH-body*  $f \equiv \lambda(xs, \sigma). do \{$   
 $x \leftarrow RETURNNT(hd\ xs); \sigma' \leftarrow f\ x\ \sigma; RETURNNT(tl\ xs, \sigma')$   
 $\}$

**definition** *FOREACH-cond* **where** *FOREACH-cond*  $c \equiv (\lambda(xs, \sigma). xs \neq [] \wedge c\ \sigma)$

Foreach with continuation condition, order and annotated invariant:

**definition**  $FOREACH_{oci}$  ( $\langle FOREACH_{OC} \cdot \cdot \rangle$ ) **where**  $FOREACH_{oci} R \Phi S c f$   
 $\sigma 0$  *inittime body-time*  $\equiv$  *do* {  
 ASSERT (*finite S*);  
 $xs \leftarrow SPECT$  (*emb* ( $\lambda xs. distinct\ xs \wedge S = set\ xs \wedge sorted-wrt\ R\ xs$ ) *inittime*);  
 $(-, \sigma) \leftarrow whileIET$   
 $(\lambda(it, \sigma). \exists xs'. xs = xs' @ it \wedge \Phi (set\ it)\ \sigma) (\lambda(it, -). length\ it * body-time)$   
 (*FOREACH-cond c*) (*FOREACH-body f*) ( $xs, \sigma 0$ );  
 RETURN  $\sigma$  }

Foreach with continuation condition and annotated invariant:

**definition**  $FOREACH_{ci}$  ( $\langle FOREACH_C \cdot \cdot \rangle$ ) **where**  $FOREACH_{ci} \equiv FOREACH_{oci}$   
 $(\lambda - . True)$

### 13.15 Proof Rules

**lemma**  $FOREACH_{oci}$ -rule:

**assumes** *IP*:

$\bigwedge x\ it\ \sigma. \llbracket c\ \sigma; x \in it; it \subseteq S; I\ it\ \sigma; \forall y \in it - \{x\}. R\ x\ y;$   
 $\forall y \in S - it. R\ y\ x \rrbracket \implies f\ x\ \sigma \leq SPECT\ (emb\ (I\ (it - \{x\})))\ (enat$   
*body-time)*

**assumes** *FIN*: *finite S*

**assumes** *I0*:  $I\ S\ \sigma 0$

**assumes** *II1*:  $\bigwedge \sigma. \llbracket I\ \{\} \sigma \rrbracket \implies P\ \sigma$

**assumes** *II2*:  $\bigwedge it\ \sigma. \llbracket it \neq \{\}; it \subseteq S; I\ it\ \sigma; \neg c\ \sigma;$   
 $\forall x \in it. \forall y \in S - it. R\ y\ x \rrbracket \implies P\ \sigma$

**assumes** *time-ub*: *inittime* + *enat* ( $(card\ S) * body-time$ )  $\leq$  *enat overall-time*

**assumes** *progress-f*[*progress-rules*]:  $\bigwedge a\ b. progress\ (f\ a\ b)$

**shows**  $FOREACH_{oci}\ R\ I\ S\ c\ f\ \sigma 0$  *inittime body-time*  $\leq SPECT\ (emb\ P\ (enat$   
*overall-time))*

*<proof>*

**lemma**  $FOREACH_{ci}$ -rule :

**assumes** *IP*:

$\bigwedge x\ it\ \sigma. \llbracket x \in it; it \subseteq S; I\ it\ \sigma; c\ \sigma \rrbracket \implies f\ x\ \sigma \leq SPECT\ (emb\ (I\ (it - \{x\})))$   
*(enat body-time)*

**assumes** *II1*:  $\bigwedge \sigma. \llbracket I\ \{\} \sigma \rrbracket \implies P\ \sigma$

**assumes** *II2*:  $\bigwedge it\ \sigma. \llbracket it \neq \{\}; it \subseteq S; I\ it\ \sigma; \neg c\ \sigma \rrbracket \implies P\ \sigma$

