

# PAC Checker

Mathias Fleury and Daniela Kaufmann

July 21, 2020

## Abstract

Abstract—Generating and checking proof certificates is important to increase the trust in automated reasoning tools. In recent years formal verification using computer algebra became more important and is heavily used in automated circuit verification. An existing proof format which covers algebraic reasoning and allows efficient proof checking is the practical algebraic calculus. In this development, we present the verified checker Pastèque that is obtained by synthesis via the Refinement Framework.

This is the formalization going with our FMCAD’20 tool presentation [1].

## Contents

<b>1</b>	<b>Duplicate Free Multisets</b>	<b>2</b>
1.1	More Lists . . . . .	4
1.2	Generic Multiset . . . . .	4
1.3	Other . . . . .	5
1.4	More Theorem about Loops . . . . .	36
<b>2</b>	<b>Libraries</b>	<b>39</b>
2.1	More Polynoms . . . . .	39
2.2	More Ideals . . . . .	53
<b>3</b>	<b>Specification of the PAC checker</b>	<b>54</b>
3.1	Ideals . . . . .	54
3.2	PAC Format . . . . .	55
<b>4</b>	<b>Finite maps and multisets</b>	<b>64</b>
4.1	Finite sets and multisets . . . . .	64
4.2	Finite map and multisets . . . . .	64
<b>5</b>	<b>Hash-Map for finite mappings</b>	<b>68</b>
5.1	Operations . . . . .	69
5.2	Patterns . . . . .	70
5.3	Mapping to Normal Hashmaps . . . . .	71
<b>6</b>	<b>Checker Algorithm</b>	<b>73</b>
6.1	Specification . . . . .	73
6.2	Algorithm . . . . .	75
6.3	Full Checker . . . . .	86

<b>7</b>	<b>Polynomials of strings</b>	<b>88</b>
7.1	Polynomials and Variables . . . . .	88
7.2	Addition . . . . .	90
7.3	Normalisation . . . . .	91
7.4	Correctness . . . . .	92
<b>8</b>	<b>Terms</b>	<b>97</b>
8.1	Ordering . . . . .	97
8.2	Polynomials . . . . .	98
<b>9</b>	<b>Polynomialss as Lists</b>	<b>100</b>
9.1	Addition . . . . .	100
9.2	Multiplication . . . . .	106
9.3	Normalisation . . . . .	112
9.4	Multiplication and normalisation . . . . .	120
9.5	Correctness . . . . .	121
<b>10</b>	<b>Executable Checker</b>	<b>122</b>
10.1	Definitions . . . . .	123
10.2	Correctness . . . . .	131
<b>11</b>	<b>Various Refinement Relations</b>	<b>147</b>
<b>12</b>	<b>Initial Normalisation of Polynoms</b>	<b>154</b>
12.1	Sorting . . . . .	154
12.2	Sorting applied to monomials . . . . .	156
12.3	Lifting to polynomials . . . . .	159
<b>13</b>	<b>Code Synthesis of the Complete Checker</b>	<b>170</b>
<b>14</b>	<b>Correctness theorem</b>	<b>185</b>
<b>theory</b> <i>Duplicate-Free-Multiset</i>		
<b>imports</b> <i>Nested-Multisets-Ordinals.Multiset-More</i>		
<b>begin</b>		

## 1 Duplicate Free Multisets

Duplicate free multisets are isomorphic to finite sets, but it can be useful to reason about duplication to speak about intermediate execution steps in the refinements.

**lemma** *distinct-mset-remdups-mset-id*:  $\langle \text{distinct-mset } C \implies \text{remdups-mset } C = C \rangle$   
**by** (*induction C*) *auto*

**lemma** *notin-add-mset-remdups-mset*:  
 $\langle a \notin \# A \implies \text{add-mset } a (\text{remdups-mset } A) = \text{remdups-mset } (\text{add-mset } a A) \rangle$   
**by** *auto*

**lemma** *distinct-mset-image-mset*:  
 $\langle \text{distinct-mset } (\text{image-mset } f (\text{mset } xs)) \longleftrightarrow \text{distinct } (\text{map } f xs) \rangle$   
**apply** (*subst mset-map[symmetric]*)  
**apply** (*subst distinct-mset-mset-distinct*)  
**..**

**lemma** *distinct-image-mset-not-equal*:  
**assumes**  
 $LL': \langle L \neq L' \rangle$  **and**  
 $dist: \langle distinct\text{-}mset\ (image\text{-}mset\ f\ M) \rangle$  **and**  
 $L: \langle L \in\# M \rangle$  **and**  
 $L': \langle L' \in\# M \rangle$  **and**  
 $fLL'[simp]: \langle f\ L = f\ L' \rangle$   
**shows**  $\langle False \rangle$   
**proof** –  
**obtain**  $M1$  **where**  $M1: \langle M = add\text{-}mset\ L\ M1 \rangle$   
**using** *multi-member-split*[*OF*  $L$ ] **by** *blast*  
**obtain**  $M2$  **where**  $M2: \langle M1 = add\text{-}mset\ L'\ M2 \rangle$   
**using** *multi-member-split*[*of*  $L'\ M1$ ]  $LL'\ L'$  **unfolding**  $M1$  **by** (*auto simp: add-mset-eq-add-mset*)  
**show**  $False$   
**using**  $dist$  **unfolding**  $M1\ M2$  **by** *auto*  
**qed**  
**lemma** *distinct-mset-mono*:  $\langle D' \subseteq\# D \implies distinct\text{-}mset\ D \implies distinct\text{-}mset\ D' \rangle$   
**by** (*metis distinct-mset-union subset-mset.le-iff-add*)  
  
**lemma** *distinct-mset-mono-strict*:  $\langle D' \subset\# D \implies distinct\text{-}mset\ D \implies distinct\text{-}mset\ D' \rangle$   
**using** *distinct-mset-mono* **by** *auto*  
  
**lemma** *distinct-set-mset-eq-iff*:  
**assumes**  $\langle distinct\text{-}mset\ M \rangle\ \langle distinct\text{-}mset\ N \rangle$   
**shows**  $\langle set\text{-}mset\ M = set\text{-}mset\ N \longleftrightarrow M = N \rangle$   
**using** *assms distinct-mset-set-mset-ident* **by** *fastforce*  
  
**lemma** *distinct-mset-union2*:  
 $\langle distinct\text{-}mset\ (A + B) \implies distinct\text{-}mset\ B \rangle$   
**using** *distinct-mset-union*[*of*  $B\ A$ ]  
**by** (*auto simp: ac-simps*)  
  
**lemma** *distinct-mset-mset-set*:  $\langle distinct\text{-}mset\ (mset\text{-}set\ A) \rangle$   
**unfolding** *distinct-mset-def count-mset-set-if* **by** (*auto simp: not-in-iff*)  
  
**lemma** *distinct-mset-inter-remdups-mset*:  
**assumes**  $dist: \langle distinct\text{-}mset\ A \rangle$   
**shows**  $\langle A \cap\# remdups\text{-}mset\ B = A \cap\# B \rangle$   
**proof** –  
**have** [*simp*]:  $\langle A' \cap\# remove1\text{-}mset\ a\ (remdups\text{-}mset\ Aa) = A' \cap\# Aa \rangle$   
**if**  
 $\langle A' \cap\# remdups\text{-}mset\ Aa = A' \cap\# Aa \rangle$  **and**  
 $\langle a \notin\# A' \rangle$  **and**  
 $\langle a \in\# Aa \rangle$   
**for**  $A'\ Aa :: \langle 'a\ multiset \rangle$  **and**  $a$   
**by** (*metis insert-DiffM inter-add-right1 set-mset-remdups-mset that*)  
  
**show** *?thesis*  
**using**  $dist$   
**apply** (*induction*  $A$ )  
**subgoal** **by** *auto*  
**subgoal** **for**  $a\ A'$   
**apply** (*cases*  $\langle a \in\# B \rangle$ )  
**using** *multi-member-split*[*of*  $a\ \langle B \rangle$ ] *multi-member-split*[*of*  $a\ \langle A \rangle$ ]  
**by** (*auto simp: mset-set.insert-remove*)

done  
qed

**lemma** *finite-mset-set-inter*:

$\langle \text{finite } A \implies \text{finite } B \implies \text{mset-set } (A \cap B) = \text{mset-set } A \cap \# \text{ mset-set } B \rangle$   
**apply** (*induction*  $A$  *rule*: *finite-induct*)  
**subgoal** *by auto*  
**subgoal** *for*  $a \ A$   
**apply** (*cases*  $\langle a \in B \rangle$ ; *cases*  $\langle a \in \# \text{ mset-set } B \rangle$ )  
**using** *multi-member-split*[*of*  $a \ \langle \text{mset-set } B \rangle$ ]  
**by** (*auto simp*: *mset-set.insert-remove*)  
done

**lemma** *removeAll-notin*:  $\langle a \notin \# A \implies \text{removeAll-mset } a \ A = A \rangle$   
**using** *count-inI* **by** *force*

**lemma** *same-mset-distinct-iff*:

$\langle \text{mset } M = \text{mset } M' \implies \text{distinct } M \longleftrightarrow \text{distinct } M' \rangle$   
**by** (*auto simp*: *distinct-mset-mset-distinct*[*symmetric*] *simp del*: *distinct-mset-mset-distinct*)

## 1.1 More Lists

**lemma** *in-set-conv-iff*:

$\langle x \in \text{set } (\text{take } n \ xs) \longleftrightarrow (\exists i < n. i < \text{length } xs \wedge xs ! i = x) \rangle$   
**apply** (*induction*  $n$ )  
**subgoal** *by auto*  
**subgoal** *for*  $n$   
**apply** (*cases*  $\langle \text{Suc } n < \text{length } xs \rangle$ )  
**subgoal** *by* (*auto simp*: *take-Suc-conv-app-nth less-Suc-eq dest*: *in-set-takeD*)  
**subgoal**  
**apply** (*cases*  $\langle n < \text{length } xs \rangle$ )  
**subgoal**  
**apply** (*auto simp*: *in-set-conv-nth*)  
**by** (*rule-tac*  $x=i$  *in* *exI*; *auto*; *fail*)  
**subgoal**  
**apply** (*auto simp*: *take-Suc-conv-app-nth dest*: *in-set-takeD*)  
**by** (*rule-tac*  $x=i$  *in* *exI*; *auto*; *fail*)  
**done**  
**done**  
**done**

**lemma** *in-set-take-conv-nth*:

$\langle x \in \text{set } (\text{take } n \ xs) \longleftrightarrow (\exists m < \min n \ (\text{length } xs). xs ! m = x) \rangle$   
**by** (*metis in-set-conv-nth length-take min.commute min.strict-boundedE nth-take*)

**lemma** *in-set-remove1D*:

$\langle a \in \text{set } (\text{remove1 } x \ xs) \implies a \in \text{set } xs \rangle$   
**by** (*meson notin-set-remove1*)

## 1.2 Generic Multiset

**lemma** *mset-drop-upto*:  $\langle \text{mset } (\text{drop } a \ N) = \{ \#N ! i. i \in \# \text{ mset-set } \{ a..<\text{length } N \} \# \} \rangle$

**proof** (*induction*  $N$  *arbitrary*:  $a$ )

**case** *Nil*

**then show** *?case* **by** *simp*

**next**

```

case (Cons c N)
have upt: ⟨{0..Suc (length N)} = insert 0 {1..Suc (length N)}⟩
  by auto
then have H: ⟨mset-set {0..Suc (length N)} = add-mset 0 (mset-set {1..Suc (length N)})⟩
  unfolding upt by auto
have mset-case-Suc: ⟨{#case x of 0 ⇒ c | Suc x ⇒ N ! x . x ∈# mset-set {Suc a..Suc b}#} =
  {#N ! (x-1) . x ∈# mset-set {Suc a..Suc b}#}⟩ for a b
  by (rule image-mset-cong) (auto split: nat.splits)
have Suc-Suc: ⟨{Suc a..Suc b} = Suc ‘ {a..b}⟩ for a b
  by auto
then have mset-set-Suc-Suc: ⟨mset-set {Suc a..Suc b} = {#Suc n. n ∈# mset-set {a..b}#}⟩ for
a b
  unfolding Suc-Suc by (subst image-mset-mset-set[symmetric]) auto
have *: ⟨{#N ! (x-Suc 0) . x ∈# mset-set {Suc a..Suc b}#} = {#N ! x . x ∈# mset-set {a..b}#}⟩
  for a b
  by (auto simp add: mset-set-Suc-Suc)
show ?case
  apply (cases a)
  using Cons[of 0] Cons by (auto simp: nth-Cons drop-Cons H mset-case-Suc *)
qed

```

### 1.3 Other

I believe this should be activated by default, as the set becomes much easier...

```

lemma Collect-eq-comp': ⟨ {(x, y). P x y} O {(c, a). c = f a} = {(x, a). P x (f a)}⟩
  by auto

```

**end**

**theory** *WB-Sort*

```

imports Refine-Imperative-HOL.IICF HOL-Library.Rewrite Duplicate-Free-Multiset
begin

```

This a complete copy-paste of the IsaFoL version because sharing is too hard.

Every element between *lo* and *hi* can be chosen as pivot element.

```

definition choose-pivot :: ⟨('b ⇒ 'b ⇒ bool) ⇒ ('a ⇒ 'b) ⇒ 'a list ⇒ nat ⇒ nat ⇒ nat nres⟩ where
  ⟨choose-pivot - - lo hi = SPEC(λk. k ≥ lo ∧ k ≤ hi)⟩

```

The element at index *p* partitions the subarray *lo*..*hi*. This means that every element

```

definition isPartition-wrt :: ⟨('b ⇒ 'b ⇒ bool) ⇒ 'b list ⇒ nat ⇒ nat ⇒ nat ⇒ bool⟩ where
  ⟨isPartition-wrt R xs lo hi p ≡ (∀ i. i ≥ lo ∧ i < p ⟶ R (xs!i) (xs!p)) ∧ (∀ j. j > p ∧ j ≤ hi ⟶
  R (xs!p) (xs!j))⟩

```

**lemma** *isPartition-wrtI*:

```

  ⟨(∧ i. [i ≥ lo; i < p] ⟹ R (xs!i) (xs!p)) ⟹ (∧ j. [j > p; j ≤ hi] ⟹ R (xs!p) (xs!j)) ⟹
  isPartition-wrt R xs lo hi p⟩
  by (simp add: isPartition-wrt-def)

```

**definition** *isPartition* :: ⟨'a :: order list ⇒ nat ⇒ nat ⇒ nat ⇒ bool⟩ **where**

```

  ⟨isPartition xs lo hi p ≡ isPartition-wrt (≤) xs lo hi p⟩

```

**abbreviation** *isPartition-map* :: ⟨('b ⇒ 'b ⇒ bool) ⇒ ('a ⇒ 'b) ⇒ 'a list ⇒ nat ⇒ nat ⇒ nat ⇒ bool⟩  
**where**

```

  ⟨isPartition-map R h xs i j k ≡ isPartition-wrt (λa b. R (h a) (h b)) xs i j k⟩

```

**lemma** *isPartition-map-def'*:

$\langle lo \leq p \implies p \leq hi \implies hi < \text{length } xs \implies \text{isPartition-map } R \ h \ xs \ lo \ hi \ p = \text{isPartition-wrt } R \ (\text{map } h \ xs) \ lo \ hi \ p \rangle$

**by** (*auto simp add: isPartition-wrt-def conjI*)

Example: 6 is the pivot element (with index 4);  $7::'a$  is equal to the  $\text{length } xs - 1$ .

**lemma**  $\langle \text{isPartition } [0,5,3,4,6,9,8,10::\text{nat}] \ 0 \ 7 \ 4 \rangle$

**by** (*auto simp add: isPartition-def isPartition-wrt-def nth-Cons'*)

**definition** *sublist* ::  $\langle 'a \text{ list} \Rightarrow \text{nat} \Rightarrow \text{nat} \Rightarrow 'a \text{ list} \rangle$  **where**

$\langle \text{sublist } xs \ i \ j \equiv \text{take } (\text{Suc } j - i) \ (\text{drop } i \ xs) \rangle$

**lemma** *take-Suc0*:

$l \neq [] \implies \text{take } (\text{Suc } 0) \ l = [!l0]$

$0 < \text{length } l \implies \text{take } (\text{Suc } 0) \ l = [!l0]$

$\text{Suc } n \leq \text{length } l \implies \text{take } (\text{Suc } 0) \ l = [!l0]$

**by** (*cases l, auto*)+

**lemma** *sublist-single*:  $\langle i < \text{length } xs \implies \text{sublist } xs \ i \ i = [xs!i] \rangle$

**by** (*cases xs*) (*auto simp add: sublist-def take-Suc0*)

**lemma** *insert-eq*:  $\langle \text{insert } a \ b = b \cup \{a\} \rangle$

**by** *auto*

**lemma** *sublist-nth*:  $\langle [lo \leq hi; hi < \text{length } xs; k+lo \leq hi] \implies (\text{sublist } xs \ lo \ hi)!k = xs!(lo+k) \rangle$

**by** (*simp add: sublist-def*)

**lemma** *sublist-length*:  $\langle [i \leq j; j < \text{length } xs] \implies \text{length } (\text{sublist } xs \ i \ j) = 1 + j - i \rangle$

**by** (*simp add: sublist-def*)

**lemma** *sublist-not-empty*:  $\langle [i \leq j; j < \text{length } xs; xs \neq []] \implies \text{sublist } xs \ i \ j \neq [] \rangle$

**apply** *simp*

**apply** (*rewrite List.length-greater-0-conv[symmetric]*)

**apply** (*rewrite sublist-length*)

**by** *auto*

**lemma** *sublist-app*:  $\langle [i1 \leq i2; i2 \leq i3] \implies \text{sublist } xs \ i1 \ i2 @ \text{sublist } xs \ (\text{Suc } i2) \ i3 = \text{sublist } xs \ i1 \ i3 \rangle$

**unfolding** *sublist-def*

**by** (*smt Suc-eq-plus1-left Suc-le-mono append.assoc le-SucI le-add-diff-inverse le-trans same-append-eq take-add*)

**definition** *sorted-sublist-wrt* ::  $\langle ('b \Rightarrow 'b \Rightarrow \text{bool}) \Rightarrow 'b \text{ list} \Rightarrow \text{nat} \Rightarrow \text{nat} \Rightarrow \text{bool} \rangle$  **where**

$\langle \text{sorted-sublist-wrt } R \ xs \ lo \ hi = \text{sorted-wrt } R \ (\text{sublist } xs \ lo \ hi) \rangle$

**definition** *sorted-sublist* ::  $\langle 'a :: \text{linorder} \text{ list} \Rightarrow \text{nat} \Rightarrow \text{nat} \Rightarrow \text{bool} \rangle$  **where**

$\langle \text{sorted-sublist } xs \ lo \ hi = \text{sorted-sublist-wrt } (\leq) \ xs \ lo \ hi \rangle$

**abbreviation** *sorted-sublist-map* ::  $\langle ('b \Rightarrow 'b \Rightarrow \text{bool}) \Rightarrow ('a \Rightarrow 'b) \Rightarrow 'a \text{ list} \Rightarrow \text{nat} \Rightarrow \text{nat} \Rightarrow \text{bool} \rangle$  **where**

$\langle \text{sorted-sublist-map } R \ h \ xs \ lo \ hi \equiv \text{sorted-sublist-wrt } (\lambda a \ b. \ R \ (h \ a) \ (h \ b)) \ xs \ lo \ hi \rangle$

**lemma** *sorted-sublist-map-def'*:

$\langle lo < \text{length } xs \implies \text{sorted-sublist-map } R \ h \ xs \ lo \ hi \equiv \text{sorted-sublist-wrt } R \ (\text{map } h \ xs) \ lo \ hi \rangle$

**apply** (*simp add: sorted-sublist-wrt-def*)

**by** (*simp add: drop-map sorted-wrt-map sublist-def take-map*)

**lemma** *sorted-sublist-wrt-refl*:  $\langle i < \text{length } xs \implies \text{sorted-sublist-wrt } R \ xs \ i \ i \rangle$

**by** (*auto simp add: sorted-sublist-wrt-def sublist-single*)

**lemma** *sorted-sublist-refl*:  $\langle i < \text{length } xs \implies \text{sorted-sublist } xs \ i \ i \rangle$

**by** (*auto simp add: sorted-sublist-def sorted-sublist-wrt-refl*)

**lemma** *sublist-map*:  $\langle \text{sublist } (\text{map } f \ xs) \ i \ j = \text{map } f \ (\text{sublist } xs \ i \ j) \rangle$

**apply** (*auto simp add: sublist-def*)

**by** (*simp add: drop-map take-map*)

**lemma** *take-set*:  $\langle j \leq \text{length } xs \implies x \in \text{set } (\text{take } j \ xs) \equiv (\exists \ k. \ k < j \wedge xs!k = x) \rangle$

**apply** (*induction xs*)

**apply** *simp*

**by** (*meson in-set-conv-iff less-le-trans*)

**lemma** *drop-set*:  $\langle j \leq \text{length } xs \implies x \in \text{set } (\text{drop } j \ xs) \equiv (\exists \ k. \ j \leq k \wedge k < \text{length } xs \wedge xs!k = x) \rangle$

**by** (*smt Misc.in-set-drop-conv-nth*)

**lemma** *sublist-el*:  $\langle i \leq j \implies j < \text{length } xs \implies x \in \text{set } (\text{sublist } xs \ i \ j) \equiv (\exists \ k. \ k < \text{Suc } j - i \wedge xs!(i+k) = x) \rangle$

**by** (*auto simp add: take-set sublist-def*)

**lemma** *sublist-el'*:  $\langle i \leq j \implies j < \text{length } xs \implies x \in \text{set } (\text{sublist } xs \ i \ j) \equiv (\exists \ k. \ i \leq k \wedge k \leq j \wedge xs!k = x) \rangle$

**apply** (*auto simp add: sublist-el*)

**by** (*smt Groups.add-ac(2) le-add1 le-add-diff-inverse less-Suc-eq less-diff-conv nat-less-le order-refl*)

**lemma** *sublist-lt*:  $\langle hi < lo \implies \text{sublist } xs \ lo \ hi = [] \rangle$

**by** (*auto simp add: sublist-def*)

**lemma** *nat-le-eq-or-lt*:  $\langle (a :: \text{nat}) \leq b = (a = b \vee a < b) \rangle$

**by** *linarith*

**lemma** *sorted-sublist-wrt-le*:  $\langle hi \leq lo \implies hi < \text{length } xs \implies \text{sorted-sublist-wrt } R \ xs \ lo \ hi \rangle$

**apply** (*auto simp add: nat-le-eq-or-lt*)

**unfolding** *sorted-sublist-wrt-def*

**subgoal apply** (*rewrite sublist-single*) **by** *auto*

**subgoal by** (*auto simp add: sublist-lt*)

**done**

Elements in a sorted sublists are actually sorted

**lemma** *sorted-sublist-wrt-nth-le*:

**assumes**  $\langle \text{sorted-sublist-wrt } R \ xs \ lo \ hi \rangle$  **and**  $\langle lo \leq hi \rangle$  **and**  $\langle hi < \text{length } xs \rangle$  **and**

$\langle lo \leq i \rangle$  **and**  $\langle i < j \rangle$  **and**  $\langle j \leq hi \rangle$

**shows**  $\langle R \ (xs!i) \ (xs!j) \rangle$

**proof** –

**have** *A*:  $\langle lo < \text{length } xs \rangle$  **using** *assms(2)* *assms(3)* **by** *linarith*

```

obtain  $i'$  where  $I: \langle i = lo + i' \rangle$  using  $assms(4)$   $le\text{-}Suc\text{-}ex$  by  $auto$ 
obtain  $j'$  where  $J: \langle j = lo + j' \rangle$  by  $(meson\ assms(4)\ assms(5)\ dual\text{-}order.trans\ le\text{-}iff\text{-}add\ less\text{-}imp\text{-}le\text{-}nat)$ 
show  $?thesis$ 
  using  $assms(1)$  apply  $(simp\ add: sorted\text{-}sublist\text{-}wrt\text{-}def\ I\ J)$ 
  apply  $(rewrite\ sublist\text{-}nth[symmetric, \text{where } k=i', \text{where } lo=lo, \text{where } hi=hi])$ 
  using  $assms$  apply  $auto$  subgoal using  $I$  by  $linarith$ 
  apply  $(rewrite\ sublist\text{-}nth[symmetric, \text{where } k=j', \text{where } lo=lo, \text{where } hi=hi])$ 
  using  $assms$  apply  $auto$  subgoal using  $J$  by  $linarith$ 
  apply  $(rule\ sorted\text{-}wrt\text{-}nth\text{-}less)$ 
  apply  $auto$ 
  subgoal using  $I\ J\ nat\text{-}add\text{-}left\text{-}cancel\text{-}less$  by  $blast$ 
  subgoal apply  $(simp\ add: sublist\text{-}length)$  using  $J$  by  $linarith$ 
done
qed

```

We can make the assumption  $i < j$  weaker if we have a reflexive relation.

```

lemma  $sorted\text{-}sublist\text{-}wrt\text{-}nth\text{-}le'$ :
  assumes  $ref: \langle \bigwedge x. R\ x\ x \rangle$ 
  and  $\langle sorted\text{-}sublist\text{-}wrt\ R\ xs\ lo\ hi \rangle$  and  $\langle lo \leq hi \rangle$  and  $\langle hi < length\ xs \rangle$ 
  and  $\langle lo \leq i \rangle$  and  $\langle i \leq j \rangle$  and  $\langle j \leq hi \rangle$ 
  shows  $\langle R\ (xs!i)\ (xs!j) \rangle$ 
proof -
  have  $\langle i < j \vee i = j \rangle$  using  $\langle i \leq j \rangle$  by  $linarith$ 
  then consider  $(a)\ \langle i < j \rangle \mid$ 
     $(b)\ \langle i = j \rangle$  by  $blast$ 
  then show  $?thesis$ 
proof cases
  case  $a$ 
  then show  $?thesis$ 
    using  $assms(2-5,7)$   $sorted\text{-}sublist\text{-}wrt\text{-}nth\text{-}le$  by  $blast$ 
next
  case  $b$ 
  then show  $?thesis$ 
    by  $(simp\ add: ref)$ 
qed
qed

```

```

lemma  $sorted\text{-}sublist\text{-}le: \langle hi \leq lo \implies hi < length\ xs \implies sorted\text{-}sublist\ xs\ lo\ hi \rangle$ 
by  $(auto\ simp\ add: sorted\text{-}sublist\text{-}def\ sorted\text{-}sublist\text{-}wrt\text{-}le)$ 

```

```

lemma  $sorted\text{-}sublist\text{-}map\text{-}le: \langle hi \leq lo \implies hi < length\ xs \implies sorted\text{-}sublist\text{-}map\ R\ h\ xs\ lo\ hi \rangle$ 
by  $(auto\ simp\ add: sorted\text{-}sublist\text{-}wrt\text{-}le)$ 

```

```

lemma  $sublist\text{-}cons: \langle lo < hi \implies hi < length\ xs \implies sublist\ xs\ lo\ hi = xs!lo \# sublist\ xs\ (Suc\ lo)\ hi \rangle$ 
apply  $(simp\ add: sublist\text{-}def)$ 
by  $(metis\ Cons\text{-}nth\text{-}drop\text{-}Suc\ Suc\text{-}diff\text{-}le\ le\text{-}trans\ less\text{-}imp\text{-}le\text{-}nat\ not\text{-}le\ take\text{-}Suc\ Cons)$ 

```

```

lemma  $sorted\text{-}sublist\text{-}wrt\text{-}cons'$ :
   $\langle sorted\text{-}sublist\text{-}wrt\ R\ xs\ (lo+1)\ hi \implies lo \leq hi \implies hi < length\ xs \implies (\forall j. lo < j \wedge j \leq hi \longrightarrow R\ (xs!lo)) \rangle$ 

```



```

(xs!j))  $\implies$  sorted-sublist-wrt R xs lo hi
  apply (simp add: sorted-sublist-wrt-def)
  apply (auto simp add: nat-le-eq-or-lt)
  subgoal by (simp add: sublist-single)
  apply (auto simp add: sublist-cons sublist-el)
  by (metis Suc-lessI ab-semigroup-add-class.add commute less-add-Suc1 less-diff-conv)

```

**lemma** sorted-sublist-wrt-cons:

```

  assumes trans:  $\langle \bigwedge x y z. \llbracket R x y; R y z \rrbracket \implies R x z \rangle$  and
     $\langle \text{sorted-sublist-wrt } R \text{ xs } (lo+1) \text{ hi} \rangle$  and
     $\langle lo \leq hi \rangle$  and  $\langle hi < \text{length xs} \rangle$  and  $\langle R (xs!lo) (xs!(lo+1)) \rangle$ 
  shows  $\langle \text{sorted-sublist-wrt } R \text{ xs } lo \text{ hi} \rangle$ 

```

**proof** –

show ?thesis

apply (rule sorted-sublist-wrt-cons') using assms apply auto

subgoal premises assms' for j

**proof** –

have A:  $\langle j=lo+1 \vee j>lo+1 \rangle$  using assms'(5) by linarith

show ?thesis

using A **proof**

assume A:  $\langle j=lo+1 \rangle$  show ?thesis

by (simp add: A assms')

**next**

assume A:  $\langle j>lo+1 \rangle$  show ?thesis

apply (rule trans)

apply (rule assms(5))

apply (rule sorted-sublist-wrt-nth-le[OF assms(2), where i= $\langle lo+1 \rangle$ , where j= $\langle j \rangle$ ])

subgoal using A assms'(6) by linarith

subgoal using assms'(3) less-imp-diff-less by blast

subgoal using assms'(5) by auto

subgoal using A by linarith

subgoal by (simp add: assms'(6))

done

qed

qed

done

qed

**lemma** sorted-sublist-map-cons:

```

 $\langle \bigwedge x y z. \llbracket R (h x) (h y); R (h y) (h z) \rrbracket \implies R (h x) (h z) \rangle \implies$ 
  sorted-sublist-map R h xs (lo+1) hi  $\implies lo \leq hi \implies hi < \text{length xs} \implies R (h (xs!lo)) (h (xs!(lo+1)))$ 
 $\implies \text{sorted-sublist-map } R \text{ h xs } lo \text{ hi}$ 
  by (blast intro: sorted-sublist-wrt-cons)

```

**lemma** sublist-snoc:  $\langle lo < hi \implies hi < \text{length xs} \implies \text{sublist xs } lo \text{ hi} = \text{sublist xs } lo (hi-1) @ [xs!hi] \rangle$

apply (simp add: sublist-def)

**proof** –

assume a1:  $lo < hi$

assume  $hi < \text{length xs}$

**then have** take lo xs @ take (Suc hi - lo) (drop lo xs) = (take lo xs @ take (hi - lo) (drop lo xs)) @ [xs ! hi]

using a1 by (metis (no-types) Suc-diff-le add-Suc-right hd-drop-conv-nth le-add-diff-inverse less-imp-le-nat take-add take-hd-drop)

**then show** take (Suc hi - lo) (drop lo xs) = take (hi - lo) (drop lo xs) @ [xs ! hi]

by simp  
qed

**lemma** *sorted-sublist-wrt-snoc'*:

$\langle \text{sorted-sublist-wrt } R \text{ } xs \text{ } lo \text{ } (hi-1) \implies lo \leq hi \implies hi < \text{length } xs \implies (\forall j. lo \leq j \wedge j < hi \longrightarrow R \text{ } (xs!j) \text{ } (xs!hi)) \implies \text{sorted-sublist-wrt } R \text{ } xs \text{ } lo \text{ } hi \rangle$   
 apply (simp add: sorted-sublist-wrt-def)  
 apply (auto simp add: nat-le-eq-or-lt)  
 subgoal by (simp add: sublist-single)  
 apply (auto simp add: sublist-snoc sublist-el sorted-wrt-append)  
 by (metis ab-semigroup-add-class.add.commute leI less-diff-conv nat-le-eq-or-lt not-add-less1)

**lemma** *sorted-sublist-wrt-snoc*:

assumes *trans*:  $\langle \bigwedge x \ y \ z. \llbracket R \ x \ y; R \ y \ z \rrbracket \implies R \ x \ z \rangle$  and  
 $\langle \text{sorted-sublist-wrt } R \text{ } xs \text{ } lo \text{ } (hi-1) \rangle$  and  
 $\langle lo \leq hi \rangle$  and  $\langle hi < \text{length } xs \rangle$  and  $\langle (R \text{ } (xs!(hi-1)) \text{ } (xs!hi)) \rangle$   
 shows  $\langle \text{sorted-sublist-wrt } R \text{ } xs \text{ } lo \text{ } hi \rangle$   
 proof –  
 show ?thesis  
 apply (rule sorted-sublist-wrt-snoc') using assms apply auto  
 subgoal premises *assms'* for *j*  
 proof –  
 have *A*:  $\langle j=hi-1 \vee j < hi-1 \rangle$  using *assms'*(6) by linarith  
 show ?thesis  
 using *A* proof  
 assume *A*:  $\langle j=hi-1 \rangle$  show ?thesis  
 by (simp add: *A* *assms'*)  
 next  
 assume *A*:  $\langle j < hi-1 \rangle$  show ?thesis  
 apply (rule *trans*)  
 apply (rule sorted-sublist-wrt-nth-le[OF *assms'*(2), where *i=j*, where *j=hi-1*])  
 prefer 6  
 apply (rule *assms'*(5))  
 apply auto  
 subgoal using *A* *assms'*(5) by linarith  
 subgoal using *assms'*(3) less-imp-diff-less by blast  
 subgoal using *assms'*(5) by auto  
 subgoal using *A* by linarith  
 done  
 qed  
 qed  
 done  
 qed

**lemma** *sublist-split*:  $\langle lo \leq hi \implies lo < p \implies p < hi \implies hi < \text{length } xs \implies \text{sublist } xs \text{ } lo \text{ } p \text{ } @ \text{ sublist } xs \text{ } (p+1) \text{ } hi = \text{sublist } xs \text{ } lo \text{ } hi \rangle$   
 by (simp add: sublist-app)

**lemma** *sublist-split-part*:  $\langle lo \leq hi \implies lo < p \implies p < hi \implies hi < \text{length } xs \implies \text{sublist } xs \text{ } lo \text{ } (p-1) \text{ } @ \text{ xs!p} \# \text{ sublist } xs \text{ } (p+1) \text{ } hi = \text{sublist } xs \text{ } lo \text{ } hi \rangle$   
 by (auto simp add: sublist-split[symmetric] sublist-snoc[where *xs=xs*, where *lo=lo*, where *hi=p*])

A property for partitions (we always assume that *R* is transitive.

**lemma** *isPartition-wrt-trans*:

$\langle (\bigwedge x y z. \llbracket R x y; R y z \rrbracket \implies R x z) \implies$   
 $isPartition-wrt R xs lo hi p \implies$   
 $(\forall i j. lo \leq i \wedge i < p \wedge p < j \wedge j \leq hi \longrightarrow R (xs!i) (xs!j)) \rangle$   
**by** (auto simp add: isPartition-wrt-def)

**lemma** *isPartition-map-trans:*

$\langle (\bigwedge x y z. \llbracket R (h x) (h y); R (h y) (h z) \rrbracket \implies R (h x) (h z)) \implies$   
 $hi < length xs \implies$   
 $isPartition-map R h xs lo hi p \implies$   
 $(\forall i j. lo \leq i \wedge i < p \wedge p < j \wedge j \leq hi \longrightarrow R (h (xs!i)) (h (xs!j))) \rangle$   
**by** (auto simp add: isPartition-wrt-def)

**lemma** *merge-sorted-wrt-partitions-between':*

$\langle lo \leq hi \implies lo < p \implies p < hi \implies hi < length xs \implies$   
 $isPartition-wrt R xs lo hi p \implies$   
 $sorted-sublist-wrt R xs lo (p-1) \implies sorted-sublist-wrt R xs (p+1) hi \implies$   
 $(\forall i j. lo \leq i \wedge i < p \wedge p < j \wedge j \leq hi \longrightarrow R (xs!i) (xs!j)) \implies$   
 $sorted-sublist-wrt R xs lo hi \rangle$

**apply** (auto simp add: isPartition-def isPartition-wrt-def sorted-sublist-def sorted-sublist-wrt-def sublist-map)

**apply** (simp add: sublist-split-part[symmetric])

**apply** (auto simp add: List.sorted-wrt-append)

**subgoal by** (auto simp add: sublist-el)

**subgoal by** (auto simp add: sublist-el)

**subgoal by** (auto simp add: sublist-el')

**done**

**lemma** *merge-sorted-wrt-partitions-between:*

$\langle (\bigwedge x y z. \llbracket R x y; R y z \rrbracket \implies R x z) \implies$   
 $isPartition-wrt R xs lo hi p \implies$   
 $sorted-sublist-wrt R xs lo (p-1) \implies sorted-sublist-wrt R xs (p+1) hi \implies$   
 $lo \leq hi \implies hi < length xs \implies lo < p \implies p < hi \implies hi < length xs \implies$   
 $sorted-sublist-wrt R xs lo hi \rangle$

**by** (simp add: merge-sorted-wrt-partitions-between' isPartition-wrt-trans)

The main theorem to merge sorted lists

**lemma** *merge-sorted-wrt-partitions:*

$\langle isPartition-wrt R xs lo hi p \implies$   
 $sorted-sublist-wrt R xs lo (p - Suc 0) \implies sorted-sublist-wrt R xs (Suc p) hi \implies$   
 $lo \leq hi \implies lo \leq p \implies p \leq hi \implies hi < length xs \implies$   
 $(\forall i j. lo \leq i \wedge i < p \wedge p < j \wedge j \leq hi \longrightarrow R (xs!i) (xs!j)) \implies$   
 $sorted-sublist-wrt R xs lo hi \rangle$

**subgoal premises** *assms*

**proof** —

**have** *C*:  $\langle lo=p \wedge p=hi \vee lo=p \wedge p < hi \vee lo < p \wedge p=hi \vee lo < p \wedge p < hi \rangle$

**using** *assms* **by** *linarith*

**show** *?thesis*

**using** *C* **apply** *auto*

**subgoal** —  $lo=p=hi$

**apply** (rule sorted-sublist-wrt-refl)

**using** *assms* **by** *auto*

**subgoal** —  $lo=p < hi$

**using** *assms* **by** (simp add: isPartition-def isPartition-wrt-def sorted-sublist-wrt-cons')

**subgoal** —  $lo < p=hi$

**using** *assms* **by** (simp add: isPartition-def isPartition-wrt-def sorted-sublist-wrt-snoc')

```

subgoal — lo < p < hi
using assms
apply (rewrite merge-sorted-wrt-partitions-between'[where p=p])
by auto
done
qed
done

```

**theorem** *merge-sorted-map-partitions*:

```

⟨(∧ x y z. [R (h x) (h y); R (h y) (h z)] ⇒ R (h x) (h z)) ⇒
  isPartition-map R h xs lo hi p ⇒
  sorted-sublist-map R h xs lo (p - Suc 0) ⇒ sorted-sublist-map R h xs (Suc p) hi ⇒
  lo ≤ hi ⇒ lo ≤ p ⇒ p ≤ hi ⇒ hi < length xs ⇒
  sorted-sublist-map R h xs lo hi⟩
apply (rule merge-sorted-wrt-partitions) apply auto
by (simp add: merge-sorted-wrt-partitions isPartition-map-trans)

```

**lemma** *partition-wrt-extend*:

```

⟨isPartition-wrt R xs lo' hi' p ⇒
  hi < length xs ⇒
  lo ≤ lo' ⇒ lo' ≤ hi ⇒ hi' ≤ hi ⇒
  lo' ≤ p ⇒ p ≤ hi' ⇒
  (∧ i. lo ≤ i ⇒ i < lo' ⇒ R (xs!i) (xs!p)) ⇒
  (∧ j. hi' < j ⇒ j ≤ hi ⇒ R (xs!p) (xs!j)) ⇒
  isPartition-wrt R xs lo hi p⟩
unfolding isPartition-wrt-def
apply auto
subgoal by (meson not-le)
subgoal by (metis nat-le-eq-or-lt nat-le-linear)
done

```

**lemma** *partition-map-extend*:

```

⟨isPartition-map R h xs lo' hi' p ⇒
  hi < length xs ⇒
  lo ≤ lo' ⇒ lo' ≤ hi ⇒ hi' ≤ hi ⇒
  lo' ≤ p ⇒ p ≤ hi' ⇒
  (∧ i. lo ≤ i ⇒ i < lo' ⇒ R (h (xs!i)) (h (xs!p))) ⇒
  (∧ j. hi' < j ⇒ j ≤ hi ⇒ R (h (xs!p)) (h (xs!j))) ⇒
  isPartition-map R h xs lo hi p⟩
by (auto simp add: partition-wrt-extend)

```

**lemma** *isPartition-empty*:

```

⟨(∧ j. [lo < j; j ≤ hi] ⇒ R (xs ! lo) (xs ! j)) ⇒
  isPartition-wrt R xs lo hi lo⟩
by (auto simp add: isPartition-wrt-def)

```

**lemma** *take-ext*:

```

⟨(∀ i < k. xs ! i = xs' ! i) ⇒
  k < length xs ⇒ k < length xs' ⇒
  take k xs' = take k xs⟩
by (simp add: nth-take-lemma)

```

**lemma** *drop-ext'*:

$\langle (\forall i. i \geq k \wedge i < \text{length } xs \longrightarrow xs!i = xs!i) \implies$   
 $0 < k \implies xs \neq [] \implies \text{— These corner cases will be dealt with in the next lemma}$   
 $\text{length } xs' = \text{length } xs \implies$   
 $\text{drop } k \text{ } xs' = \text{drop } k \text{ } xs \rangle$   
**apply** (*rewrite in*  $\langle \text{drop } - \sqsupset = - \rangle \text{List.rev-rev-ident[symmetric]}$ )  
**apply** (*rewrite in*  $\langle - = \text{drop } - \sqsupset \rangle \text{List.rev-rev-ident[symmetric]}$ )  
**apply** (*rewrite in*  $\langle \sqsupset = - \rangle \text{List.drop-rev}$ )  
**apply** (*rewrite in*  $\langle - = \sqsupset \rangle \text{List.drop-rev}$ )  
**apply** *simp*  
**apply** (*rule take-ext*)  
**by** (*auto simp add: nth-rev*)

**lemma** *drop-ext*:

$\langle (\forall i. i \geq k \wedge i < \text{length } xs \longrightarrow xs!i = xs!i) \implies$   
 $\text{length } xs' = \text{length } xs \implies$   
 $\text{drop } k \text{ } xs' = \text{drop } k \text{ } xs \rangle$   
**apply** (*cases xs*)  
**apply** *auto*  
**apply** (*cases k*)  
**subgoal by** (*simp add: nth-equalityI*)  
**subgoal apply** (*rule drop-ext'*) **by** *auto*  
**done**

**lemma** *sublist-ext'*:

$\langle (\forall i. lo \leq i \wedge i \leq hi \longrightarrow xs!i = xs!i) \implies$   
 $\text{length } xs' = \text{length } xs \implies$   
 $lo \leq hi \implies \text{Suc } hi < \text{length } xs \implies$   
 $\text{sublist } xs' \text{ } lo \text{ } hi = \text{sublist } xs \text{ } lo \text{ } hi \rangle$   
**apply** (*simp add: sublist-def*)  
**apply** (*rule take-ext*)  
**by** *auto*

**lemma** *lt-Suc*:  $\langle (a < b) = (\text{Suc } a = b \vee \text{Suc } a < b) \rangle$

**by** *auto*

**lemma** *sublist-until-end-eq-drop*:  $\langle \text{Suc } hi = \text{length } xs \implies \text{sublist } xs \text{ } lo \text{ } hi = \text{drop } lo \text{ } xs \rangle$

**by** (*simp add: sublist-def*)

**lemma** *sublist-ext*:

$\langle (\forall i. lo \leq i \wedge i \leq hi \longrightarrow xs!i = xs!i) \implies$   
 $\text{length } xs' = \text{length } xs \implies$   
 $lo \leq hi \implies hi < \text{length } xs \implies$   
 $\text{sublist } xs' \text{ } lo \text{ } hi = \text{sublist } xs \text{ } lo \text{ } hi \rangle$   
**apply** (*auto simp add: lt-Suc[where a=hi]*)  
**subgoal by** (*auto simp add: sublist-until-end-eq-drop drop-ext*)  
**subgoal by** (*auto simp add: sublist-ext'*)  
**done**