**assumes** *FIN*: *finite S*

**assumes** *I0*:  $I\ S\ \sigma 0$

**assumes** *progress-f*:  $\bigwedge a\ b. progress\ (f\ a\ b)$

**assumes** *inittime* + *enat* ( $card\ S * body-time$ )  $\leq$  *enat overall-time*

**shows**  $FOREACH_{ci}\ I\ S\ c\ f\ \sigma 0$  *inittime body-time*  $\leq SPECT\ (emb\ P\ (enat$   
*overall-time))*

*<proof>*

We define a relation between distinct lists and sets.

**definition** [*to-relAPP*]: *list-set-rel R*  $\equiv \langle R \rangle list-rel\ O$  *br set distinct*

### 13.16 Nres-Fold with Interruption (nfoldli)

A foreach-loop can be conveniently expressed as an operation that converts the set to a list, followed by folding over the list.

This representation is handy for automatic refinement, as the complex foreach-operation is expressed by two relatively simple operations.

We first define a fold-function in the nrest-monad

**definition** *nfoldli* **where**

$$\begin{aligned} \text{nfoldli } l \ c \ f \ s &= \text{RECT } (\lambda D \ (l, s). \ (\text{case } l \ \text{of} \\ &\quad [] \Rightarrow \text{RETURNNT } s \\ &\quad | \ x\#ls \Rightarrow \text{if } c \ s \ \text{then } do \ \{ \ s \leftarrow f \ x \ s; \ D \ (ls, s) \} \ \text{else } \text{RETURNNT } s \\ &\quad )) \ (l, s) \end{aligned}$$

**lemma** *nfoldli-simps*[*simp*]:

$$\begin{aligned} \text{nfoldli } [] \ c \ f \ s &= \text{RETURNNT } s \\ \text{nfoldli } (x\#ls) \ c \ f \ s &= \\ &\quad (\text{if } c \ s \ \text{then } do \ \{ \ s \leftarrow f \ x \ s; \ \text{nfoldli } ls \ c \ f \ s \} \ \text{else } \text{RETURNNT } s) \\ &\langle \text{proof} \rangle \end{aligned}$$

**lemma** *param-nfoldli*[*param*]:

$$\begin{aligned} \text{shows } (\text{nfoldli}, \text{nfoldli}) &\in \\ &\langle Ra \rangle \text{list-rel} \rightarrow (Rb \rightarrow Id) \rightarrow (Ra \rightarrow Rb \rightarrow \langle Rb \rangle \text{nrest-rel}) \rightarrow Rb \rightarrow \langle Rb \rangle \text{nrest-rel} \\ &\langle \text{proof} \rangle \end{aligned}$$

**lemma** *nfoldli-no-ctd*[*simp*]:  $\neg \text{ctd } s \implies \text{nfoldli } l \ \text{ctd } f \ s = \text{RETURNNT } s$

$\langle \text{proof} \rangle$

**lemma** *nfoldli-append*:  $\text{nfoldli } (l1 @ l2) \ \text{ctd } f \ s = \text{nfoldli } l1 \ \text{ctd } f \ s \gg \text{nfoldli } l2 \ \text{ctd } f$

$\langle \text{proof} \rangle$

**lemma** *nfoldli-assert*:

$$\begin{aligned} \text{assumes } \text{set } l &\subseteq S \\ \text{shows } \text{nfoldli } l \ c \ (\lambda x \ s. \ \text{ASSERT } (x \in S) \gg f \ x \ s) \ s &= \text{nfoldli } l \ c \ f \ s \\ &\langle \text{proof} \rangle \end{aligned}$$