**lemma** *sorted-wrt-lower-sublist-still-sorted*:

**assumes**  $\langle \text{sorted-sublist-wrt } R \text{ } xs \text{ } lo \text{ } (lo' - \text{Suc } 0) \rangle$  **and**  
 $\langle lo \leq lo' \rangle$  **and**  $\langle lo' < \text{length } xs \rangle$  **and**

$\langle (\forall i. lo \leq i \wedge i < lo' \longrightarrow xs!i = xs!i) \rangle$  and  $\langle length\ xs' = length\ xs \rangle$   
**shows**  $\langle sorted\_sublist\_wrt\ R\ xs'\ lo\ (lo' - Suc\ 0) \rangle$   
**proof** –  
**have**  $l: \langle lo < lo' - 1 \vee lo \geq lo' - 1 \rangle$   
**by** *linarith*  
**show** *?thesis*  
**using**  $l$  **apply** *auto*  
**subgoal** —  $lo < lo' - 1$   
**apply** (*auto simp add: sorted\\_sublist\\_wrt-def*)  
**apply** (*rewrite sublist-ext[where xs=xs]*)  
**using** *assms* **by** (*auto simp add: sorted\\_sublist\\_wrt-def*)  
**subgoal** —  $lo \geq lo' - 1$   
**using** *assms* **by** (*auto simp add: sorted\\_sublist\\_wrt-le*)  
**done**  
**qed**

**lemma** *sorted-map-lower-sublist-still-sorted*:  
**assumes**  $\langle sorted\_sublist\_map\ R\ h\ xs\ lo\ (lo' - Suc\ 0) \rangle$  and  
 $\langle lo \leq lo' \rangle$  and  $\langle lo' < length\ xs \rangle$  and  
 $\langle (\forall i. lo \leq i \wedge i < lo' \longrightarrow xs!i = xs!i) \rangle$  and  $\langle length\ xs' = length\ xs \rangle$   
**shows**  $\langle sorted\_sublist\_map\ R\ h\ xs'\ lo\ (lo' - Suc\ 0) \rangle$   
**using** *assms* **by** (*rule sorted-wrt-lower-sublist-still-sorted*)

**lemma** *sorted-wrt-upper-sublist-still-sorted*:  
**assumes**  $\langle sorted\_sublist\_wrt\ R\ xs\ (hi' + 1)\ hi \rangle$  and  
 $\langle lo \leq lo' \rangle$  and  $\langle hi < length\ xs \rangle$  and  
 $\langle \forall j. hi' < j \wedge j \leq hi \longrightarrow xs!j = xs!j \rangle$  and  $\langle length\ xs' = length\ xs \rangle$   
**shows**  $\langle sorted\_sublist\_wrt\ R\ xs'\ (hi' + 1)\ hi \rangle$   
**proof** –  
**have**  $l: \langle hi' + 1 < hi \vee hi' + 1 \geq hi \rangle$   
**by** *linarith*  
**show** *?thesis*  
**using**  $l$  **apply** *auto*  
**subgoal** —  $hi' + 1 < h$   
**apply** (*auto simp add: sorted\\_sublist\\_wrt-def*)  
**apply** (*rewrite sublist-ext[where xs=xs]*)  
**using** *assms* **by** (*auto simp add: sorted\\_sublist\\_wrt-def*)  
**subgoal** —  $hi \leq hi' + 1$   
**using** *assms* **by** (*auto simp add: sorted\\_sublist\\_wrt-le*)  
**done**  
**qed**

**lemma** *sorted-map-upper-sublist-still-sorted*:  
**assumes**  $\langle sorted\_sublist\_map\ R\ h\ xs\ (hi' + 1)\ hi \rangle$  and  
 $\langle lo \leq lo' \rangle$  and  $\langle hi < length\ xs \rangle$  and  
 $\langle \forall j. hi' < j \wedge j \leq hi \longrightarrow xs!j = xs!j \rangle$  and  $\langle length\ xs' = length\ xs \rangle$   
**shows**  $\langle sorted\_sublist\_map\ R\ h\ xs'\ (hi' + 1)\ hi \rangle$   
**using** *assms* **by** (*rule sorted-wrt-upper-sublist-still-sorted*)

The specification of the partition function

**definition** *partition-spec* ::  $\langle ('b \Rightarrow 'b \Rightarrow bool) \Rightarrow ('a \Rightarrow 'b) \Rightarrow 'a\ list \Rightarrow nat \Rightarrow nat \Rightarrow 'a\ list \Rightarrow nat \Rightarrow bool \rangle$  **where**  
 $\langle partition\_spec\ R\ h\ xs\ lo\ hi\ xs'\ p \equiv$   
 $mset\ xs' = mset\ xs \wedge$  — The list is a permutation  
 $isPartition\_map\ R\ h\ xs'\ lo\ hi\ p \wedge$  — We have a valid partition on the resulting list

$lo \leq p \wedge p \leq hi \wedge \text{---}$  The partition index is in bounds  
 $(\forall i. i < lo \longrightarrow xs!i = xs!i) \wedge (\forall i. hi < i \wedge i < length\ xs' \longrightarrow xs!i = xs!i)$  — Everything else is unchanged.

**lemma** *in-set-take-conv-nth*:

$\langle x \in set\ (take\ n\ xs) \longleftrightarrow (\exists m < min\ n\ (length\ xs). xs!\ m = x) \rangle$   
**by** (*metis in-set-conv-nth length-take min.commute min.strict-boundedE nth-take*)

**lemma** *mset-drop-upto*:  $\langle mset\ (drop\ a\ N) = \{\#N!i. i \in \# mset\text{-}set\ \{a..<length\ N\}\#\} \rangle$

**proof** (*induction N arbitrary: a*)

**case** *Nil*

**then show** *?case by simp*

**next**

**case** (*Cons c N*)

**have** *upt*:  $\langle \{0..<Suc\ (length\ N)\} = insert\ 0\ \{1..<Suc\ (length\ N)\} \rangle$

**by** *auto*

**then have** *H*:  $\langle mset\text{-}set\ \{0..<Suc\ (length\ N)\} = add\text{-}mset\ 0\ (mset\text{-}set\ \{1..<Suc\ (length\ N)\}) \rangle$

**unfolding** *upt by auto*

**have** *mset-case-Suc*:  $\langle \{\#case\ x\ of\ 0 \Rightarrow c \mid Suc\ x \Rightarrow N!\ x.\ x \in \# mset\text{-}set\ \{Suc\ a..<Suc\ b\}\#\} = \{\#N!\ (x-1) . x \in \# mset\text{-}set\ \{Suc\ a..<Suc\ b\}\#\} \rangle$  **for** *a b*

**by** (*rule image-mset-cong*) (*auto split: nat.splits*)

**have** *Suc-Suc*:  $\langle \{Suc\ a..<Suc\ b\} = Suc\ ' \{a..<b\} \rangle$  **for** *a b*

**by** *auto*

**then have** *mset-set-Suc-Suc*:  $\langle mset\text{-}set\ \{Suc\ a..<Suc\ b\} = \{\#Suc\ n. n \in \# mset\text{-}set\ \{a..<b\}\#\} \rangle$  **for** *a b*

**unfolding** *Suc-Suc by* (*subst image-mset-mset-set[symmetric]*) *auto*

**have** *\**:  $\langle \{\#N!\ (x-Suc\ 0) . x \in \# mset\text{-}set\ \{Suc\ a..<Suc\ b\}\#\} = \{\#N!\ x . x \in \# mset\text{-}set\ \{a..<b\}\#\} \rangle$  **for** *a b*

**by** (*auto simp add: mset-set-Suc-Suc multiset.map-comp comp-def*)

**show** *?case*

**apply** (*cases a*)

**using** *Cons[of 0] Cons by* (*auto simp: nth-Cons drop-Cons H mset-case-Suc \**)

**qed**

**lemma** *mathias*:

**assumes**

*Perm*:  $\langle mset\ xs' = mset\ xs \rangle$

**and** *I*:  $\langle lo \leq i \rangle \langle i \leq hi \rangle \langle xs!i = x \rangle$

**and** *Bounds*:  $\langle hi < length\ xs \rangle$

**and** *Fix*:  $\langle \bigwedge i. i < lo \implies xs!i = xs!i \rangle \langle \bigwedge j. \llbracket hi < j; j < length\ xs \rrbracket \implies xs!j = xs!j \rangle$

**shows**  $\langle \exists j. lo \leq j \wedge j \leq hi \wedge xs!j = x \rangle$

**proof** —

**define** *xs1 xs2 xs3 xs1' xs2' xs3'* **where**

$\langle xs1 = take\ lo\ xs \rangle$  **and**

$\langle xs2 = take\ (Suc\ hi - lo)\ (drop\ lo\ xs) \rangle$  **and**

$\langle xs3 = drop\ (Suc\ hi)\ xs \rangle$  **and**

$\langle xs1' = take\ lo\ xs' \rangle$  **and**

$\langle xs2' = take\ (Suc\ hi - lo)\ (drop\ lo\ xs') \rangle$  **and**

$\langle xs3' = drop\ (Suc\ hi)\ xs' \rangle$

**have** [*simp*]:  $\langle length\ xs' = length\ xs \rangle$

**using** *Perm by* (*auto dest: mset-eq-length*)

**have** [*simp*]:  $\langle mset\ xs1 = mset\ xs1' \rangle$

**using** *Fix(1) unfolding xs1-def xs1'-def*

**by** (*metis Perm le-cases mset-eq-length nth-take-lemma take-all*)

**have** [*simp*]:  $\langle mset\ xs3 = mset\ xs3' \rangle$

```

    using Fix(2) unfolding xs3-def xs3'-def mset-drop-upto
    by (auto intro: image-mset-cong)
  have ⟨xs = xs1 @ xs2 @ xs3⟩ ⟨xs' = xs1' @ xs2' @ xs3'⟩
    using I unfolding xs1-def xs2-def xs3-def xs1'-def xs2'-def xs3'-def
    by (metis append.assoc append-take-drop-id le-SucI le-add-diff-inverse order-trans take-add)+
  moreover have ⟨xs ! i = xs2 ! (i - lo)⟩ ⟨i ≥ length xs1⟩
    using I Bounds unfolding xs2-def xs1-def by (auto simp: nth-take min-def)
  moreover have ⟨x ∈ set xs2'⟩
    using I Bounds unfolding xs2'-def
    by (auto simp: in-set-take-conv-nth
        intro!: exI[of - ⟨i - lo⟩])
  ultimately have ⟨x ∈ set xs2⟩
    using Perm I by (auto dest: mset-eq-setD)
  then obtain j where ⟨xs ! (lo + j) = x⟩ ⟨j ≤ hi - lo⟩
    unfolding in-set-conv-nth xs2-def
    by auto
  then show ?thesis
    using Bounds I
    by (auto intro: exI[of - ⟨lo+j⟩])
qed

```

If we fix the left and right rest of two permutated lists, then the sublists are also permutations.

But we only need that the sets are equal.

**lemma** *mset-sublist-incl*:

```

  assumes Perm: ⟨mset xs' = mset xs⟩
    and Fix: ⟨ $\bigwedge i. i < lo \implies xs ! i = xs' ! i$ ⟩  $\langle \bigwedge j. \llbracket hi < j; j < length\ xs \rrbracket \implies xs ! j = xs' ! j \rangle$ 
    and bounds: ⟨lo ≤ hi⟩ ⟨hi < length xs⟩
  shows ⟨set (sublist xs' lo hi) ⊆ set (sublist xs lo hi)⟩
proof
  fix x
  assume ⟨x ∈ set (sublist xs' lo hi)⟩
  then have ⟨ $\exists i. lo \leq i \wedge i \leq hi \wedge xs' ! i = x$ ⟩
    by (metis assms(1) bounds(1) bounds(2) size-mset sublist-el')
  then obtain i where I: ⟨lo ≤ i⟩ ⟨i ≤ hi⟩ ⟨xs' ! i = x⟩ by blast
  have ⟨ $\exists j. lo \leq j \wedge j \leq hi \wedge xs ! j = x$ ⟩
    using Perm I bounds(2) Fix by (rule mathias, auto)
  then show ⟨x ∈ set (sublist xs lo hi)⟩
    by (simp add: bounds(1) bounds(2) sublist-el')
qed

```

**lemma** *mset-sublist-eq*:

```

  assumes ⟨mset xs' = mset xs⟩
    and  $\langle \bigwedge i. i < lo \implies xs' ! i = xs ! i \rangle$ 
    and  $\langle \bigwedge j. \llbracket hi < j; j < length\ xs \rrbracket \implies xs' ! j = xs ! j \rangle$ 
    and bounds: ⟨lo ≤ hi⟩ ⟨hi < length xs⟩
  shows ⟨set (sublist xs' lo hi) = set (sublist xs lo hi)⟩
proof
  show ⟨set (sublist xs' lo hi) ⊆ set (sublist xs lo hi)⟩
    apply (rule mset-sublist-incl)
    using assms by auto
  show ⟨set (sublist xs lo hi) ⊆ set (sublist xs' lo hi)⟩
    apply (rule mset-sublist-incl)
    by (metis assms size-mset)+
qed

```



Our abstract recursive quicksort procedure. We abstract over a partition procedure.

**definition** *quicksort* ::  $\langle ('b \Rightarrow 'b \Rightarrow \text{bool}) \Rightarrow ('a \Rightarrow 'b) \Rightarrow \text{nat} \times \text{nat} \times 'a \text{ list} \Rightarrow 'a \text{ list nres} \rangle$  **where**  
 $\langle \text{quicksort } R \ h = (\lambda(lo, hi, xs0). \text{ do } \{$   
 $\quad \text{RECT } (\lambda f \ (lo, hi, xs). \text{ do } \{$   
 $\quad \quad \text{ASSERT}(lo \leq hi \wedge hi < \text{length } xs \wedge \text{mset } xs = \text{mset } xs0);$  — Premise for a partition function  
 $\quad \quad (xs, p) \leftarrow \text{SPEC}(\text{uncurry } (\text{partition-spec } R \ h \ xs \ lo \ hi));$  — Abstract partition function  
 $\quad \quad \text{ASSERT}(\text{mset } xs = \text{mset } xs0);$   
 $\quad \quad xs \leftarrow (\text{if } p-1 \leq lo \text{ then RETURN } xs \text{ else } f \ (lo, p-1, xs));$   
 $\quad \quad \text{ASSERT}(\text{mset } xs = \text{mset } xs0);$   
 $\quad \quad \text{if } hi \leq p+1 \text{ then RETURN } xs \text{ else } f \ (p+1, hi, xs)$   
 $\quad \quad \}) \ (lo, hi, xs0)$   
 $\quad \quad \}) \rangle$

As premise for quicksort, we only need that the indices are ok.

**definition** *quicksort-pre* ::  $\langle ('b \Rightarrow 'b \Rightarrow \text{bool}) \Rightarrow ('a \Rightarrow 'b) \Rightarrow 'a \text{ list} \Rightarrow \text{nat} \Rightarrow \text{nat} \Rightarrow 'a \text{ list} \Rightarrow \text{bool} \rangle$   
**where**  
 $\langle \text{quicksort-pre } R \ h \ xs0 \ lo \ hi \ xs \equiv lo \leq hi \wedge hi < \text{length } xs \wedge \text{mset } xs = \text{mset } xs0 \rangle$

**definition** *quicksort-post* ::  $\langle ('b \Rightarrow 'b \Rightarrow \text{bool}) \Rightarrow ('a \Rightarrow 'b) \Rightarrow \text{nat} \Rightarrow \text{nat} \Rightarrow 'a \text{ list} \Rightarrow 'a \text{ list} \Rightarrow \text{bool} \rangle$   
**where**  
 $\langle \text{quicksort-post } R \ h \ lo \ hi \ xs \ xs' \equiv$   
 $\quad \text{mset } xs' = \text{mset } xs \wedge$   
 $\quad \text{sorted-sublist-map } R \ h \ xs' \ lo \ hi \wedge$   
 $\quad (\forall i. i < lo \longrightarrow xs'!i = xs!i) \wedge$   
 $\quad (\forall j. hi < j \wedge j < \text{length } xs \longrightarrow xs'!j = xs!j) \rangle$

Convert Pure to HOL

**lemma** *quicksort-postI*:  
 $\langle \llbracket \text{mset } xs' = \text{mset } xs; \text{sorted-sublist-map } R \ h \ xs' \ lo \ hi; (\bigwedge i. \llbracket i < lo \rrbracket \implies xs'!i = xs!i); (\bigwedge j. \llbracket hi < j; j < \text{length } xs \rrbracket \implies xs'!j = xs!j) \rrbracket \implies \text{quicksort-post } R \ h \ lo \ hi \ xs \ xs' \rangle$   
**by** (auto simp add: quicksort-post-def)

The first case for the correctness proof of (abstract) quicksort: We assume that we called the partition function, and we have  $p - (1::'a) \leq lo$  and  $hi \leq p + (1::'a)$ .

**lemma** *quicksort-correct-case1*:  
**assumes** *trans*:  $\langle \bigwedge x \ y \ z. \llbracket R \ (h \ x) \ (h \ y); R \ (h \ y) \ (h \ z) \rrbracket \implies R \ (h \ x) \ (h \ z) \rangle$  **and** *lin*:  $\langle \bigwedge x \ y. x \neq y \implies R \ (h \ x) \ (h \ y) \vee R \ (h \ y) \ (h \ x) \rangle$   
**and pre**:  $\langle \text{quicksort-pre } R \ h \ xs0 \ lo \ hi \ xs \rangle$   
**and part**:  $\langle \text{partition-spec } R \ h \ xs \ lo \ hi \ xs' \ p \rangle$   
**and ifs**:  $\langle p-1 \leq lo \rangle \langle hi \leq p+1 \rangle$   
**shows**  $\langle \text{quicksort-post } R \ h \ lo \ hi \ xs \ xs' \rangle$   
**proof** —

First boilerplate code step: 'unfold' the HOL definitions in the assumptions and convert them to Pure

**have pre**:  $\langle lo \leq hi \rangle \langle hi < \text{length } xs \rangle$   
**using pre by** (auto simp add: quicksort-pre-def)  
  
**have part**:  $\langle \text{mset } xs' = \text{mset } xs \rangle \text{ True}$   
 $\langle \text{isPartition-map } R \ h \ xs' \ lo \ hi \ p \rangle \langle lo \leq p \rangle \langle p \leq hi \rangle$   
 $\langle \bigwedge i. i < lo \implies xs'!i = xs!i \rangle \langle \bigwedge i. \llbracket hi < i; i < \text{length } xs \rrbracket \implies xs'!i = xs!i \rangle$   
**using part by** (auto simp add: partition-spec-def)

```

have sorted-lower: ⟨sorted-sublist-map R h xs' lo (p - Suc 0)⟩
proof -
  show ?thesis
    apply (rule sorted-sublist-wrt-le)
    subgoal using ifs(1) by auto
    subgoal using ifs(1) mset-eq-length part(1) pre(1) pre(2) by fastforce
    done
qed

have sorted-upper: ⟨sorted-sublist-map R h xs' (Suc p) hi⟩
proof -
  show ?thesis
    apply (rule sorted-sublist-wrt-le)
    subgoal using ifs(2) by auto
    subgoal using ifs(1) mset-eq-length part(1) pre(1) pre(2) by fastforce
    done
qed

have sorted-middle: ⟨sorted-sublist-map R h xs' lo hi⟩
proof -
  show ?thesis
    apply (rule merge-sorted-map-partitions[where p=p])
    subgoal by (rule trans)
    subgoal by (rule part)
    subgoal by (rule sorted-lower)
    subgoal by (rule sorted-upper)
    subgoal using pre(1) by auto
    subgoal by (simp add: part(4))
    subgoal by (simp add: part(5))
    subgoal by (metis part(1) pre(2) size-mset)
    done
qed

show ?thesis
proof (intro quicksort-postI)
  show ⟨mset xs' = mset xs⟩
    by (simp add: part(1))
next
  show ⟨sorted-sublist-map R h xs' lo hi⟩
    by (rule sorted-middle)
next
  show ⟨ $\bigwedge i. i < lo \implies xs' ! i = xs ! i$ ⟩
    using part(6) by blast
next
  show ⟨ $\bigwedge j. \llbracket hi < j; j < length\ xs \rrbracket \implies xs' ! j = xs ! j$ ⟩
    by (metis part(1) part(7) size-mset)
qed
qed

```

In the second case, we have to show that the precondition still holds for (p+1, hi, x') after the partition.

**lemma** *quicksort-correct-case2*:

```

assumes
  pre: ⟨quicksort-pre R h xs0 lo hi xs⟩
  and part: ⟨partition-spec R h xs lo hi xs' p⟩

```

**and** *ifs*:  $\langle \neg hi \leq p + 1 \rangle$   
**shows**  $\langle quicksort\text{-}pre\ R\ h\ xs0\ (Suc\ p)\ hi\ xs' \rangle$   
**proof** –

First boilerplate code step: 'unfold' the HOL definitions in the assumptions and convert them to Pure

**have** *pre*:  $\langle lo \leq hi \rangle \langle hi < length\ xs \rangle \langle mset\ xs = mset\ xs0 \rangle$   
**using** *pre* **by** (*auto simp add: quicksort-pre-def*)  
**have** *part*:  $\langle mset\ xs' = mset\ xs \rangle\ True$   
 $\langle isPartition\text{-}map\ R\ h\ xs'\ lo\ hi\ p \rangle \langle lo \leq p \rangle \langle p \leq hi \rangle$   
 $\langle \bigwedge i. i < lo \implies xs!i = xs!i \rangle \langle \bigwedge i. \llbracket hi < i; i < length\ xs \rrbracket \implies xs!i = xs!i \rangle$   
**using** *part* **by** (*auto simp add: partition-spec-def*)  
**show** *?thesis*  
**unfolding** *quicksort-pre-def*  
**proof** (*intro conjI*)  
**show**  $\langle Suc\ p \leq hi \rangle$   
**using** *ifs* **by** *linarith*  
**show**  $\langle hi < length\ xs' \rangle$   
**by** (*metis part(1) pre(2) size-mset*)  
**show**  $\langle mset\ xs' = mset\ xs0 \rangle$   
**using** *pre(3) part(1)* **by** (*auto dest: mset-eq-setD*)  
**qed**  
**qed**

**lemma** *quicksort-post-set*:

**assumes**  $\langle quicksort\text{-}post\ R\ h\ lo\ hi\ xs\ xs' \rangle$   
**and** *bounds*:  $\langle lo \leq hi \rangle \langle hi < length\ xs \rangle$   
**shows**  $\langle set\ (sublist\ xs'\ lo\ hi) = set\ (sublist\ xs\ lo\ hi) \rangle$   
**proof** –  
**have**  $\langle mset\ xs' = mset\ xs \rangle \langle \bigwedge i. i < lo \implies xs!i = xs!i \rangle \langle \bigwedge j. \llbracket hi < j; j < length\ xs \rrbracket \implies xs!j = xs!j \rangle$   
**using** *assms* **by** (*auto simp add: quicksort-post-def*)  
**then show** *?thesis*  
**using** *bounds* **by** (*rule mset-sublist-eq, auto*)  
**qed**

In the third case, we have run quicksort recursively on  $(p+1, hi, xs')$  after the partition, with  $hi \leq p+1$  and  $p-1 \leq lo$ .

**lemma** *quicksort-correct-case3*:

**assumes** *trans*:  $\langle \bigwedge x\ y\ z. \llbracket R\ (h\ x)\ (h\ y); R\ (h\ y)\ (h\ z) \rrbracket \implies R\ (h\ x)\ (h\ z) \rangle$  **and** *lin*:  $\langle \bigwedge x\ y. x \neq y \implies R\ (h\ x)\ (h\ y) \vee R\ (h\ y)\ (h\ x) \rangle$   
**and** *pre*:  $\langle quicksort\text{-}pre\ R\ h\ xs0\ lo\ hi\ xs \rangle$   
**and** *part*:  $\langle partition\text{-}spec\ R\ h\ xs\ lo\ hi\ xs'\ p \rangle$   
**and** *ifs*:  $\langle p - Suc\ 0 \leq lo \rangle \langle \neg hi \leq Suc\ p \rangle$   
**and** *IH1'*:  $\langle quicksort\text{-}post\ R\ h\ (Suc\ p)\ hi\ xs'\ xs'' \rangle$   
**shows**  $\langle quicksort\text{-}post\ R\ h\ lo\ hi\ xs\ xs'' \rangle$   
**proof** –

First boilerplate code step: 'unfold' the HOL definitions in the assumptions and convert them to Pure

**have** *pre*:  $\langle lo \leq hi \rangle \langle hi < length\ xs \rangle \langle mset\ xs = mset\ xs0 \rangle$   
**using** *pre* **by** (*auto simp add: quicksort-pre-def*)  
**have** *part*:  $\langle mset\ xs' = mset\ xs \rangle\ True$   
 $\langle isPartition\text{-}map\ R\ h\ xs'\ lo\ hi\ p \rangle \langle lo \leq p \rangle \langle p \leq hi \rangle$

```

  ⟨ $\bigwedge i. i < lo \implies xs!i = xs!i$ ⟩ ⟨ $\bigwedge i. \llbracket hi < i; i < \text{length } xs \rrbracket \implies xs!i = xs!i$ ⟩
  using part by (auto simp add: partition-spec-def)
have IH1: ⟨ $\text{mset } xs'' = \text{mset } xs'$ ⟩ ⟨ $\text{sorted-sublist-map } R \ h \ xs'' \ (Suc \ p) \ hi$ ⟩
  ⟨ $\bigwedge i. i < Suc \ p \implies xs''!i = xs'!i$ ⟩ ⟨ $\bigwedge j. \llbracket hi < j; j < \text{length } xs' \rrbracket \implies xs''!j = xs'!j$ ⟩
  using IH1' by (auto simp add: quicksort-post-def)
note IH1-perm = quicksort-post-set[OF IH1]

have still-partition: ⟨ $\text{isPartition-map } R \ h \ xs'' \ lo \ hi \ p$ ⟩
proof(intro isPartition-wrtI)
  fix i assume ⟨ $lo \leq i$ ⟩ ⟨ $i < p$ ⟩
  show ⟨ $R \ (h \ (xs''!i)) \ (h \ (xs''!p))$ ⟩

```

This holds because this part hasn't changed

```

  using IH1(3) ⟨ $i < p$ ⟩ ⟨ $lo \leq i$ ⟩ isPartition-wrt-def part(3) by fastforce
next
  fix j assume ⟨ $p < j$ ⟩ ⟨ $j \leq hi$ ⟩

```

Obtain the position  $posJ$  where  $xs''!j$  was stored in  $xs'$ .

```

  have ⟨ $xs''!j \in \text{set } (\text{sublist } xs'' \ (Suc \ p) \ hi)$ ⟩
  by (metis IH1(1) Suc-leI ⟨ $j \leq hi$ ⟩ ⟨ $p < j$ ⟩ less-le-trans mset-eq-length part(1) pre(2) sublist-el')
  then have ⟨ $xs''!j \in \text{set } (\text{sublist } xs' \ (Suc \ p) \ hi)$ ⟩
  by (metis IH1-perm ifs(2) nat-le-linear part(1) pre(2) size-mset)
  then have ⟨ $\exists posJ. Suc \ p \leq posJ \wedge posJ \leq hi \wedge xs''!j = xs'!posJ$ ⟩
  by (metis Suc-leI ⟨ $j \leq hi$ ⟩ ⟨ $p < j$ ⟩ less-le-trans part(1) pre(2) size-mset sublist-el')
  then obtain posJ :: nat where PosJ: ⟨ $Suc \ p \leq posJ$ ⟩ ⟨ $posJ \leq hi$ ⟩ ⟨ $xs''!j = xs'!posJ$ ⟩ by blast

  then show ⟨ $R \ (h \ (xs''!p)) \ (h \ (xs''!j))$ ⟩
  by (metis IH1(3) Suc-le-lessD isPartition-wrt-def lessI part(3))
qed

```

```

have sorted-lower: ⟨ $\text{sorted-sublist-map } R \ h \ xs'' \ lo \ (p - Suc \ 0)$ ⟩
proof -
  show ?thesis
  apply (rule sorted-sublist-wrt-le)
  subgoal by (simp add: ifs(1))
  subgoal using IH1(1) mset-eq-length part(1) part(5) pre(2) by fastforce
  done
qed

```

note sorted-upper = IH1(2)

```

have sorted-middle: ⟨ $\text{sorted-sublist-map } R \ h \ xs'' \ lo \ hi$ ⟩
proof -
  show ?thesis
  apply (rule merge-sorted-map-partitions[where p=p])
  subgoal by (rule trans)
  subgoal by (rule still-partition)
  subgoal by (rule sorted-lower)
  subgoal by (rule sorted-upper)
  subgoal using pre(1) by auto
  subgoal by (simp add: part(4))
  subgoal by (simp add: part(5))
  subgoal by (metis IH1(1) part(1) pre(2) size-mset)
  done
qed

```

```

show ?thesis
proof (intro quicksort-postI)
  show  $\langle \text{mset } xs'' = \text{mset } xs \rangle$ 
    using part(1) IH1(1) by auto — I was faster than sledgehammer :-
next
  show  $\langle \text{sorted-sublist-map } R \text{ } h \text{ } xs'' \text{ } lo \text{ } hi \rangle$ 
    by (rule sorted-middle)
next
  show  $\langle \bigwedge i. i < lo \implies xs'' ! i = xs ! i \rangle$ 
    using IH1(3) le-SucI part(4) part(6) by auto
next show  $\langle \bigwedge j. hi < j \implies j < \text{length } xs \implies xs'' ! j = xs ! j \rangle$ 
    by (metis IH1(4) part(1) part(7) size-mset)
qed
qed

```

In the 4th case, we have to show that the premise holds for  $(lo, p - (1::'b), xs')$ , in case  $\neg p - (1::'a) \leq lo$

Analogous to case 2.

**lemma** *quicksort-correct-case4*:

```

assumes
  pre:  $\langle \text{quicksort-pre } R \text{ } h \text{ } xs0 \text{ } lo \text{ } hi \text{ } xs \rangle$ 
  and part:  $\langle \text{partition-spec } R \text{ } h \text{ } xs \text{ } lo \text{ } hi \text{ } xs' \text{ } p \rangle$ 
  and ifs:  $\langle \neg p - \text{Suc } 0 \leq lo \rangle$ 
shows  $\langle \text{quicksort-pre } R \text{ } h \text{ } xs0 \text{ } lo \text{ } (p - \text{Suc } 0) \text{ } xs' \rangle$ 
proof —

```

First boilerplate code step: 'unfold' the HOL definitions in the assumptions and convert them to Pure

```

have pre:  $\langle lo \leq hi \rangle \langle hi < \text{length } xs \rangle \langle \text{mset } xs0 = \text{mset } xs \rangle$ 
  using pre by (auto simp add: quicksort-pre-def)
have part:  $\langle \text{mset } xs' = \text{mset } xs \rangle \text{True}$ 
   $\langle \text{isPartition-map } R \text{ } h \text{ } xs' \text{ } lo \text{ } hi \text{ } p \rangle \langle lo \leq p \rangle \langle p \leq hi \rangle$ 
   $\langle \bigwedge i. i < lo \implies xs' ! i = xs ! i \rangle \langle \bigwedge i. \llbracket hi < i; i < \text{length } xs \rrbracket \implies xs' ! i = xs ! i \rangle$ 
  using part by (auto simp add: partition-spec-def)

```

```

show ?thesis
unfolding quicksort-pre-def
proof (intro conjI)
  show  $\langle lo \leq p - \text{Suc } 0 \rangle$ 
    using ifs by linarith
  show  $\langle p - \text{Suc } 0 < \text{length } xs' \rangle$ 
    using mset-eq-length part(1) part(5) pre(2) by fastforce
  show  $\langle \text{mset } xs' = \text{mset } xs0 \rangle$ 
    using pre(3) part(1) by (auto dest: mset-eq-setD)
qed
qed

```

In the 5th case, we have run quicksort recursively on  $(lo, p-1, xs')$ .

**lemma** *quicksort-correct-case5*:

```

assumes trans:  $\langle \bigwedge x \text{ } y \text{ } z. \llbracket R \text{ } (h \text{ } x) \text{ } (h \text{ } y); R \text{ } (h \text{ } y) \text{ } (h \text{ } z) \rrbracket \implies R \text{ } (h \text{ } x) \text{ } (h \text{ } z) \rangle$  and lin:  $\langle \bigwedge x \text{ } y. x \neq y \implies R \text{ } (h \text{ } x) \text{ } (h \text{ } y) \vee R \text{ } (h \text{ } y) \text{ } (h \text{ } x) \rangle$ 
and pre:  $\langle \text{quicksort-pre } R \text{ } h \text{ } xs0 \text{ } lo \text{ } hi \text{ } xs \rangle$ 

```

```

    and part: ⟨partition-spec R h xs lo hi xs' p⟩
    and ifs: ⟨¬ p − Suc 0 ≤ lo⟩ ⟨hi ≤ Suc p⟩
    and IH1': ⟨quicksort-post R h lo (p − Suc 0) xs' xs''⟩
  shows ⟨quicksort-post R h lo hi xs xs''⟩
proof −

```

First boilerplate code step: 'unfold' the HOL definitions in the assumptions and convert them to Pure

```

  have pre: ⟨lo ≤ hi⟩ ⟨hi < length xs⟩
    using pre by (auto simp add: quicksort-pre-def)
  have part: ⟨mset xs' = mset xs⟩ True
    ⟨isPartition-map R h xs' lo hi p⟩ ⟨lo ≤ p⟩ ⟨p ≤ hi⟩
    ⟨∧ i. i < lo ⟹ xs'!i = xs!i⟩ ⟨∧ i. ⟦hi < i; i < length xs⟧ ⟹ xs'!i = xs!i⟩
    using part by (auto simp add: partition-spec-def)
  have IH1: ⟨mset xs'' = mset xs'⟩ ⟨sorted-sublist-map R h xs'' lo (p − Suc 0)⟩
    ⟨∧ i. i < lo ⟹ xs''!i = xs'!i⟩ ⟨∧ j. ⟦p − Suc 0 < j; j < length xs'⟧ ⟹ xs''!j = xs'!j⟩
    using IH1' by (auto simp add: quicksort-post-def)
  note IH1-perm = quicksort-post-set[OF IH1]

```

```

  have still-partition: ⟨isPartition-map R h xs'' lo hi p⟩
  proof (intro isPartition-wrtI)
    fix i assume ⟨lo ≤ i⟩ ⟨i < p⟩

```

Obtain the position  $posI$  where  $xs''!i$  was stored in  $xs'$ .

```

    have ⟨xs''!i ∈ set (sublist xs'' lo (p − Suc 0))⟩
      by (metis (no-types, lifting) IH1(1) Suc-leI Suc-pred ⟨i < p⟩ ⟨lo ≤ i⟩ le-less-trans less-imp-diff-less
mset-eq-length not-le not-less-zero part(1) part(5) pre(2) sublist-el')
    then have ⟨xs''!i ∈ set (sublist xs' lo (p − Suc 0))⟩
      by (metis IH1-perm ifs(1) le-less-trans less-imp-diff-less mset-eq-length nat-le-linear part(1)
part(5) pre(2))
    then have ⟨∃ posI. lo ≤ posI ∧ posI ≤ p − Suc 0 ∧ xs''!i = xs'!posI⟩
    proof − — sledgehammer
      have p − Suc 0 < length xs
        by (meson diff-le-self le-less-trans part(5) pre(2))
      then show ?thesis
        by (metis (no-types) ⟨xs''!i ∈ set (sublist xs' lo (p − Suc 0))⟩ ifs(1) mset-eq-length nat-le-linear
part(1) sublist-el')
    qed
    then obtain posI :: nat where PosI: ⟨lo ≤ posI⟩ ⟨posI ≤ p − Suc 0⟩ ⟨xs''!i = xs'!posI⟩ by blast
    then show ⟨R (h (xs''!i)) (h (xs''!p))⟩
      by (metis (no-types, lifting) IH1(4) ⟨i < p⟩ diff-Suc-less isPartition-wrt-def le-less-trans mset-eq-length
not-le not-less-eq part(1) part(3) part(5) pre(2) zero-less-Suc)
  next
    fix j assume ⟨p < j⟩ ⟨j ≤ hi⟩
    then show ⟨R (h (xs''!p)) (h (xs''!j))⟩

```

This holds because this part hasn't changed

```

    by (smt IH1(4) add-diff-cancel-left' add-diff-inverse-nat diff-Suc-eq-diff-pred diff-le-self ifs(1)
isPartition-wrt-def le-less-Suc-eq less-le-trans mset-eq-length nat-less-le part(1) part(3) part(4) plus-1-eq-Suc
pre(2))
  qed

```

```

  note sorted-lower = IH1(2)

```

```

have sorted-upper: ⟨sorted-sublist-map R h xs'' (Suc p) hi⟩
proof -
  show ?thesis
    apply (rule sorted-sublist-wrt-le)
    subgoal by (simp add: ifs(2))
    subgoal using IH1(1) mset-eq-length part(1) part(5) pre(2) by fastforce
    done
qed

```

```

have sorted-middle: ⟨sorted-sublist-map R h xs'' lo hi⟩
proof -
  show ?thesis
    apply (rule merge-sorted-map-partitions[where p=p])
    subgoal by (rule trans)
    subgoal by (rule still-partition)
    subgoal by (rule sorted-lower)
    subgoal by (rule sorted-upper)
    subgoal using pre(1) by auto
    subgoal by (simp add: part(4))
    subgoal by (simp add: part(5))
    subgoal by (metis IH1(1) part(1) pre(2) size-mset)
    done
qed

```

```

show ?thesis
proof (intro quicksort-postI)
  show ⟨mset xs'' = mset xs⟩
    by (simp add: IH1(1) part(1))
next
  show ⟨sorted-sublist-map R h xs'' lo hi⟩
    by (rule sorted-middle)
next
  show ⟨ $\bigwedge i. i < lo \implies xs'' ! i = xs ! i$ ⟩
    by (simp add: IH1(3) part(6))
next
  show ⟨ $\bigwedge j. hi < j \implies j < length\ xs \implies xs'' ! j = xs ! j$ ⟩
    by (metis IH1(4) diff-le-self dual-order.strict-trans2 mset-eq-length part(1) part(5) part(7))
qed
qed

```

In the 6th case, we have run quicksort recursively on (lo, p-1, xs'). We show the precondition on the second call on (p+1, hi, xs'')

```

lemma quicksort-correct-case6:
  assumes
    pre: ⟨quicksort-pre R h xs0 lo hi xs⟩
    and part: ⟨partition-spec R h xs lo hi xs' p⟩
    and ifs: ⟨ $\neg p - Suc\ 0 \leq lo$ ⟩  $\neg hi \leq Suc\ p$ 
    and IH1: ⟨quicksort-post R h lo (p - Suc 0) xs' xs''⟩
  shows ⟨quicksort-pre R h xs0 (Suc p) hi xs''⟩
proof -

```

First boilerplate code step: 'unfold' the HOL definitions in the assumptions and convert them to Pure

```

have pre: ⟨lo ≤ hi⟩ ⟨hi < length xs⟩ ⟨mset xs0 = mset xs⟩
  using pre by (auto simp add: quicksort-pre-def)
have part: ⟨mset xs' = mset xs⟩ True
  ⟨isPartition-map R h xs' lo hi p⟩ ⟨lo ≤ p⟩ ⟨p ≤ hi⟩
  ⟨∧ i. i < lo ⇒ xs!i = xs!i⟩ ⟨∧ i. [hi < i; i < length xs] ⇒ xs!i = xs!i⟩
  using part by (auto simp add: partition-spec-def)
have IH1: ⟨mset xs'' = mset xs'⟩ ⟨sorted-sublist-map R h xs'' lo (p - Suc 0)⟩
  ⟨∧ i. i < lo ⇒ xs''!i = xs!i⟩ ⟨∧ j. [p - Suc 0 < j; j < length xs] ⇒ xs''!j = xs!j⟩
  using IH1 by (auto simp add: quicksort-post-def)

show ?thesis
  unfolding quicksort-pre-def
proof (intro conjI)
  show ⟨Suc p ≤ hi⟩
    using ifs(2) by linarith
  show ⟨hi < length xs''⟩
    using IH1(1) mset-eq-length part(1) pre(2) by fastforce
  show ⟨mset xs'' = mset xs0⟩
    using pre(3) part(1) IH1(1) by (auto dest: mset-eq-setD)
qed
qed

```

In the 7th (and last) case, we have run quicksort recursively on (lo, p-1, xs'). We show the postcondition on the second call on (p+1, hi, xs'')

**lemma** *quicksort-correct-case7*:

```

assumes trans: ⟨∧ x y z. [R (h x) (h y); R (h y) (h z)] ⇒ R (h x) (h z)⟩ and lin: ⟨∧ x y. x ≠ y ⇒
R (h x) (h y) ∨ R (h y) (h x)⟩
and pre: ⟨quicksort-pre R h xs0 lo hi xs⟩
and part: ⟨partition-spec R h xs lo hi xs' p⟩
and ifs: ⟨¬ p - Suc 0 ≤ lo⟩ ⟨¬ hi ≤ Suc p⟩
and IH1': ⟨quicksort-post R h lo (p - Suc 0) xs' xs''⟩
and IH2': ⟨quicksort-post R h (Suc p) hi xs'' xs'''⟩
shows ⟨quicksort-post R h lo hi xs xs'''⟩

```

**proof** –

First boilerplate code step: 'unfold' the HOL definitions in the assumptions and convert them to Pure

```

have pre: ⟨lo ≤ hi⟩ ⟨hi < length xs⟩
  using pre by (auto simp add: quicksort-pre-def)
have part: ⟨mset xs' = mset xs⟩ True
  ⟨isPartition-map R h xs' lo hi p⟩ ⟨lo ≤ p⟩ ⟨p ≤ hi⟩
  ⟨∧ i. i < lo ⇒ xs!i = xs!i⟩ ⟨∧ i. [hi < i; i < length xs] ⇒ xs!i = xs!i⟩
  using part by (auto simp add: partition-spec-def)
have IH1: ⟨mset xs'' = mset xs'⟩ ⟨sorted-sublist-map R h xs'' lo (p - Suc 0)⟩
  ⟨∧ i. i < lo ⇒ xs''!i = xs!i⟩ ⟨∧ j. [p - Suc 0 < j; j < length xs] ⇒ xs''!j = xs!j⟩
  using IH1' by (auto simp add: quicksort-post-def)
note IH1-perm = quicksort-post-set[OF IH1]
have IH2: ⟨mset xs''' = mset xs''⟩ ⟨sorted-sublist-map R h xs''' (Suc p) hi⟩
  ⟨∧ i. i < Suc p ⇒ xs'''!i = xs''!i⟩ ⟨∧ j. [hi < j; j < length xs'] ⇒ xs'''!j = xs''!j⟩
  using IH2' by (auto simp add: quicksort-post-def)
note IH2-perm = quicksort-post-set[OF IH2]

```

We still have a partition after the first call (same as in case 5)

```

have still-partition1: ⟨isPartition-map R h xs'' lo hi p⟩
proof (intro isPartition-wrtI)

```



**fix**  $i$  **assume**  $\langle lo \leq i \rangle \langle i < p \rangle$

Obtain the position  $posI$  where  $xs'' ! i$  was stored in  $xs'$ .

```

have  $\langle xs'' ! i \in \text{set } (\text{sublist } xs'' \text{ lo } (p - \text{Suc } 0)) \rangle$ 
by (metis (no-types, lifting) IH1(1) Suc-leI Suc-pred  $\langle i < p \rangle \langle lo \leq i \rangle$  le-less-trans less-imp-diff-less
mset-eq-length not-le not-less-zero part(1) part(5) pre(2) sublist-el')
then have  $\langle xs'' ! i \in \text{set } (\text{sublist } xs' \text{ lo } (p - \text{Suc } 0)) \rangle$ 
by (metis IH1-perm ifs(1) le-less-trans less-imp-diff-less mset-eq-length nat-le-linear part(1)
part(5) pre(2))
then have  $\langle \exists \text{ posI. } lo \leq posI \wedge posI \leq p - \text{Suc } 0 \wedge xs'' ! i = xs' ! posI \rangle$ 
proof — — sledgehammer
have  $p - \text{Suc } 0 < \text{length } xs$ 
by (meson diff-le-self le-less-trans part(5) pre(2))
then show ?thesis
by (metis (no-types)  $\langle xs'' ! i \in \text{set } (\text{sublist } xs' \text{ lo } (p - \text{Suc } 0)) \rangle$  ifs(1) mset-eq-length nat-le-linear
part(1) sublist-el')
qed
then obtain  $posI :: \text{nat}$  where  $PosI: \langle lo \leq posI \rangle \langle posI \leq p - \text{Suc } 0 \rangle \langle xs'' ! i = xs' ! posI \rangle$  by blast
then show  $\langle R \text{ (h } (xs'' ! i)) \text{ (h } (xs'' ! p)) \rangle$ 
by (metis (no-types, lifting) IH1(4)  $\langle i < p \rangle$  diff-Suc-less isPartition-wrt-def le-less-trans mset-eq-length
not-le not-less-eq part(1) part(3) part(5) pre(2) zero-less-Suc)
next
fix  $j$  assume  $\langle p < j \rangle \langle j \leq hi \rangle$ 
then show  $\langle R \text{ (h } (xs'' ! p)) \text{ (h } (xs'' ! j)) \rangle$ 