**lemmas** *nfoldli-assert'* = *nfoldli-assert*[*OF order.refl*]

**definition** *nfoldliIE* ::  $( 'd \ \text{list} \Rightarrow 'd \ \text{list} \Rightarrow 'a \Rightarrow \text{bool} ) \Rightarrow \text{nat} \Rightarrow 'd \ \text{list} \Rightarrow ( 'a \Rightarrow \text{bool} ) \Rightarrow ( 'd \Rightarrow 'a \Rightarrow 'a \ \text{nrest} ) \Rightarrow 'a \Rightarrow 'a \ \text{nrest}$  **where**

$$\text{nfoldliIE } I \ E \ l \ c \ f \ s = \text{nfoldli } l \ c \ f \ s$$

**lemma** *nfoldliIE-rule'*:

$$\begin{aligned} \text{assumes } IS: \bigwedge x \ l1 \ l2 \ \sigma. \ [ \ l0 = l1 @ x \# l2; \ I \ l1 \ (x \# l2) \ \sigma; \ c \ \sigma ] &\implies f \ x \ \sigma \leq \text{SPECT} \\ (\text{emb } (I \ (l1 @ [x]) \ l2) \ (\text{enat body-time})) & \\ \text{assumes } \text{FNC}: \bigwedge l1 \ l2 \ \sigma. \ [ \ l0 = l1 @ l2; \ I \ l1 \ l2 \ \sigma; \ \neg c \ \sigma ] &\implies P \ \sigma \\ \text{assumes } \text{FC}: \bigwedge \sigma. \ [ \ I \ l0 \ [] \ \sigma ] &\implies P \ \sigma \end{aligned}$$

**shows**  $lr @ l = l0 \implies I lr l \sigma \implies \text{nfoldliIE } I \text{ body-time } l c f \sigma \leq \text{SPECT } (\text{emb } P (\text{body-time} * \text{length } l))$   
 ⟨proof⟩

**lemma** *nfoldliIE-rule*:

**assumes**  $I0: I \sqsupset l0 \sigma 0$   
**assumes**  $IS: \bigwedge x l1 l2 \sigma. \llbracket l0 = l1 @ x \# l2; I l1 (x \# l2) \sigma; c \sigma \rrbracket \implies f x \sigma \leq \text{SPECT}$   
*(emb (I (l1 @ [x]) l2) (enat body-time))*  
**assumes**  $FNC: \bigwedge l1 l2 \sigma. \llbracket l0 = l1 @ l2; I l1 l2 \sigma; \neg c \sigma \rrbracket \implies P \sigma$   
**assumes**  $FC: \bigwedge \sigma. \llbracket I l0 \sqsupset \sigma \rrbracket \implies P \sigma$   
**assumes**  $T: (\text{body-time} * \text{length } l0) \leq t$   
**shows**  $\text{nfoldliIE } I \text{ body-time } l0 c f \sigma 0 \leq \text{SPECT } (\text{emb } P t)$   
 ⟨proof⟩

**lemma** *nfoldli-refine[refine]*:

**assumes**  $(li, l) \in \langle S \rangle \text{list-rel}$   
**and**  $(si, s) \in R$   
**and**  $CR: (ci, c) \in R \rightarrow \text{bool-rel}$   
**and**  $[\text{refine}]: \bigwedge xi x si s. \llbracket (xi, x) \in S; (si, s) \in R; c s \rrbracket \implies fi xi si \leq \Downarrow R (f x s)$   
**shows**  $\text{nfoldli } li ci fi si \leq \Downarrow R (\text{nfoldli } l c f s)$   
 ⟨proof⟩

**definition**  $\text{nfoldliIE}' I bt l0 f s0 = \text{nfoldliIE } I bt l0 (\lambda-. \text{True}) f s0$

**lemma** *nfoldliIE'-rule*:

**assumes**  
 $\bigwedge x l1 l2 \sigma.$   
 $l0 = l1 @ x \# l2 \implies$   
 $I l1 (x \# l2) \sigma \implies \text{Some } 0 \leq \text{lst } (f x \sigma) (\text{emb } (I (l1 @ [x]) l2) (\text{enat body-time}))$   
 $I \sqsupset l0 \sigma 0$   
 $(\bigwedge \sigma. I l0 \sqsupset \sigma \implies \text{Some } (t + \text{enat } (\text{body-time} * \text{length } l0)) \leq Q \sigma)$   
**shows**  $\text{Some } t \leq \text{lst } (\text{nfoldliIE}' I \text{ body-time } l0 f \sigma 0) Q$   
 ⟨proof⟩