```

This holds because this part hasn't changed

```

by (smt IH1(4) add-diff-cancel-left' add-diff-inverse-nat diff-Suc-eq-diff-pred diff-le-self ifs(1)
isPartition-wrt-def le-less-Suc-eq less-le-trans mset-eq-length nat-less-le part(1) part(3) part(4) plus-1-eq-Suc
pre(2))
qed

```

We still have a partition after the second call (similar as in case 3)

```

have still-partition2:  $\langle \text{isPartition-map } R \text{ h } xs''' \text{ lo hi } p \rangle$ 
proof(intro isPartition-wrtI)
fix  $i$  assume  $\langle lo \leq i \rangle \langle i < p \rangle$ 
show  $\langle R \text{ (h } (xs''' ! i)) \text{ (h } (xs''' ! p)) \rangle$ 

```

This holds because this part hasn't changed

```

using IH2(3)  $\langle i < p \rangle \langle lo \leq i \rangle$  isPartition-wrt-def still-partition1 by fastforce
next
fix  $j$  assume  $\langle p < j \rangle \langle j \leq hi \rangle$ 

```

Obtain the position  $posJ$  where  $xs''' ! j$  was stored in  $xs'''$ .

```

have  $\langle xs''' ! j \in \text{set } (\text{sublist } xs''' \text{ (Suc } p) \text{ hi}) \rangle$ 
by (metis IH1(1) IH2(1) Suc-leI  $\langle j \leq hi \rangle \langle p < j \rangle$  ifs(2) nat-le-linear part(1) pre(2) size-mset
sublist-el')
then have  $\langle xs''' ! j \in \text{set } (\text{sublist } xs'' \text{ (Suc } p) \text{ hi}) \rangle$ 
by (metis IH1(1) IH2-perm ifs(2) mset-eq-length nat-le-linear part(1) pre(2))
then have  $\langle \exists \text{ posJ. } \text{Suc } p \leq posJ \wedge posJ \leq hi \wedge xs''' ! j = xs'' ! posJ \rangle$ 
by (metis IH1(1) ifs(2) mset-eq-length nat-le-linear part(1) pre(2) sublist-el')
then obtain  $posJ :: \text{nat}$  where  $PosJ: \langle \text{Suc } p \leq posJ \rangle \langle posJ \leq hi \rangle \langle xs''' ! j = xs'' ! posJ \rangle$  by blast

then show  $\langle R \text{ (h } (xs''' ! p)) \text{ (h } (xs''' ! j)) \rangle$ 
proof — — sledgehammer

```

```

have  $\forall n \text{ na } as \text{ p. } (p \text{ (as ! na::'a) (as ! posJ) } \vee \text{ posJ } \leq \text{ na}) \vee \neg \text{ isPartition-wrt } p \text{ as } n \text{ hi na}$ 
by (metis (no-types) PosJ(2) isPartition-wrt-def not-less)
then show ?thesis
by (metis IH2(3) PosJ(1) PosJ(3) lessI not-less-eq-eq still-partition1)
qed
qed

```

We have that the lower part is sorted after the first recursive call

```

note sorted-lower1 = IH1(2)

```

We show that it is still sorted after the second call.

```

have sorted-lower2:  $\langle \text{sorted-sublist-map } R \text{ h } xs''' \text{ lo } (p - \text{Suc } 0) \rangle$ 
proof –
show ?thesis
using sorted-lower1 apply (rule sorted-wrt-lower-sublist-still-sorted)
subgoal by (rule part)
subgoal
using IH1(1) mset-eq-length part(1) part(5) pre(2) by fastforce
subgoal
by (simp add: IH2(3))
subgoal
by (metis IH2(1) size-mset)
done
qed

```

The second IH gives us the the upper list is sorted after the second recursive call

```

note sorted-upper2 = IH2(2)

```

Finally, we have to show that the entire list is sorted after the second recursive call.

```

have sorted-middle:  $\langle \text{sorted-sublist-map } R \text{ h } xs''' \text{ lo hi} \rangle$ 
proof –
show ?thesis
apply (rule merge-sorted-map-partitions[where  $p=p$ ])
subgoal by (rule trans)
subgoal by (rule still-partition2)
subgoal by (rule sorted-lower2)
subgoal by (rule sorted-upper2)
subgoal using pre(1) by auto
subgoal by (simp add: part(4))
subgoal by (simp add: part(5))
subgoal by (metis IH1(1) IH2(1) part(1) pre(2) size-mset)
done
qed

```

```

show ?thesis
proof (intro quicksort-postI)
show  $\langle \text{mset } xs''' = \text{mset } xs \rangle$ 
by (simp add: IH1(1) IH2(1) part(1))
next
show  $\langle \text{sorted-sublist-map } R \text{ h } xs''' \text{ lo hi} \rangle$ 
by (rule sorted-middle)
next
show  $\langle \bigwedge i. i < \text{lo} \implies xs''' ! i = xs ! i \rangle$ 
using IH1(3) IH2(3) part(4) part(6) by auto
next

```

```

show  $\langle \bigwedge j. hi < j \implies j < \text{length } xs \implies xs''' ! j = xs ! j \rangle$ 
by (metis IH1(1) IH1(4) IH2(4) diff-le-self ifs(2) le-SucI less-le-trans nat-le-eq-or-lt not-less
part(1) part(7) size-mset)
qed

```

**qed**

We can now show the correctness of the abstract quicksort procedure, using the refinement framework and the above case lemmas.

**lemma** *quicksort-correct*:

```

assumes trans:  $\langle \bigwedge x y z. \llbracket R(h x)(h y); R(h y)(h z) \rrbracket \implies R(h x)(h z) \rangle$  and lin:  $\langle \bigwedge x y. x \neq y \implies R(h x)(h y) \vee R(h y)(h x) \rangle$ 
and Pre:  $\langle lo0 \leq hi0 \rangle \langle hi0 < \text{length } xs0 \rangle$ 
shows  $\langle \text{quicksort } R \ h \ (lo0, hi0, xs0) \leq \Downarrow Id \ (SPEC(\lambda xs. \text{quicksort-post } R \ h \ lo0 \ hi0 \ xs0 \ xs)) \rangle$ 
proof –
have wf:  $\langle wf \ (measure \ (\lambda(lo, hi, xs). Suc \ hi - lo)) \rangle$ 
by auto
define pre where  $\langle pre = (\lambda(lo, hi, xs). \text{quicksort-pre } R \ h \ xs0 \ lo \ hi \ xs) \rangle$ 
define post where  $\langle post = (\lambda(lo, hi, xs). \text{quicksort-post } R \ h \ lo \ hi \ xs) \rangle$ 
have pre:  $\langle pre \ (lo0, hi0, xs0) \rangle$ 
unfolding quicksort-pre-def pre-def by (simp add: Pre)

```

We first generalize the goal a over all states.

```

have  $\langle WB\text{-Sort.} \text{quicksort } R \ h \ (lo0, hi0, xs0) \leq \Downarrow Id \ (SPEC \ (post \ (lo0, hi0, xs0))) \rangle$ 
unfolding quicksort-def prod.case
apply (rule RECT-rule)
apply (refine-mono)
apply (rule wf)
apply (rule pre)
subgoal premises IH for f x
apply (refine-vcg ASSERT-leI)
unfolding pre-def post-def

```

```

subgoal — First premise (assertion) for partition
using IH(2) by (simp add: quicksort-pre-def pre-def)
subgoal — Second premise (assertion) for partition
using IH(2) by (simp add: quicksort-pre-def pre-def)
subgoal
using IH(2) by (auto simp add: quicksort-pre-def pre-def dest: mset-eq-setD)

```

Termination case:  $p - (1::'c) \leq lo'$  and  $hi' \leq p + (1::'c)$ ; directly show postcondition

```

subgoal unfolding partition-spec-def by (auto dest: mset-eq-setD)
subgoal — Postcondition (after partition)
apply –
using IH(2) unfolding pre-def apply (simp, elim conjE, split prod.splits)
using trans lin apply (rule quicksort-correct-case1) by auto

```

Case  $p - (1::'c) \leq lo'$  and  $hi' < p + (1::'c)$  (Only second recursive call)

```

subgoal
apply (rule IH(1)[THEN order-trans])

```

Show that the invariant holds for the second recursive call

```

subgoal
using IH(2) unfolding pre-def apply (simp, elim conjE, split prod.splits)

```

**apply** (*rule quicksort-correct-case2*) **by** *auto*

Wellfoundedness (easy)

**subgoal by** (*auto simp add: quicksort-pre-def partition-spec-def*)

Show that the postcondition holds

**subgoal**  
**apply** (*simp add: Misc.subset-Collect-conv post-def, intro allI impI, elim conjE*)  
**using** *trans lin* **apply** (*rule quicksort-correct-case3*)  
**using** *IH(2)* **unfolding** *pre-def* **by** *auto*  
**done**

Case: At least the first recursive call

**subgoal**  
**apply** (*rule IH(1)[THEN order-trans]*)

Show that the precondition holds for the first recursive call

**subgoal**  
**using** *IH(2)* **unfolding** *pre-def post-def* **apply** (*simp, elim conjE, split prod.splits*) **apply** *auto*  
**apply** (*rule quicksort-correct-case4*) **by** *auto*

Wellfoundedness for first recursive call (easy)

**subgoal by** (*auto simp add: quicksort-pre-def partition-spec-def*)

Simplify some refinement suff...

**apply** (*simp add: Misc.subset-Collect-conv ASSERT-leI, intro allI impI conjI, elim conjE*)  
**apply** (*rule ASSERT-leI*)  
**apply** (*simp-all add: Misc.subset-Collect-conv ASSERT-leI*)  
**subgoal unfolding** *quicksort-post-def pre-def post-def* **by** (*auto dest: mset-eq-setD*)

Only the first recursive call: show postcondition

**subgoal**  
**using** *trans lin* **apply** (*rule quicksort-correct-case5*)  
**using** *IH(2)* **unfolding** *pre-def post-def* **by** *auto*  
  
**apply** (*rule ASSERT-leI*)  
**subgoal unfolding** *quicksort-post-def pre-def post-def* **by** (*auto dest: mset-eq-setD*)

Both recursive calls.

**subgoal**  
**apply** (*rule IH(1)[THEN order-trans]*)

Show precondition for second recursive call (after the first call)

**subgoal**  
**unfolding** *pre-def post-def*  
**apply** *auto*  
**apply** (*rule quicksort-correct-case6*)  
**using** *IH(2)* **unfolding** *pre-def post-def* **by** *auto*

Wellfoundedness for second recursive call (easy)

**subgoal by** (*auto simp add: quicksort-pre-def partition-spec-def*)

Show that the postcondition holds (after both recursive calls)

```

    subgoal
      apply (simp add: Misc.subset-Collect-conv, intro allI impI, elim conjE)
      using trans lin apply (rule quicksort-correct-case7)
      using IH(2) unfolding pre-def post-def by auto
    done
  done
done
done

```

Finally, apply the generalized lemma to show the thesis.

```

then show ?thesis unfolding post-def by auto
qed

```

**definition** *partition-main-inv* ::  $\langle ('b \Rightarrow 'b \Rightarrow \text{bool}) \Rightarrow ('a \Rightarrow 'b) \Rightarrow \text{nat} \Rightarrow \text{nat} \Rightarrow 'a \text{ list} \Rightarrow (\text{nat} \times \text{nat} \times 'a \text{ list}) \Rightarrow \text{bool} \rangle$  **where**

```

  partition-main-inv R h lo hi xs0 p  $\equiv$ 
    case p of (i,j,xs)  $\Rightarrow$ 
      j < length xs  $\wedge$  j  $\leq$  hi  $\wedge$  i < length xs  $\wedge$  lo  $\leq$  i  $\wedge$  i  $\leq$  j  $\wedge$  mset xs = mset xs0  $\wedge$ 
      ( $\forall k. k \geq lo \wedge k < i \longrightarrow R (h (xs!k)) (h (xs!hi))$ )  $\wedge$  — All elements from lo to i - (1::'c) are smaller
      than the pivot
      ( $\forall k. k \geq i \wedge k < j \longrightarrow R (h (xs!hi)) (h (xs!k))$ )  $\wedge$  — All elements from i to j - (1::'c) are greater
      than the pivot
      ( $\forall k. k < lo \longrightarrow xs!k = xs0!k$ )  $\wedge$  — Everything below lo is unchanged
      ( $\forall k. k \geq j \wedge k < \text{length } xs \longrightarrow xs!k = xs0!k$ ) — All elements from j are unchanged (including
      everything above hi)
  }

```

The main part of the partition function. The pivot is assumed to be the last element. This is exactly the "Lomuto partition scheme" partition function from Wikipedia.

**definition** *partition-main* ::  $\langle ('b \Rightarrow 'b \Rightarrow \text{bool}) \Rightarrow ('a \Rightarrow 'b) \Rightarrow \text{nat} \Rightarrow \text{nat} \Rightarrow 'a \text{ list} \Rightarrow ('a \text{ list} \times \text{nat}) \text{ nres} \rangle$  **where**

```

  partition-main R h lo hi xs0 = do {
    ASSERT(hi < length xs0);
    pivot  $\leftarrow$  RETURN (h (xs0 ! hi));
    (i,j,xs)  $\leftarrow$  WHILE_T partition-main-inv R h lo hi xs0 — We loop from j = lo to j = hi - (1::'c).
    ( $\lambda(i,j,xs). j < hi$ )
    ( $\lambda(i,j,xs). \text{do } \{$ 
      ASSERT(i < length xs  $\wedge$  j < length xs);
      if R (h (xs!j)) pivot
      then RETURN (i+1, j+1, swap xs i j)
      else RETURN (i, j+1, xs)
    })
    (lo, lo, xs0); — i and j are both initialized to lo
    ASSERT(i < length xs  $\wedge$  j = hi  $\wedge$  lo  $\leq$  i  $\wedge$  hi < length xs  $\wedge$  mset xs = mset xs0);
    RETURN (swap xs i hi, i)
  }

```

**lemma** *partition-main-correct*:

**assumes** *bounds*:  $\langle hi < \text{length } xs \rangle \langle lo \leq hi \rangle$  **and**  
*trans*:  $\langle \bigwedge x y z. \llbracket R(h x) (h y); R(h y) (h z) \rrbracket \implies R(h x) (h z) \rangle$  **and** *lin*:  $\langle \bigwedge x y. R(h x) (h y) \vee R(h y) (h x) \rangle$   
**shows**  $\langle \text{partition-main } R \ h \ lo \ hi \ xs \leq \text{SPEC}(\lambda(xs', p). \text{mset } xs = \text{mset } xs' \wedge$   
 $lo \leq p \wedge p \leq hi \wedge \text{isPartition-map } R \ h \ xs' \ lo \ hi \ p \wedge (\forall i. i < lo \longrightarrow xs'!i = xs!i) \wedge (\forall i. hi < i \wedge i < \text{length } xs' \longrightarrow xs'!i = xs!i) \rangle$   
**proof** –  
**have** *K*:  $\langle b \leq hi - \text{Suc } n \implies n > 0 \implies \text{Suc } n \leq hi \implies \text{Suc } b \leq hi - n \rangle$  **for**  $b \ hi \ n$   
**by** *auto*  
**have** *L*:  $\langle \sim R(h x) (h y) \implies R(h y) (h x) \rangle$  **for**  $x \ y$  — Corollary of linearity  
**using** *assms* **by** *blast*  
**have** *M*:  $\langle a < \text{Suc } b \equiv a = b \vee a < b \rangle$  **for**  $a \ b$   
**by** *linarith*  
**have** *N*:  $\langle (a::\text{nat}) \leq b \equiv a = b \vee a < b \rangle$  **for**  $a \ b$   
**by** *arith*  
  
**show** *?thesis*  
**unfolding** *partition-main-def choose-pivot-def*  
**apply** (*refine-vcg WHILEIT-rule*[**where**  $R = \langle \text{measure}(\lambda(i,j,xs). \text{hi} - j) \rangle$ ])  
**subgoal using** *assms* **by** *blast* — We feed our assumption to the assertion  
**subgoal by** *auto* — WF  
**subgoal** — Invariant holds before the first iteration  
**unfolding** *partition-main-inv-def*  
**using** *assms* **apply** *simp* **by** *linarith*  
**subgoal unfolding** *partition-main-inv-def* **by** *simp*  
**subgoal unfolding** *partition-main-inv-def* **by** *simp*  
**subgoal**  
**unfolding** *partition-main-inv-def*  
**apply** (*auto dest: mset-eq-length*)  
**done**  
**subgoal unfolding** *partition-main-inv-def* **by** (*auto dest: mset-eq-length*)  
**subgoal**  
**unfolding** *partition-main-inv-def* **apply** (*auto dest: mset-eq-length*)  
**by** (*metis L M mset-eq-length nat-le-eq-or-lt*)  
  
**subgoal unfolding** *partition-main-inv-def* **by** *simp* — assertions, etc  
**subgoal unfolding** *partition-main-inv-def* **by** *simp*  
**subgoal unfolding** *partition-main-inv-def* **by** (*auto dest: mset-eq-length*)  
**subgoal unfolding** *partition-main-inv-def* **by** *simp*  
**subgoal unfolding** *partition-main-inv-def* **by** (*auto dest: mset-eq-length*)  
**subgoal unfolding** *partition-main-inv-def* **by** (*auto dest: mset-eq-length*)  
**subgoal unfolding** *partition-main-inv-def* **by** (*auto dest: mset-eq-length*)  
**subgoal unfolding** *partition-main-inv-def* **by** *simp*  
**subgoal unfolding** *partition-main-inv-def* **by** *simp*  
  
**subgoal** — After the last iteration, we have a partitioning! :-)  
**unfolding** *partition-main-inv-def* **by** (*auto simp add: isPartition-wrt-def*)  
**subgoal** — And the lower out-of-bounds parts of the list haven't been changed  
**unfolding** *partition-main-inv-def* **by** *auto*  
**subgoal** — And the upper out-of-bounds parts of the list haven't been changed  
**unfolding** *partition-main-inv-def* **by** *auto*  
**done**  
**qed**

**definition** *partition-between* ::  $\langle 'b \Rightarrow 'b \Rightarrow \text{bool} \rangle \Rightarrow \langle 'a \Rightarrow 'b \rangle \Rightarrow \text{nat} \Rightarrow \text{nat} \Rightarrow 'a \text{ list} \Rightarrow ('a \text{ list} \times \text{nat})$   
*nres* **where**  
 $\langle \text{partition-between } R \ h \ lo \ hi \ xs0 = \text{do} \{$   
 $\quad \text{ASSERT}(hi < \text{length } xs0 \wedge lo \leq hi);$   
 $\quad k \leftarrow \text{choose-pivot } R \ h \ xs0 \ lo \ hi; \text{ — choice of pivot}$   
 $\quad \text{ASSERT}(k < \text{length } xs0);$   
 $\quad xs \leftarrow \text{RETURN } (\text{swap } xs0 \ k \ hi); \text{ — move the pivot to the last position, before we start the actual}$   
 $\text{loop}$   
 $\quad \text{ASSERT}(\text{length } xs = \text{length } xs0);$   
 $\quad \text{partition-main } R \ h \ lo \ hi \ xs$   
 $\quad \}$

**lemma** *partition-between-correct*:

**assumes**  $\langle hi < \text{length } xs \rangle$  **and**  $\langle lo \leq hi \rangle$  **and**  
 $\langle \bigwedge x \ y \ z. \llbracket R \ (h \ x) \ (h \ y); R \ (h \ y) \ (h \ z) \rrbracket \implies R \ (h \ x) \ (h \ z) \rangle$  **and**  $\langle \bigwedge x \ y. R \ (h \ x) \ (h \ y) \vee R \ (h \ y) \ (h \ x) \rangle$   
**shows**  $\langle \text{partition-between } R \ h \ lo \ hi \ xs \leq \text{SPEC}(\text{uncurry } (\text{partition-spec } R \ h \ xs \ lo \ hi)) \rangle$

**proof** —

**have**  $K: \langle b \leq hi - \text{Suc } n \implies n > 0 \implies \text{Suc } n \leq hi \implies \text{Suc } b \leq hi - n \rangle$  **for**  $b \ hi \ n$   
**by** *auto*  
**show** *?thesis*  
**unfolding** *partition-between-def choose-pivot-def*  
**apply** (*refine-vcg partition-main-correct*)  
**using** *assms* **apply** (*auto dest: mset-eq-length simp add: partition-spec-def*)  
**by** (*metis dual-order.strict-trans2 less-imp-not-eq2 mset-eq-length swap-nth*)

**qed**

We use the median of the first, the middle, and the last element.

**definition** *choose-pivot3* **where**

$\langle \text{choose-pivot3 } R \ h \ xs \ lo \ (hi::\text{nat}) = \text{do} \{$   
 $\quad \text{ASSERT}(lo < \text{length } xs);$   
 $\quad \text{ASSERT}(hi < \text{length } xs);$   
 $\quad \text{let } k' = (hi - lo) \ \text{div } 2;$   
 $\quad \text{let } k = lo + k';$   
 $\quad \text{ASSERT}(k < \text{length } xs);$   
 $\quad \text{let } start = h \ (xs \ ! \ lo);$   
 $\quad \text{let } mid = h \ (xs \ ! \ k);$   
 $\quad \text{let } end = h \ (xs \ ! \ hi);$   
 $\quad \text{if } (R \ start \ mid \wedge R \ mid \ end) \vee (R \ end \ mid \wedge R \ mid \ start) \text{ then RETURN } k$   
 $\quad \text{else if } (R \ start \ end \wedge R \ end \ mid) \vee (R \ mid \ end \wedge R \ end \ start) \text{ then RETURN } hi$   
 $\quad \text{else RETURN } lo$   
 $\quad \}$

— We only have to show that this procedure yields a valid index between *lo* and *hi*.

**lemma** *choose-pivot3-choose-pivot*:

**assumes**  $\langle lo < \text{length } xs \rangle$   $\langle hi < \text{length } xs \rangle$   $\langle hi \geq lo \rangle$   
**shows**  $\langle \text{choose-pivot3 } R \ h \ xs \ lo \ hi \leq \Downarrow Id \ (\text{choose-pivot } R \ h \ xs \ lo \ hi) \rangle$   
**unfolding** *choose-pivot3-def choose-pivot-def*  
**using** *assms* **by** (*auto intro!: ASSERT-leI simp: Let-def*)

The refined partion function: We use the above pivot function and fold instead of non-deterministic iteration.

**definition** *partition-between-ref*

::  $\langle 'b \Rightarrow 'b \Rightarrow \text{bool} \rangle \Rightarrow \langle 'a \Rightarrow 'b \rangle \Rightarrow \text{nat} \Rightarrow \text{nat} \Rightarrow 'a \text{ list} \Rightarrow ('a \text{ list} \times \text{nat})$  *nres*

**where**

```

  ⟨partition-between-ref R h lo hi xs0 = do {
    ASSERT(hi < length xs0 ∧ hi < length xs0 ∧ lo ≤ hi);
    k ← choose-pivot3 R h xs0 lo hi; — choice of pivot
    ASSERT(k < length xs0);
    xs ← RETURN (swap xs0 k hi); — move the pivot to the last position, before we start the actual
loop
    ASSERT(length xs = length xs0);
    partition-main R h lo hi xs
  }⟩

```

**lemma** *partition-main-ref'*:

```

  ⟨partition-main R h lo hi xs
    ≤ ↓ ((λ a b c d. Id) a b c d) (partition-main R h lo hi xs)⟩
by auto

```

**lemma** *Down-id-eq*:

```

  ⟨↓ Id x = x⟩
by auto

```

**lemma** *partition-between-ref-partition-between*:

```

  ⟨partition-between-ref R h lo hi xs ≤ (partition-between R h lo hi xs)⟩
proof —
  have swap: ⟨(swap xs k hi, swap xs ka hi) ∈ Id⟩ if ⟨k = ka⟩
    for k ka
    using that by auto
  have [refine0]: ⟨(h (xsa ! hi), h (xsaa ! hi)) ∈ Id⟩
    if ⟨(xsa, xsaa) ∈ Id⟩
    for xsa xsaa
    using that by auto

```

**show** *?thesis*

```

  apply (subst (2) Down-id-eq[symmetric])
  unfolding partition-between-ref-def
    partition-between-def
    OP-def
  apply (refine-vcg choose-pivot3-choose-pivot swap partition-main-correct)
  subgoal by auto
  subgoal by auto
  subgoal by auto
  subgoal by auto
  subgoal by auto
  subgoal by auto
  subgoal by auto
  subgoal by auto
  subgoal by auto
  by (auto intro: Refine-Basic.Id-refine dest: mset-eq-length)
qed

```

Technical lemma for *sepref*

**lemma** *partition-between-ref-partition-between'*:

```

  ⟨(uncurry2 (partition-between-ref R h), uncurry2 (partition-between R h)) ∈
    (nat-rel ×r nat-rel) ×r ⟨Id⟩list-rel →f ⟨⟨Id⟩list-rel ×r nat-rel⟩nres-rel⟩

```



by (intro frefI nres-relI)  
 (auto intro: partition-between-ref-partition-between)

Example instantiation for pivot

**definition** choose-pivot3-impl **where**  
 ⟨choose-pivot3-impl = choose-pivot3 (≤) id⟩

**lemma** partition-between-ref-correct:

**assumes** trans: ⟨ $\bigwedge x y z. \llbracket R(h x) (h y); R(h y) (h z) \rrbracket \implies R(h x) (h z)$ ⟩ **and** lin: ⟨ $\bigwedge x y. R(h x) (h y) \vee R(h y) (h x)$ ⟩

**and** bounds: ⟨ $hi < \text{length } xs \wedge lo \leq hi$ ⟩

**shows** ⟨partition-between-ref R h lo hi xs ≤ SPEC (uncurry (partition-spec R h xs lo hi))⟩

**proof** –

**show** ?thesis

**apply** (rule partition-between-ref-partition-between[THEN order-trans])

**using** bounds **apply** (rule partition-between-correct[where h=h])

**subgoal** by (rule trans)

**subgoal** by (rule lin)

**done**

**qed**

Refined quicksort algorithm: We use the refined partition function.

**definition** quicksort-ref ::  $\langle \cdot \Rightarrow \cdot \Rightarrow \text{nat} \times \text{nat} \times 'a \text{ list} \Rightarrow 'a \text{ list nres} \rangle$  **where**

⟨quicksort-ref R h = (λ(lo,hi,xs0).

do {

RECT (λf (lo,hi,xs). do {

ASSERT(lo ≤ hi ∧ hi < length xs0 ∧ mset xs = mset xs0);

(xs, p) ← partition-between-ref R h lo hi xs; — This is the refined partition function. Note that we need the premises (trans,lin,bounds) here.

ASSERT(mset xs = mset xs0 ∧ p ≥ lo ∧ p < length xs0);

xs ← (if p-1 ≤ lo then RETURN xs else f (lo, p-1, xs));

ASSERT(mset xs = mset xs0);

if hi ≤ p+1 then RETURN xs else f (p+1, hi, xs)

}) (lo,hi,xs0)

})⟩

**lemma** fref-to-Down-curry2:

⟨(uncurry2 f, uncurry2 g) ∈ [P]<sub>f</sub> A → ⟨B⟩nres-rel ⟹

( $\bigwedge x x' y y' z z'. P((x', y'), z') \implies ((x, y), z), ((x', y'), z') \in A \implies$

$f x y z \leq \Downarrow B (g x' y' z')$ ⟩

**unfolding** fref-def uncurry-def nres-rel-def

**by** auto

**lemma** fref-to-Down-curry:

⟨(f, g) ∈ [P]<sub>f</sub> A → ⟨B⟩nres-rel ⟹

( $\bigwedge x x'. P x' \implies (x, x') \in A \implies$

$f x \leq \Downarrow B (g x')$ ⟩

**unfolding** fref-def uncurry-def nres-rel-def

**by** auto

```

lemma quicksort-ref-quicksort:
  assumes bounds:  $\langle hi < \text{length } xs \rangle \langle lo \leq hi \rangle$  and
    trans:  $\langle \bigwedge x y z. \llbracket R(h x) (h y); R(h y) (h z) \rrbracket \implies R(h x) (h z) \rangle$  and lin:  $\langle \bigwedge x y. R(h x) (h y) \vee R(h y) (h x) \rangle$ 
  shows  $\langle \text{quicksort-ref } R \ h \ x0 \leq \Downarrow Id (\text{quicksort } R \ h \ x0) \rangle$ 
proof -
  have wf:  $\langle wf (\text{measure } (\lambda(lo, hi, xs). \text{Suc } hi - lo)) \rangle$ 
    by auto
  have pre:  $\langle x0 = x0' \implies (x0, x0') \in Id \times_r Id \times_r \langle Id \rangle \text{list-rel} \rangle$  for  $x0 \ x0' :: \langle nat \times nat \times 'b \text{ list} \rangle$ 
    by auto
  have [refine0]:  $\langle (x1e = x1d) \implies (x1e, x1d) \in Id \rangle$  for  $x1e \ x1d :: \langle 'b \text{ list} \rangle$ 
    by auto

  show ?thesis
  unfolding quicksort-def quicksort-ref-def
  apply (refine-vcg pre partition-between-ref-partition-between'[THEN fref-to-Down-curry2])

```

First assertion (premise for partition)

```

subgoal
by auto

```

First assertion (premise for partition)

```

subgoal
by auto
subgoal
by (auto dest: mset-eq-length)
subgoal
by (auto dest: mset-eq-length mset-eq-setD)

```

Correctness of the concrete partition function

```

subgoal
apply (simp, rule partition-between-ref-correct)
subgoal by (rule trans)
subgoal by (rule lin)
subgoal by auto — first premise
subgoal by auto — second premise
done
subgoal
by (auto dest: mset-eq-length mset-eq-setD)
subgoal by (auto simp: partition-spec-def isPartition-wrt-def)
subgoal by (auto simp: partition-spec-def isPartition-wrt-def dest: mset-eq-length)
subgoal
by (auto dest: mset-eq-length mset-eq-setD)
subgoal
by (auto dest: mset-eq-length mset-eq-setD)
subgoal
by (auto dest: mset-eq-length mset-eq-setD)
subgoal
by (auto dest: mset-eq-length mset-eq-setD)

```

```

by simp+
qed

```

— Sort the entire list

**definition** full-quicksort **where**

$\langle \text{full-quicksort } R \ h \ xs \equiv \text{if } xs = [] \text{ then RETURN } xs \text{ else quicksort } R \ h \ (0, \text{length } xs - 1, xs) \rangle$

**definition** *full-quicksort-ref* **where**

$\langle \text{full-quicksort-ref } R \ h \ xs \equiv$   
 $\text{if List.null } xs \text{ then RETURN } xs$   
 $\text{else quicksort-ref } R \ h \ (0, \text{length } xs - 1, xs) \rangle$

**definition** *full-quicksort-impl* ::  $\langle \text{nat list} \Rightarrow \text{nat list nres} \rangle$  **where**

$\langle \text{full-quicksort-impl } xs = \text{full-quicksort-ref } (\leq) \ id \ xs \rangle$

**lemma** *full-quicksort-ref-full-quicksort*:

**assumes** *trans*:  $\langle \bigwedge x \ y \ z. \llbracket R \ (h \ x) \ (h \ y); R \ (h \ y) \ (h \ z) \rrbracket \Longrightarrow R \ (h \ x) \ (h \ z) \rangle$  **and** *lin*:  $\langle \bigwedge x \ y. R \ (h \ x) \ (h \ y) \vee R \ (h \ y) \ (h \ x) \rangle$

**shows**  $\langle (\text{full-quicksort-ref } R \ h, \text{full-quicksort } R \ h) \in$   
 $\langle Id \rangle \text{list-rel} \rightarrow_f \langle \langle Id \rangle \text{list-rel} \rangle \text{nres-rel} \rangle$

**proof** –

**show** *?thesis*

**unfolding** *full-quicksort-ref-def full-quicksort-def*

**apply** (*intro frefI nres-relI*)

**apply** (*auto intro! quicksort-ref-quicksort[unfolded Down-id-eq] simp: List.null-def*)

**subgoal by** (*rule trans*)

**subgoal using** *lin* **by** *blast*

**done**

**qed**

**lemma** *sublist-entire*:

$\langle \text{sublist } xs \ 0 \ (\text{length } xs - 1) = xs \rangle$

**by** (*simp add: sublist-def*)

**lemma** *sorted-sublist-wrt-entire*:

**assumes**  $\langle \text{sorted-sublist-wrt } R \ xs \ 0 \ (\text{length } xs - 1) \rangle$

**shows**  $\langle \text{sorted-wrt } R \ xs \rangle$

**proof** –

**have**  $\langle \text{sorted-wrt } R \ (\text{sublist } xs \ 0 \ (\text{length } xs - 1)) \rangle$

**using** *assms* **by** (*simp add: sorted-sublist-wrt-def*)

**then show** *?thesis*

**by** (*metis sublist-entire*)

**qed**

**lemma** *sorted-sublist-map-entire*:

**assumes**  $\langle \text{sorted-sublist-map } R \ h \ xs \ 0 \ (\text{length } xs - 1) \rangle$

**shows**  $\langle \text{sorted-wrt } (\lambda x \ y. R \ (h \ x) \ (h \ y)) \ xs \rangle$

**proof** –

**show** *?thesis*

**using** *assms* **by** (*rule sorted-sublist-wrt-entire*)

**qed**

Final correctness lemma

**theorem** *full-quicksort-correct-sorted*:

**assumes**

*trans*:  $\langle \bigwedge x \ y \ z. \llbracket R \ (h \ x) \ (h \ y); R \ (h \ y) \ (h \ z) \rrbracket \Longrightarrow R \ (h \ x) \ (h \ z) \rangle$  **and** *lin*:  $\langle \bigwedge x \ y. x \neq y \Longrightarrow R \ (h \ x) \ (h \ y) \vee R \ (h \ y) \ (h \ x) \rangle$

**shows**  $\langle \text{full-quicksort } R \ h \ xs \leq \Downarrow Id \ (\text{SPEC}(\lambda xs'. \text{mset } xs' = \text{mset } xs \wedge \text{sorted-wrt } (\lambda x \ y. R \ (h \ x) \ (h \ y)) \ xs') \rangle$

```

y)) xs'))
proof -
  show ?thesis
    unfolding full-quicksort-def
    apply (refine-vcg)
    subgoal by simp — case xs=[]
    subgoal by simp — case xs=[]

    apply (rule quicksort-correct[THEN order-trans])
    subgoal by (rule trans)
    subgoal by (rule lin)
    subgoal by linarith
    subgoal by simp

    apply (simp add: Misc.subset-Collect-conv, intro allI impI conjI)
    subgoal
      by (auto simp add: quicksort-post-def)
    subgoal
      apply (rule sorted-sublist-map-entire)
      by (auto simp add: quicksort-post-def dest: mset-eq-length)
    done
qed

lemma full-quicksort-correct:
  assumes
    trans:  $\langle \bigwedge x y z. \llbracket R(h x) (h y); R(h y) (h z) \rrbracket \implies R(h x) (h z) \rangle$  and
    lin:  $\langle \bigwedge x y. R(h x) (h y) \vee R(h y) (h x) \rangle$ 
  shows  $\langle \text{full-quicksort } R \ h \ xs \leq \Downarrow Id \ (SPEC(\lambda xs'. \text{mset } xs' = \text{mset } xs)) \rangle$ 
  by (rule order-trans[OF full-quicksort-correct-sorted])
  (use assms in auto)

end
theory More-Loops
imports
  Refine-Monadic.Refine-While
  Refine-Monadic.Refine-Foreach
  HOL-Library.Rewrite
begin

```

## 1.4 More Theorem about Loops

Most theorem below have a counterpart in the Refinement Framework that is weaker (by missing assertions for example that are critical for code generation).

**lemma** *Down-id-eq*:

```

 $\langle \Downarrow Id \ x = x \rangle$ 
by auto

```

**lemma** *while-upt-while-direct1*:

```

 $b \geq a \implies$ 
do {
   $(-, \sigma) \leftarrow WHILE_T \ (FOREACH-cond \ c) \ (\lambda x. \text{do } \{ ASSERT \ (FOREACH-cond \ c \ x); FOREACH-body \ f \ x \})$ 
   $([a..<b], \sigma);$ 
  RETURN  $\sigma$ 
}  $\leq$  do {

```

```

   $(-, \sigma) \leftarrow \text{WHILE}_T (\lambda(i, x). i < b \wedge c \ x) (\lambda(i, x). \text{do } \{\text{ASSERT } (i < b); \sigma' \leftarrow f \ i \ x; \text{RETURN } (i+1, \sigma')\}) (a, \sigma);$ 
  RETURN  $\sigma$ 
}
apply (rewrite at  $\langle - \leq \sqsupset \rangle$  Down-id-eq[symmetric])
apply (refine-vcg WHILET-refine[where  $R = \langle \{((l, x'), (i::\text{nat}, x::'a)). x = x' \wedge i \leq b \wedge i \geq a \wedge l = \text{drop } (i-a) [a..<b]\} \rangle$ ])
subgoal by auto
subgoal by (auto simp: FOREACH-cond-def)
subgoal by (auto simp: FOREACH-body-def intro!: bind-refine[OF Id-refine])
subgoal by auto
done

```

**lemma** while-upt-while-direct2:

```

   $b \geq a \implies$ 
  do {
     $(-, \sigma) \leftarrow \text{WHILE}_T (\text{FOREACH-cond } c) (\lambda x. \text{do } \{\text{ASSERT } (\text{FOREACH-cond } c \ x); \text{FOREACH-body } f \ x\})$ 
     $([a..<b], \sigma);$ 
    RETURN  $\sigma$ 
  }  $\geq$  do {
     $(-, \sigma) \leftarrow \text{WHILE}_T (\lambda(i, x). i < b \wedge c \ x) (\lambda(i, x). \text{do } \{\text{ASSERT } (i < b); \sigma' \leftarrow f \ i \ x; \text{RETURN } (i+1, \sigma')\}) (a, \sigma);$ 
    RETURN  $\sigma$ 
  }
apply (rewrite at  $\langle - \leq \sqsupset \rangle$  Down-id-eq[symmetric])
apply (refine-vcg WHILET-refine[where  $R = \langle \{((i::\text{nat}, x::'a), (l, x')). x = x' \wedge i \leq b \wedge i \geq a \wedge l = \text{drop } (i-a) [a..<b]\} \rangle$ ])
subgoal by auto
subgoal by (auto simp: FOREACH-cond-def)
subgoal by (auto simp: FOREACH-body-def intro!: bind-refine[OF Id-refine])
subgoal by (auto simp: FOREACH-body-def intro!: bind-refine[OF Id-refine])
subgoal by auto
done

```

**lemma** while-upt-while-direct:

```

   $b \geq a \implies$ 
  do {
     $(-, \sigma) \leftarrow \text{WHILE}_T (\text{FOREACH-cond } c) (\lambda x. \text{do } \{\text{ASSERT } (\text{FOREACH-cond } c \ x); \text{FOREACH-body } f \ x\})$ 
     $([a..<b], \sigma);$ 
    RETURN  $\sigma$ 
  } = do {
     $(-, \sigma) \leftarrow \text{WHILE}_T (\lambda(i, x). i < b \wedge c \ x) (\lambda(i, x). \text{do } \{\text{ASSERT } (i < b); \sigma' \leftarrow f \ i \ x; \text{RETURN } (i+1, \sigma')\}) (a, \sigma);$ 
    RETURN  $\sigma$ 
  }
using while-upt-while-direct1[of a b] while-upt-while-direct2[of a b]
unfolding order-class.eq-iff by fast

```

**lemma** while-nfoldli:

```

  do {
     $(-, \sigma) \leftarrow \text{WHILE}_T (\text{FOREACH-cond } c) (\lambda x. \text{do } \{\text{ASSERT } (\text{FOREACH-cond } c \ x); \text{FOREACH-body } f \ x\}) (l, \sigma);$ 
    RETURN  $\sigma$ 
  }

```

```

} ≤ nfoldli l c f σ
apply (induct l arbitrary: σ)
apply (subst WHILET-unfold)
apply (simp add: FOREACH-cond-def)

apply (subst WHILET-unfold)
apply (auto
  simp: FOREACH-cond-def FOREACH-body-def
  intro: bind-mono Refine-Basic.bind-mono(1))
done
lemma nfoldli-while: nfoldli l c f σ
  ≤
  (WHILETI
    (FOREACH-cond c) (λx. do {ASSERT (FOREACH-cond c x); FOREACH-body f x})) (l, σ)
  >>=
  (λ(-, σ). RETURN σ))
proof (induct l arbitrary: σ)
  case Nil thus ?case by (subst WHILEIT-unfold) (auto simp: FOREACH-cond-def)
next
  case (Cons x ls)
  show ?case
  proof (cases c σ)
    case False thus ?thesis
      apply (subst WHILEIT-unfold)
      unfolding FOREACH-cond-def
      by simp
  next
    case [simp]: True
    from Cons show ?thesis
      apply (subst WHILEIT-unfold)
      unfolding FOREACH-cond-def FOREACH-body-def
      apply clarsimp
      apply (rule Refine-Basic.bind-mono)
      apply simp-all
      done
  qed
qed

lemma while-eq-nfoldli: do {
  (-,σ) ← WHILET (FOREACH-cond c) (λx. do {ASSERT (FOREACH-cond c x); FOREACH-body
f x}) (l,σ);
  RETURN σ
} = nfoldli l c f σ
apply (rule antisym)
apply (rule while-nfoldli)
apply (rule order-trans[OF nfoldli-while[where I=λ-. True]])
apply (simp add: WHILET-def)
done

end
theory PAC-More-Poly
imports HOL-Library.Poly-Mapping HOL-Algebra.Polynomials Polynomials.MPoly-Type-Class
HOL-Algebra.Module
HOL-Library.Countable-Set
begin

```

## 2 Libraries

### 2.1 More Polynoms

Here are more theorems on polynoms. Most of these facts are extremely trivial and should probably be generalised and moved to the Isabelle distribution.