We relate our fold-function to the while-loop that we used in the original definition of the foreach-loop

**lemma** *nfoldli-while*:  $\text{nfoldli } l c f \sigma$

$\leq$   
 $(\text{whileIET } I E$   
 $(\text{FOREACH-cond } c) (\text{FOREACH-body } f) (l, \sigma) \gg=$   
 $(\lambda(-, \sigma). \text{RETURN } \sigma))$

⟨proof⟩

**lemma** *rr*:  $l0 = l1 @ a \implies a \neq [] \implies l0 = l1 @ \text{hd } a \# \text{tl } a$  ⟨proof⟩

**lemma** *nfoldli-rule*:

**assumes**  $I0: I \sqsupset l0 \sigma 0$   
**assumes**  $IS: \bigwedge x l1 l2 \sigma. \llbracket l0 = l1 @ x \# l2; I l1 (x \# l2) \sigma; c \sigma \rrbracket \implies f x \sigma \leq \text{SPECT}$

$(emb (I (l1@[x]) l2) (enat \text{body-time}))$   
**assumes**  $FNC: \bigwedge l1 l2 \sigma. \llbracket l0=l1@[x]; I l1 l2 \sigma; \neg c \sigma \rrbracket \implies P \sigma$   
**assumes**  $FC: \bigwedge \sigma. \llbracket I l0 \rrbracket \sigma; c \sigma \rrbracket \implies P \sigma$   
**assumes**  $progressf: \bigwedge x y. progress (f x y)$   
**shows**  $nfoldli l0 c f \sigma 0 \leq SPECT (emb P (\text{body-time} * length l0))$   
 $\langle proof \rangle$

**definition**  $LIST-FOREACH' \text{tsl} c f \sigma \equiv do \{xs \leftarrow \text{tsl}; nfoldli xs c f \sigma\}$

This constant is a placeholder to be converted to custom operations by pattern rules

**definition**  $it\text{-to-sorted-list} R s \text{to-sorted-list-time}$   
 $\equiv SPECT (emb (\lambda l. distinct l \wedge s = set l \wedge sorted\text{-wrt} R l) \text{to-sorted-list-time})$

**definition**  $LIST-FOREACH \Phi \text{tsl} c f \sigma 0 \text{bodytime} \equiv do \{$   
 $xs \leftarrow \text{tsl};$   
 $(-, \sigma) \leftarrow whileIET (\lambda(it, \sigma). \exists xs'. xs = xs' @ it \wedge \Phi (set it) \sigma) (\lambda(it, \sigma). length$   
 $it * \text{bodytime})$   
 $(FOREACH\text{-cond} c) (FOREACH\text{-body} f) (xs, \sigma 0);$   
 $RETURN \sigma\}$

**lemma**  $FOREACHoci\text{-by-LIST-FOREACH}$ :

$FOREACHoci R \Phi S c f \sigma 0 \text{to-sorted-list-time} \text{bodytime} = do \{$   
 $ASSERT (finite S);$   
 $LIST-FOREACH \Phi (it\text{-to-sorted-list} R S \text{to-sorted-list-time}) c f \sigma 0 \text{bodytime}$   
 $\}$   
 $\langle proof \rangle$

**end**

## References

- [1] M. P. L. Haslbeck and P. Lammich. For a few dollars more. In N. Yoshida, editor, *Programming Languages and Systems*, pages 292–319, Cham, 2021. Springer International Publishing.
- [2] M. P. L. Haslbeck and P. Lammich. For a few dollars more: Verified fine-grained algorithm analysis down to llvm. *ACM Trans. Program. Lang. Syst.*, 44(3), jul 2022.