**lemma** *Const<sub>0</sub>-add*:

$\langle \text{Const}_0 (a + b) = \text{Const}_0 a + \text{Const}_0 b \rangle$

**by** *transfer*

(*simp add: Const<sub>0</sub>-def single-add*)

**lemma** *Const-mult*:

$\langle \text{Const} (a * b) = \text{Const} a * \text{Const} b \rangle$

**by** *transfer*

(*simp add: Const<sub>0</sub>-def times-monomial-monomial*)

**lemma** *Const<sub>0</sub>-mult*:

$\langle \text{Const}_0 (a * b) = \text{Const}_0 a * \text{Const}_0 b \rangle$

**by** *transfer*

(*simp add: Const<sub>0</sub>-def times-monomial-monomial*)

**lemma** *Const0[simp]*:

$\langle \text{Const } 0 = 0 \rangle$

**by** *transfer (simp add: Const<sub>0</sub>-def)*

**lemma** (*in -*) *Const-uminus[simp]*:

$\langle \text{Const} (-n) = - \text{Const } n \rangle$

**by** *transfer*

(*auto simp: Const<sub>0</sub>-def monomial-uminus*)

**lemma** [*simp*]:  $\langle \text{Const}_0 0 = 0 \rangle$

$\langle \text{MPoly } 0 = 0 \rangle$

**supply** [*show-sorts*]

**by** (*auto simp: Const<sub>0</sub>-def zero-mpoly-def*)

**lemma** *Const-add*:

$\langle \text{Const} (a + b) = \text{Const} a + \text{Const} b \rangle$

**by** *transfer*

(*simp add: Const<sub>0</sub>-def single-add*)

**instance** *mpoly* :: (*comm-semiring-1*) *comm-semiring-1*

**by** *standard*

**lemma** *degree-uminus[simp]*:

$\langle \text{degree} (-A) x' = \text{degree } A x' \rangle$

**by** (*auto simp: degree-def uminus-mpoly.rep-eq*)

**lemma** *degree-sum-notin*:

$\langle x' \notin \text{vars } B \implies \text{degree} (A + B) x' = \text{degree } A x' \rangle$

**apply** (*auto simp: degree-def*)

**apply** (*rule arg-cong[of - - Max]*)

**apply** (*auto simp: plus-mpoly.rep-eq*)

**apply** (*smt Poly-Mapping.keys-add UN-I UnE image-iff in-keys-iff subsetD vars-def*)

**by** (*smt UN-I add.right-neutral imageI lookup-add not-in-keys-iff-lookup-eq-zero vars-def*)

**lemma** *degree-notin-vars*:

$\langle x \notin \text{vars } B \implies \text{degree } (B :: 'a :: \{\text{monoid-add}\} \text{ mpoly}) \ x = 0 \rangle$   
**using** *degree-sum-notin*[*of x B 0*]  
**by** *auto*

**lemma** *not-in-vars-coeff0*:

$\langle x \notin \text{vars } p \implies \text{MPoly-Type.coeff } p \ (\text{monomial } (\text{Suc } 0) \ x) = 0 \rangle$   
**apply** (*subst not-not[symmetric]*)  
**apply** (*subst coeff-keys[symmetric]*)  
**apply** (*auto simp: vars-def*)  
**done**

**lemma** *keys-mapping-sum-add*:

$\langle \text{finite } A \implies \text{keys } (\text{mapping-of } (\sum v \in A. f \ v)) \subseteq \bigcup (\text{keys } 'f \ ' \text{UNIV}) \rangle$   
**apply** (*induction A rule: finite-induct*)  
**apply** (*auto simp add: zero-mpoly.rep-eq plus-mpoly.rep-eq*  
*keys-plus-ninv-comm-monoid-add*)  
**by** (*metis (no-types, lifting) Poly-Mapping.keys-add UN-E UnE subset-eq*)

**lemma** *vars-sum-vars-union*:

**fixes**  $f :: \langle \text{int mpoly} \Rightarrow \text{int mpoly} \rangle$   
**assumes**  $\langle \text{finite } \{v. f \ v \neq 0\} \rangle$   
**shows**  $\langle \text{vars } (\sum v \mid f \ v \neq 0. f \ v * v) \subseteq \bigcup (\text{vars } 'f \ ' \{v. f \ v \neq 0\}) \cup \bigcup (\text{vars } 'f \ ' \{v. f \ v \neq 0\}) \rangle$   
**(is**  $\langle ?A \subseteq ?B \rangle$ **)**

**proof**

**fix**  $p$   
**assume**  $\langle p \in \text{vars } (\sum v \mid f \ v \neq 0. f \ v * v) \rangle$   
**then obtain**  $x$  **where**  $\langle x \in \text{keys } (\text{mapping-of } (\sum v \mid f \ v \neq 0. f \ v * v)) \rangle$  **and**  
 $p: \langle p \in \text{keys } x \rangle$   
**by** (*auto simp: vars-def times-mpoly.rep-eq simp del: keys-mult*)  
**then have**  $\langle x \in (\bigcup x. \text{keys } (\text{mapping-of } (f \ x) * \text{mapping-of } x)) \rangle$   
**using** *keys-mapping-sum-add*[*of*  $\langle \{v. f \ v \neq 0\} \rangle \langle \lambda x. f \ x * x \rangle$ ] *assms*  
**by** (*auto simp: vars-def times-mpoly.rep-eq*)  
**then have**  $\langle x \in (\bigcup x. \{a+b \mid a \ b. a \in \text{keys } (\text{mapping-of } (f \ x)) \wedge b \in \text{keys } (\text{mapping-of } x)\}) \rangle$   
**using** *Union-mono[OF ] keys-mult* **by** *fast*  
**then show**  $\langle p \in ?B \rangle$   
**using**  $p$  **apply** (*auto simp: keys-add*)  
**by** (*metis (no-types, lifting) Poly-Mapping.keys-add UN-I UnE empty-iff*  
*in-mono keys-zero vars-def zero-mpoly.rep-eq*)

**qed**

**lemma** *vars-in-right-only*:

$x \in \text{vars } q \implies x \notin \text{vars } p \implies x \in \text{vars } (p+q)$   
**apply** (*auto simp: vars-def keys-def plus-mpoly.rep-eq*  
*lookup-plus-fun*)  
**by** (*metis add.left-neutral gr-implies-not0*)

**lemma** [*simp*]:

$\langle \text{vars } 0 = \{\} \rangle$   
**by** (*simp add: vars-def zero-mpoly.rep-eq*)

**lemma** *vars-Un-nointer*:

$\langle \text{keys } (\text{mapping-of } p) \cap \text{keys } (\text{mapping-of } q) = \{\} \implies \text{vars } (p + q) = \text{vars } p \cup \text{vars } q \rangle$



```

apply (auto simp: vars-def)
apply (metis (no-types, hide-lams) Poly-Mapping.keys-add UnE in-mono plus-mpoly.rep-eq)
apply (smt IntI UN-I add.right-neutral coeff-add coeff-keys empty-iff empty-iff in-keys-iff)
apply (smt IntI UN-I add.left-neutral coeff-add coeff-keys empty-iff empty-iff in-keys-iff)
done

```

**lemmas** [simp] = zero-mpoly.rep-eq

**lemma** polynom-sum-monom:

**fixes**  $p :: \langle 'a :: \{ \text{comm-monoid-add}, \text{cancel-comm-monoid-add} \} \text{ mpoly} \rangle$

**shows**

$\langle p = (\sum x \in \text{keys} (\text{mapping-of } p). \text{MPoly-Type.monom } x (\text{MPoly-Type.coeff } p \ x)) \rangle$

$\langle \text{keys} (\text{mapping-of } p) \subseteq I \implies \text{finite } I \implies p = (\sum x \in I. \text{MPoly-Type.monom } x (\text{MPoly-Type.coeff } p \ x)) \rangle$

**proof** –

**define**  $J$  **where**  $\langle J \equiv \text{keys} (\text{mapping-of } p) \rangle$

**define**  $a$  **where**  $\langle a \ x \equiv \text{coeff } p \ x \rangle$  **for**  $x$

**have**  $\langle \text{finite} (\text{keys} (\text{mapping-of } p)) \rangle$

**by** auto

**have**  $\langle p = (\sum x \in I. \text{MPoly-Type.monom } x (\text{MPoly-Type.coeff } p \ x)) \rangle$

**if**  $\langle \text{finite } I \rangle$  **and**  $\langle \text{keys} (\text{mapping-of } p) \subseteq I \rangle$

**for**  $I$

**using** that

**unfolding** a-def

**proof** (induction  $I$  arbitrary:  $p$  rule: finite-induct)

**case** empty

**then have**  $\langle p = 0 \rangle$

**using** empty coeff-all-0 coeff-keys **by** blast

**then show** ?case **using** empty **by** (auto simp: zero-mpoly.rep-eq)

**next**

**case** (insert  $x \ F$ ) **note**  $\text{fin} = \text{this}(1)$  **and**  $\text{xF} = \text{this}(2)$  **and**  $\text{IH} = \text{this}(3)$  **and**  
 $\text{incl} = \text{this}(4)$

**let**  $?p = \langle p - \text{MPoly-Type.monom } x (\text{MPoly-Type.coeff } p \ x) \rangle$

**have**  $\langle ?p = (\sum x a \in F. \text{MPoly-Type.monom } x a (\text{MPoly-Type.coeff } ?p \ x a)) \rangle$

**apply** (rule IH)

**using** incl **apply** auto

**by** (smt Diff-iff Diff-insert-absorb add-diff-cancel-right'

remove-term-keys remove-term-sum subsetD xF)

**also have**  $\langle \dots = (\sum x a \in F. \text{MPoly-Type.monom } x a (\text{MPoly-Type.coeff } p \ x a)) \rangle$

**apply** (use xF in  $\langle \text{auto intro!: sum.cong} \rangle$ )

**by** (metis (mono-tags, hide-lams) add-diff-cancel-right' remove-term-coeff

remove-term-sum when-def)

**finally show** ?case

**using** xF fin **apply** auto

**by** (metis add.commute add-diff-cancel-right' remove-term-sum)

**qed**

**from** this[of  $I$ ] this[of  $J$ ] **show**

$\langle p = (\sum x \in \text{keys} (\text{mapping-of } p). \text{MPoly-Type.monom } x (\text{MPoly-Type.coeff } p \ x)) \rangle$

$\langle \text{keys} (\text{mapping-of } p) \subseteq I \implies \text{finite } I \implies p = (\sum x \in I. \text{MPoly-Type.monom } x (\text{MPoly-Type.coeff } p \ x)) \rangle$

**by** (auto simp: J-def)

**qed**

**lemma** vars-mult-monom:

```

fixes  $p :: \langle \text{int mpoly} \rangle$ 
shows  $\langle \text{vars } (p * (\text{monom } (\text{monomial } (\text{Suc } 0) \ x') \ 1)) = (\text{if } p = 0 \text{ then } \{\} \text{ else insert } x' (\text{vars } p)) \rangle$ 
proof –

  let  $?v = \langle \text{monom } (\text{monomial } (\text{Suc } 0) \ x') \ 1 \rangle$ 
  have
     $p: \langle p = (\sum_{x \in \text{keys } (\text{mapping-of } p)} \text{MPoly-Type.monom } x \ (\text{MPoly-Type.coeff } p \ x)) \rangle$  (is  $\langle - = (\sum_{x \in ?I} ?f \ x) \rangle$ )
    using polynom-sum-monomials(1)[of p] .
  have  $pv: \langle p * ?v = (\sum_{x \in ?I} ?f \ x * ?v) \rangle$ 
    by (subst p) (auto simp: field-simps sum-distrib-left)
  define  $I$  where  $\langle I \equiv ?I \rangle$ 
  have  $\text{in-keysD}: \langle x \in \text{keys } (\text{mapping-of } (\sum_{x \in I} \text{MPoly-Type.monom } x \ (h \ x))) \implies x \in I \rangle$ 
  if  $\langle \text{finite } I \rangle$  for  $I$  and  $h :: \langle - \Rightarrow \text{int} \rangle$  and  $x$ 
    using that by (induction rule: finite-induct)
    (force simp: monom.rep-eq empty-iff insert-iff keys-single coeff-monom
     simp: coeff-keys simp flip: coeff-add
     simp del: coeff-add) +
  have  $\text{in-keys}: \langle \text{keys } (\text{mapping-of } (\sum_{x \in I} \text{MPoly-Type.monom } x \ (h \ x))) = (\bigcup_{x \in I} (\text{if } h \ x = 0 \text{ then } \{\} \text{ else } \{x\})) \rangle$ 
  if  $\langle \text{finite } I \rangle$  for  $I$  and  $h :: \langle - \Rightarrow \text{int} \rangle$  and  $x$ 
    supply  $\text{in-keysD}[dest]$ 
    using that by (induction rule: finite-induct)
    (auto simp: plus-mpoly.rep-eq MPoly-Type-Class.keys-plus-eqI)

  have  $H[\text{simp}]: \langle \text{vars } ((\sum_{x \in I} \text{MPoly-Type.monom } x \ (h \ x))) = (\bigcup_{x \in I} (\text{if } h \ x = 0 \text{ then } \{\} \text{ else keys } x)) \rangle$ 
  if  $\langle \text{finite } I \rangle$  for  $I$  and  $h :: \langle - \Rightarrow \text{int} \rangle$ 
    using that by (auto simp: vars-def in-keys)

  have  $\text{sums}: \langle (\sum_{x \in I} \text{MPoly-Type.monom } (x + a') \ (c \ x)) = (\sum_{x \in (\lambda x. x + a')^{-1} I} \text{MPoly-Type.monom } x \ (c \ (x - a'))) \rangle$ 
  if  $\langle \text{finite } I \rangle$  for  $I \ a' \ c \ q$ 
    using that apply (induction rule: finite-induct)
    subgoal by auto
    subgoal
      unfolding image-insert by (subst sum.insert) auto
    done
  have  $\text{non-zero-keysEx}: \langle p \neq 0 \implies \exists a. a \in \text{keys } (\text{mapping-of } p) \rangle$  for  $p :: \langle \text{int mpoly} \rangle$ 
    using mapping-of-inject by (fastforce simp add: ex-in-conv)
  have  $\langle \text{finite } I \rangle \langle \text{keys } (\text{mapping-of } p) \subseteq I \rangle$ 
    unfolding  $I\text{-def}$  by auto
  then show
     $\langle \text{vars } (p * (\text{monom } (\text{monomial } (\text{Suc } 0) \ x') \ 1)) = (\text{if } p = 0 \text{ then } \{\} \text{ else insert } x' (\text{vars } p)) \rangle$ 
    apply (subst pv, subst I-def[symmetric], subst mult-monom)
    apply (auto simp: mult-monom sums I-def)
    using Poly-Mapping.keys-add vars-def apply fastforce
    apply (auto dest!: non-zero-keysEx)
    apply (rule-tac x = \langle a + monomial (Suc 0) x' \rangle in bexI)
    apply (auto simp: coeff-keys)
    apply (simp add: in-keys-iff lookup-add)
    apply (auto simp: vars-def)
    apply (rule-tac x = \langle xa + monomial (Suc 0) x' \rangle in bexI)

```

```

  apply (auto simp: coeff-keys)
  apply (simp add: in-keys-iff lookup-add)
done
qed

```

**lemma** *in-mapping-mult-single*:

```

  ⟨x ∈ (λx. lookup x x') ' keys (A * (Var0 x' :: (nat ⇒0 nat) ⇒0 'b :: {monoid-mult, zero-neq-one, semiring-0}))⟩
  ⟷
  x > 0 ∧ x - 1 ∈ (λx. lookup x x') ' keys (A)
  apply (auto elim!: in-keys-timesE simp: lookup-add)
  apply (auto simp: keys-def lookup-times-monomial-right Var0-def)
  apply (metis One-nat-def lookup-single-eq lookup-single-not-eq one-neq-zero)
  apply (metis (mono-tags) add-diff-cancel-right' imageI lookup-single-eq lookup-single-not-eq mem-Collect-eq)
  apply (subst image-iff)
  apply (cases x)
  apply simp
  apply (rule-tac x = ⟨x + Poly-Mapping.single x' 1⟩ in bexI)
  apply (auto simp: lookup-add)
done

```

**lemma** *Max-Suc-Suc-Max*:

```

  ⟨finite A ⇒ A ≠ {} ⇒ Max (insert 0 (Suc ' A)) =
    Suc (Max (insert 0 A))⟩
  by (induction rule: finite-induct)
  (auto simp: hom-Max-commute)

```

**lemma** *[simp]*:

```

  ⟨keys (Var0 x' :: ('a ⇒0 nat) ⇒0 'b :: {zero-neq-one}) = {Poly-Mapping.single x' 1}⟩
  by (auto simp: Var0-def)

```

**lemma** *degree-mult-Var*:

```

  ⟨degree (A * Var x') x' = (if A = 0 then 0 else Suc (degree A x'))⟩ for A :: ⟨int mpoly⟩
  apply (auto simp: degree-def times-mpoly.rep-eq)
  apply (subst arg-cong[of - ⟨insert 0
    (Suc ' ((λx. lookup x x') ' keys (mapping-of A)))⟩ Max])
  apply (auto simp: image-image Var.rep-eq lookup-plus-fun in-mapping-mult-single
    hom-Max-commute)
  elim!: in-keys-timesE intro!: Max-Suc-Suc-Max
  split: if-splits[]
  apply (subst Max-Suc-Suc-Max)
  apply auto
  using mapping-of-inject by fastforce

```

**lemma** *degree-mult-Var'*:

```

  ⟨degree (Var x' * A) x' = (if A = 0 then 0 else Suc (degree A x'))⟩ for A :: ⟨int mpoly⟩
  by (simp add: degree-mult-Var semiring-normalization-rules(7))

```

**lemma** *degree-add-max*:

```

  ⟨degree (A + B) x ≤ max (degree A x) (degree B x)⟩
  apply (auto simp: degree-def plus-mpoly.rep-eq
    max-def)
  dest!: set-rev-mp[OF - Poly-Mapping.keys-add])
  by (smt Max-ge dual-order.trans finite-imageI finite-insert finite-keys
    image-subset-iff nat-le-linear subset-insertI)

```

**lemma** *degree-times-le*:

⟨degree (A \* B) x ≤ degree A x + degree B x⟩  
**by** (auto simp: degree-def times-mpoly.rep-eq  
max-def lookup-add add-mono  
dest!: set-rev-mp[OF - Poly-Mapping.keys-add]  
elim!: in-keys-timesE)

**lemma** *monomial-inj*:

monomial c s = monomial (d::'b::zero-neq-one) t ⟷ (c = 0 ∧ d = 0) ∨ (c = d ∧ s = t)  
**apply** (auto simp: monomial-inj Poly-Mapping.single-def  
poly-mapping.Abs-poly-mapping-inject when-def  
cong: if-cong  
split: if-splits)  
**apply** metis  
**apply** metis  
**apply** metis  
**apply** metis  
**done**

**lemma** *MPoly-monomial-power'*:

⟨MPoly (monomial 1 x') ^ (n+1) = MPoly (monomial (1) (((λx. x + x') ^~ n) x'))⟩  
**by** (induction n)  
(auto simp: times-mpoly.abs-eq mult-single ac-simps)

**lemma** *MPoly-monomial-power*:

⟨n > 0 ⟹ MPoly (monomial 1 x') ^ (n) = MPoly (monomial (1) (((λx. x + x') ^~ (n - 1)) x'))⟩  
**using** MPoly-monomial-power'[of - ⟨n-1⟩]  
**by** auto

**lemma** *vars-uminus[simp]*:

⟨vars (-p) = vars p⟩  
**by** (auto simp: vars-def uminus-mpoly.rep-eq)

**lemma** *coeff-uminus[simp]*:

⟨MPoly-Type.coeff (-p) x = -MPoly-Type.coeff p x⟩  
**by** (auto simp: coeff-def uminus-mpoly.rep-eq)

**definition** *decrease-key::'a ⇒ ('a ⇒<sub>0</sub> 'b::{monoid-add, minus, one}) ⇒ ('a ⇒<sub>0</sub> 'b) where*

*decrease-key k0 f = Abs-poly-mapping (λk. if k = k0 ∧ lookup f k ≠ 0 then lookup f k - 1 else lookup f k)*

**lemma** *remove-key-lookup*:

lookup (decrease-key k0 f) k = (if k = k0 ∧ lookup f k ≠ 0 then lookup f k - 1 else lookup f k)  
**unfolding** decrease-key-def **using** finite-subset **apply** (simp add: lookup-Abs-poly-mapping)  
**apply** (subst lookup-Abs-poly-mapping)  
**apply** (auto intro: finite-subset[of - {x. lookup f x ≠ 0}])  
**apply** (subst lookup-Abs-poly-mapping)  
**apply** (auto intro: finite-subset[of - {x. lookup f x ≠ 0}])  
**done**

**lemma** *polynom-split-on-var*:

**fixes** p :: ⟨'a :: {comm-monoid-add, cancel-comm-monoid-add, semiring-0, comm-semiring-1} mpoly⟩

```

obtains  $q\ r$  where
   $\langle p = \text{monom} (\text{monomial} (\text{Suc } 0) x')\ 1 * q + r \rangle$  and
   $\langle x' \notin \text{vars } r \rangle$ 
proof -
  have  $[\text{simp}]$ :  $\langle \{x \in \text{keys} (\text{mapping-of } p). x' \in \text{keys } x\} \cup$ 
     $\{x \in \text{keys} (\text{mapping-of } p). x' \notin \text{keys } x\} = \text{keys} (\text{mapping-of } p) \rangle$ 
  by auto
  have
     $\langle p = (\sum x \in \text{keys} (\text{mapping-of } p). \text{MPoly-Type.monom } x (\text{MPoly-Type.coeff } p\ x)) \rangle$  (is  $\langle - = (\sum x \in ?I.$ 
 $?f\ x) \rangle$ 
    using polynom-sum-monoms(1)[of p] .
    also have  $\langle \dots = (\sum x \in \{x \in ?I. x' \in \text{keys } x\}. ?f\ x) + (\sum x \in \{x \in ?I. x' \notin \text{keys } x\}. ?f\ x) \rangle$  (is  $\langle - =$ 
 $?pX + ?qX \rangle$ 
    by (subst comm-monoid-add-class.sum.union-disjoint[symmetric]) auto
  finally have  $1$ :  $\langle p = ?pX + ?qX \rangle$  .
  have  $H$ :  $\langle 0 < \text{lookup } x\ x' \implies (\lambda k. (\text{if } x' = k \text{ then } \text{Suc } 0 \text{ else } 0) +$ 
     $(\text{if } k = x' \wedge 0 < \text{lookup } x\ k \text{ then } \text{lookup } x\ k - 1$ 
     $\text{else } \text{lookup } x\ k)) = \text{lookup } x\ x' \rangle$  for  $x\ x'$ 
  by auto
  have  $H$ :  $\langle x' \in \text{keys } x \implies \text{monomial} (\text{Suc } 0) x' + \text{Abs-poly-mapping } (\lambda k. \text{if } k = x' \wedge 0 < \text{lookup } x\ k$ 
 $\text{then } \text{lookup } x\ k - 1 \text{ else } \text{lookup } x\ k) = x \rangle$ 
  for  $x$  and  $x' :: \text{nat}$ 
  apply (simp only: keys-def single.abs-eq)
  apply (subst plus-poly-mapping.abs-eq)
  apply (auto simp: eq-onp-def intro!: finite-subset[of  $\langle \{-, - \wedge -\} \rangle \langle \{x a. 0 < \text{lookup } x\ x a\} \rangle]$ )
  apply (smt bounded-nat-set-is-finite lessI mem-Collect-eq neq0-conv when-cong when-neq-zero)
  apply (rule finite-subset[of -  $\langle \{x a. 0 < \text{lookup } x\ x a\} \rangle]$ )
  by (auto simp: when-def H split: if-splits)

  have  $[\text{simp}]$ :  $\langle x' \in \text{keys } x \implies$ 
     $\text{MPoly-Type.monom} (\text{monomial} (\text{Suc } 0) x' + \text{decrease-key } x' x) n =$ 
     $\text{MPoly-Type.monom } x\ n \rangle$  for  $x\ n$  and  $x'$ 
  apply (subst mpoly.mapping-of-inject[symmetric], subst poly-mapping.lookup-inject[symmetric])
  unfolding mapping-of-monom lookup-single
  apply (auto intro!: ext simp: decrease-key-def when-def H)
  done

  have  $pX$ :  $\langle ?pX = \text{monom} (\text{monomial} (\text{Suc } 0) x')\ 1 * (\sum x \in \{x \in ?I. x' \in \text{keys } x\}. \text{MPoly-Type.monom}$ 
 $(\text{decrease-key } x' x) (\text{MPoly-Type.coeff } p\ x)) \rangle$ 
    (is  $\langle - = - * ?pX \rangle$ 
    by (subst sum-distrib-left, subst mult-monom)
    (auto intro!: sum.cong)
  have  $\langle x' \notin \text{vars } ?qX \rangle$ 
  using vars-setsum[of  $\langle \{x. x \in \text{keys} (\text{mapping-of } p) \wedge x' \notin \text{keys } x\} \rangle \langle ?f \rangle]$ 
  by auto (metis (mono-tags, lifting) UN-E mem-Collect-eq subsetD vars-monom-subset)

  then show ?thesis
  using that[of ?pX' ?qX]
  unfolding pX[symmetric] 1[symmetric]
  by blast
qed

```

```

lemma polynom-split-on-var2:
  fixes  $p :: \langle \text{int mpoly} \rangle$ 
  assumes  $\langle x' \notin \text{vars } s \rangle$ 

```

obtains  $q\ r$  where  
 $\langle p = (\text{monom } (\text{monomial } (\text{Suc } 0) \ x') \ 1 - s) * q + r \rangle$  and  
 $\langle x' \notin \text{vars } r \rangle$

**proof** –  
**have**  $\text{eq}[simp]$ :  $\langle \text{monom } (\text{monomial } (\text{Suc } 0) \ x') \ 1 = \text{Var } x' \rangle$   
**by**  $(simp \text{ add: Var.abs-eq Var}_0\text{-def monom.abs-eq})$   
**have**  $\langle \forall m \leq n. \forall P::\text{int mpoly. degree } P \ x' < m \longrightarrow (\exists A \ B. P = (\text{Var } x' - s) * A + B \wedge x' \notin \text{vars } B) \rangle$  **for**  $n$   
**proof**  $(\text{induction } n)$   
**case**  $0$   
**then show**  $?case$  **by**  $\text{auto}$   
**next**  
**case**  $(\text{Suc } n)$   
**then have**  $IH$ :  $\langle m \leq n \implies \text{MPoly-Type.degree } P \ x' < m \implies (\exists A \ B. P = (\text{Var } x' - s) * A + B \wedge x' \notin \text{vars } B) \rangle$  **for**  $m \ P$   
**by**  $\text{fast}$   
**show**  $?case$   
**proof**  $(\text{intro allI impI})$   
**fix**  $m$  **and**  $P :: \langle \text{int mpoly} \rangle$   
**assume**  $\langle m \leq \text{Suc } n \rangle$  **and**  $\text{deg}$ :  $\langle \text{MPoly-Type.degree } P \ x' < m \rangle$   
**consider**  
 $\langle m \leq n \rangle \mid$   
 $\langle m = \text{Suc } n \rangle$   
**using**  $\langle m \leq \text{Suc } n \rangle$  **by**  $\text{linarith}$   
**then show**  $\langle \exists A \ B. P = (\text{Var } x' - s) * A + B \wedge x' \notin \text{vars } B \rangle$   
**proof**  $\text{cases}$   
**case**  $1$   
**then show**  $\langle ?thesis \rangle$   
**using**  $\text{Suc deg}$  **by**  $\text{blast}$   
**next**  
**case**  $[simp]$ :  $2$   
**obtain**  $A \ B$  **where**  
 $P$ :  $\langle P = \text{Var } x' * A + B \rangle$  **and**  
 $\langle x' \notin \text{vars } B \rangle$   
**using**  $\text{polynom-split-on-var}[of \ P \ x']$  **unfolding**  $\text{eq}$  **by**  $\text{blast}$   
**have**  $P'$ :  $\langle P = (\text{Var } x' - s) * A + (s * A + B) \rangle$   
**by**  $(\text{auto simp: field-simps } P)$   
**have**  $\langle A = 0 \vee \text{degree } (s * A) \ x' < \text{degree } P \ x' \rangle$   
**using**  $\text{deg } \langle x' \notin \text{vars } B \rangle \langle x' \notin \text{vars } s \rangle \text{degree-times-le}[of \ s \ A \ x'] \text{deg}$   
**unfolding**  $P$   
**by**  $(\text{auto simp: degree-sum-notin degree-mult-Var' degree-mult-Var degree-notin-vars split: if-splits})$   
**then obtain**  $A' \ B'$  **where**  
 $sA$ :  $\langle s * A = (\text{Var } x' - s) * A' + B' \rangle$  **and**  
 $\langle x' \notin \text{vars } B' \rangle$   
**using**  $IH[of \ \langle m-1 \rangle \ \langle s * A \rangle]$  **deg** **apply**  $\text{auto}$   
**by**  $(metis \ \langle x' \notin \text{vars } B \rangle \text{add.right-neutral mult-zero-right vars-in-right-only})$   
**have**  $\langle P = (\text{Var } x' - s) * (A + A') + (B' + B) \rangle$   
**unfolding**  $P' \ sA$  **by**  $(\text{auto simp: field-simps})$   
**moreover have**  $\langle x' \notin \text{vars } (B' + B) \rangle$   
**using**  $\langle x' \notin \text{vars } B' \rangle \langle x' \notin \text{vars } B \rangle$   
**by**  $(meson \ \text{UnE subset-iff vars-add})$   
**ultimately show**  $?thesis$   
**by**  $\text{fast}$   
**qed**

```

qed
qed
then show ?thesis
  using that unfolding eq
  by blast
qed

lemma polynom-split-on-var-diff-sq2:
fixes p :: ⟨int mpoly⟩
obtains q r s where
  ⟨p = monom (monomial (Suc 0) x') 1 * q + r + s * (monom (monomial (Suc 0) x') 1^2 - monom
(monomial (Suc 0) x') 1)⟩ and
  ⟨x' ∉ vars r⟩ and
  ⟨x' ∉ vars q⟩
proof -
let ?v = ⟨monom (monomial (Suc 0) x') 1 :: int mpoly⟩
have H: ⟨n < m ⟹ n > 0 ⟹ ∃ q. ?v^n = ?v + q * (?v^2 - ?v)⟩ for n m :: nat
proof (induction m arbitrary: n)
case 0
then show ?case by auto
next
case (Suc m n) note IH = this(1-)
consider
  ⟨n < m⟩ |
  ⟨m = n⟩ ⟨n > 1⟩ |
  ⟨n = 1⟩
using IH
by (cases ⟨n < m⟩; cases n) auto
then show ?case
proof cases
case 1
then show ?thesis using IH by auto
next
case 2
have eq: ⟨?v^n = ((?v :: int mpoly) ^ (n-2)) * (?v^2 - ?v) + ?v^(n-1)⟩
  using 2 by (auto simp: field-simps power-eq-if
ideal.scale-right-diff-distrib)
obtain q where
  q: ⟨?v^(n-1) = ?v + q * (?v^2 - ?v)⟩
  using IH(1)[of ⟨n-1⟩] 2
  by auto
show ?thesis
  using q unfolding eq
  by (auto intro!: exI[of - ⟨?v ^ (n - 2) + q⟩] simp: distrib-right)
next
case 3
then show ⟨?thesis⟩
  by auto
qed
qed
have H: ⟨n>0 ⟹ ∃ q. ?v^n = ?v + q * (?v^2 - ?v)⟩ for n
  using H[of n ⟨n+1⟩]
  by auto
obtain qr :: ⟨nat ⇒ int mpoly⟩ where
  qr: ⟨n > 0 ⟹ ?v^n = ?v + qr n * (?v^2 - ?v)⟩ for n

```

```

using  $H[of]$ 
by metis
have  $vn$ :  $\langle (if\ lookup\ x\ x' = 0\ then\ 1\ else\ Var\ x' \wedge lookup\ x\ x') =$ 
 $(if\ lookup\ x\ x' = 0\ then\ 1\ else\ ?v) + (if\ lookup\ x\ x' = 0\ then\ 0\ else\ 1) * qr\ (lookup\ x\ x') * (?v^2 - ?v)\rangle$ 
for  $x$ 
by (simp add: qr[symmetric] Var-def Var0-def monom.abs-eq[symmetric] cong: if-cong)

have  $q$ :  $\langle p = (\sum_{x \in keys} (mapping-of\ p). MPoly-Type.monom\ x\ (MPoly-Type.coeff\ p\ x)) \rangle$ 
by (rule polynom-sum-monoms(1)[of p])
have [simp]:
 $\langle lookup\ x\ x' = 0 \implies$ 
 $Abs-poly-mapping\ (\lambda k. lookup\ x\ k\ when\ k \neq x') = x \rangle$  for  $x$ 
by (cases x, auto simp: poly-mapping.Abs-poly-mapping-inject)
 $(auto\ intro!:\ ext\ simp: when-def)$ 
have [simp]:  $\langle finite\ \{x.\ 0 < (a\ when\ x' = x)\} \rangle$  for  $a :: nat$ 
by (metis (no-types, lifting) infinite-nat-iff-unbounded less-not-refl lookup-single lookup-single-not-eq
 $mem-Collect-eq$ )

have [simp]:  $\langle ((\lambda x. x + monomial\ (Suc\ 0)\ x') \wedge (n))$ 
 $(monomial\ (Suc\ 0)\ x') = Abs-poly-mapping\ (\lambda k. (if\ k = x' then\ n+1\ else\ 0)) \rangle$  for  $n$ 
by (induction n)
 $(auto\ simp: single-def Abs-poly-mapping-inject plus-poly-mapping.abs-eq eq-onp-def cong:if-cong)$ 
have [simp]:  $\langle 0 < lookup\ x\ x' \implies$ 
 $Abs-poly-mapping\ (\lambda k. lookup\ x\ k\ when\ k \neq x') +$ 
 $Abs-poly-mapping\ (\lambda k. if\ k = x' then\ lookup\ x\ x' - Suc\ 0 + 1\ else\ 0) =$ 
 $x \rangle$  for  $x$ 
apply (cases x, auto simp: poly-mapping.Abs-poly-mapping-inject plus-poly-mapping.abs-eq eq-onp-def)
apply (subst plus-poly-mapping.abs-eq)
apply (auto simp: poly-mapping.Abs-poly-mapping-inject plus-poly-mapping.abs-eq eq-onp-def)
apply (metis (no-types, lifting) finite-nat-set-iff-bounded less-numeral-extra(3) mem-Collect-eq when-neq-zero
 $zero-less-iff-neq-zero$ )
apply (subst Abs-poly-mapping-inject)
apply auto
apply (metis (no-types, lifting) finite-nat-set-iff-bounded less-numeral-extra(3) mem-Collect-eq when-neq-zero
 $zero-less-iff-neq-zero$ )
done
define  $f$  where
 $\langle f\ x = (MPoly-Type.monom\ (remove-key\ x'\ x)\ (MPoly-Type.coeff\ p\ x)) *$ 
 $(if\ lookup\ x\ x' = 0\ then\ 1\ else\ Var\ x' \wedge (lookup\ x\ x')) \rangle$  for  $x$ 
have  $f-alt-def$ :  $\langle f\ x = MPoly-Type.monom\ x\ (MPoly-Type.coeff\ p\ x) \rangle$  for  $x$ 
by (auto simp: f-def monom-def remove-key-def Var-def MPoly-monomial-power Var0-def
 $mpoly.MPoly-inject monomial-inj times-mpoly.abs-eq$ 
 $times-mpoly.abs-eq mult-single$ )
have  $p$ :  $\langle p = (\sum_{x \in keys} (mapping-of\ p).$ 
 $MPoly-Type.monom\ (remove-key\ x'\ x)\ (MPoly-Type.coeff\ p\ x) *$ 
 $(if\ lookup\ x\ x' = 0\ then\ 1\ else\ ?v)) +$ 
 $(\sum_{x \in keys} (mapping-of\ p).$ 
 $MPoly-Type.monom\ (remove-key\ x'\ x)\ (MPoly-Type.coeff\ p\ x) *$ 
 $(if\ lookup\ x\ x' = 0\ then\ 0$ 
 $else\ 1) * qr\ (lookup\ x\ x')) *$ 
 $(?v^2 - ?v)\rangle$ 
(is  $\langle - = ?a + ?v2v \rangle$ )
apply (subst q)
unfolding  $f-alt-def[symmetric, abs-def]$   $f-def\ vn\ semiring-class.distrib-left$ 
 $comm-semiring-1-class.semiring-normalization-rules(18)\ semiring-0-class.sum-distrib-right$ 

```



```

by (simp add: semiring-class.distrib-left
    sum.distrib)

have I: ⟨keys (mapping-of p) = {x ∈ keys (mapping-of p). lookup x x' = 0} ∪ {x ∈ keys (mapping-of
p). lookup x x' ≠ 0}⟩
by auto

have ⟨p = (∑ x | x ∈ keys (mapping-of p) ∧ lookup x x' = 0.
    MPoly-Type.monom x (MPoly-Type.coeff p x)) +
    (∑ x | x ∈ keys (mapping-of p) ∧ lookup x x' ≠ 0.
    MPoly-Type.monom (remove-key x' x) (MPoly-Type.coeff p x)) *
    (MPoly-Type.monom (monomial (Suc 0) x') 1) +
    (∑ x | x ∈ keys (mapping-of p) ∧ lookup x x' ≠ 0.
    MPoly-Type.monom (remove-key x' x) (MPoly-Type.coeff p x) *
    qr (lookup x x')) *
    (?v2 - ?v)⟩
(is ⟨p = ?A + ?B * - + ?C * -⟩)
unfolding semiring-0-class.sum-distrib-right[of - - ⟨(MPoly-Type.monom (monomial (Suc 0) x') 1)⟩]
apply (subst p)
apply (subst (2) I)
apply (subst I)
apply (subst comm-monoid-add-class.sum.union-disjoint)
apply auto[3]
apply (subst comm-monoid-add-class.sum.union-disjoint)
apply auto[3]
apply (subst (4) sum.cong[OF refl, of - - ⟨λx. MPoly-Type.monom (remove-key x' x) (MPoly-Type.coeff
p x) *
    qr (lookup x x')⟩])
apply (auto; fail)
apply (subst (3) sum.cong[OF refl, of - - ⟨λx. 0⟩])
apply (auto; fail)
apply (subst (2) sum.cong[OF refl, of - - ⟨λx. MPoly-Type.monom (remove-key x' x) (MPoly-Type.coeff
p x) *
    (MPoly-Type.monom (monomial (Suc 0) x') 1)⟩])
apply (auto; fail)
apply (subst (1) sum.cong[OF refl, of - - ⟨λx. MPoly-Type.monom x (MPoly-Type.coeff p x)⟩])
apply (auto)
by (smt f-alt-def f-def mult-cancel-left1)

moreover have ⟨x' ∉ vars ?A⟩
using vars-setsum[of ⟨{x ∈ keys (mapping-of p). lookup x x' = 0}⟩
    ⟨λx. MPoly-Type.monom x (MPoly-Type.coeff p x)⟩]
apply auto
apply (drule set-rev-mp, assumption)
apply (auto dest!: lookup-eq-zero-in-keys-contradict)
by (meson lookup-eq-zero-in-keys-contradict subsetD vars-monom-subset)
moreover have ⟨x' ∉ vars ?B⟩
using vars-setsum[of ⟨{x ∈ keys (mapping-of p). lookup x x' ≠ 0}⟩
    ⟨λx. MPoly-Type.monom (remove-key x' x) (MPoly-Type.coeff p x)⟩]
apply auto
apply (drule set-rev-mp, assumption)
apply (auto dest!: lookup-eq-zero-in-keys-contradict)
apply (drule subsetD[OF vars-monom-subset])
apply (auto simp: remove-key-keys[symmetric])
done

```

ultimately show *?thesis* apply –  
 apply (rule that[of ?B ?A ?C])  
 apply (auto simp: ac-simps)  
 done  
 qed

lemma *polynom-decomp-alien-var*:

fixes  $q\ A\ b :: \langle \text{int mpoly} \rangle$   
 assumes  
 $q: \langle q = A * (\text{monom} (\text{monomial} (\text{Suc } 0)\ x')\ 1) + b \rangle$  and  
 $x: \langle x' \notin \text{vars } q \rangle \langle x' \notin \text{vars } b \rangle$   
 shows  
 $\langle A = 0 \rangle$  and  
 $\langle q = b \rangle$

proof –

let  $?A = \langle A * (\text{monom} (\text{monomial} (\text{Suc } 0)\ x')\ 1) \rangle$   
 have  $\langle ?A = q - b \rangle$   
 using *arg-cong*[OF  $q$ , of  $\langle \lambda a. a - b \rangle$ ]  
 by *auto*  
 moreover have  $\langle x' \notin \text{vars } (q - b) \rangle$   
 using *x vars-in-right-only*  
 by *fastforce*  
 ultimately have  $\langle x' \notin \text{vars } (?A) \rangle$   
 by *simp*  
 then have  $\langle ?A = 0 \rangle$   
 by (*auto simp: vars-mult-monom split: if-splits*)  
 then show  $\langle A = 0 \rangle$   
 apply *auto*  
 by (*metis (full-types) empty-iff insert-iff mult-zero-right vars-mult-monom*)  
 then show  $\langle q = b \rangle$   
 using  $q$  by *auto*

qed

lemma *polynom-decomp-alien-var2*:

fixes  $q\ A\ b :: \langle \text{int mpoly} \rangle$   
 assumes  
 $q: \langle q = A * (\text{monom} (\text{monomial} (\text{Suc } 0)\ x')\ 1 + p) + b \rangle$  and  
 $x: \langle x' \notin \text{vars } q \rangle \langle x' \notin \text{vars } b \rangle \langle x' \notin \text{vars } p \rangle$   
 shows  
 $\langle A = 0 \rangle$  and  
 $\langle q = b \rangle$

proof –

let  $?x = \langle \text{monom} (\text{monomial} (\text{Suc } 0)\ x')\ 1 \rangle$   
 have  $x'[simp]: \langle ?x = \text{Var } x' \rangle$   
 by (*simp add: Var.abs-eq Var<sub>0</sub>-def monom.abs-eq*)  
 have  $\langle \exists n\ Ax\ A'. A = ?x * Ax + A' \wedge x' \notin \text{vars } A' \wedge \text{degree } Ax\ x' = n \rangle$   
 using *polynom-split-on-var*[of  $A\ x'$ ] by *metis*  
 from *wellorder-class.exists-least-iff*[THEN *iffD1*, OF *this*] obtain  $Ax\ A'\ n$  where  
 $A: \langle A = Ax * ?x + A' \rangle$  and  
 $\langle x' \notin \text{vars } A' \rangle$  and  
 $n: \langle \text{MPoly-Type.degree } Ax\ x' = n \rangle$  and  
 $H: \langle \bigwedge m\ Ax\ A'. m < n \longrightarrow$   
 $A \neq Ax * \text{MPoly-Type.monom} (\text{monomial} (\text{Suc } 0)\ x')\ 1 + A' \vee$   
 $x' \in \text{vars } A' \vee \text{MPoly-Type.degree } Ax\ x' \neq m \rangle$   
 unfolding *wellorder-class.exists-least-iff*[of  $\langle \lambda n. \exists Ax\ A'. A = Ax * ?x + A' \wedge x' \notin \text{vars } A' \wedge$

```

    degree Ax x' = n)]
  by (auto simp: field-simps)

have ⟨q = (A + Ax * p) * monom (monomial (Suc 0) x') 1 + (p * A' + b)⟩
  unfolding q A by (auto simp: field-simps)
moreover have ⟨x' ∉ vars q⟩ ⟨x' ∉ vars (p * A' + b)⟩
  using x ⟨x' ∉ vars A'⟩ apply (auto elim!: )
  by (smt UnE add.assoc add.commute calculation subset-iff vars-in-right-only vars-mult)
ultimately have ⟨A + Ax * p = 0⟩ ⟨q = p * A' + b⟩
  by (rule polynom-decomp-alien-var)+

have A': ⟨A' = -Ax * ?x - Ax * p⟩
  using ⟨A + Ax * p = 0⟩ unfolding A
  by (metis (no-types, lifting) add-uminus-conv-diff eq-neg-iff-add-eq-0 minus-add-cancel mult-minus-left)

have ⟨A = - (Ax * p)⟩
  using A unfolding A'
  apply auto
  done

obtain Axx Ax' where
  Ax: ⟨Ax = ?x * Axx + Ax'⟩ and
  ⟨x' ∉ vars Ax'⟩
  using polynom-split-on-var[of Ax x] by metis

have ⟨A = ?x * (- Axx * p) + (- Ax' * p)⟩
  unfolding ⟨A = - (Ax * p)⟩ Ax
  by (auto simp: field-simps)

moreover have ⟨x' ∉ vars (-Ax' * p)⟩
  using ⟨x' ∉ vars Ax'⟩ by (metis (no-types, hide-lams) UnE add.right-neutral
    add-minus-cancel assms(4) subsetD vars-in-right-only vars-mult)
moreover have ⟨Axx ≠ 0 ⟹ MPoly-Type.degree (- Axx * p) x' < degree Ax x'⟩
  using degree-times-le[of Axx p x] x
  by (auto simp: Ax degree-sum-notin ⟨x' ∉ vars Ax'⟩ degree-mult-Var'
    degree-notin-vars)
ultimately have [simp]: ⟨Axx = 0⟩
  using H[of ⟨MPoly-Type.degree (- Axx * p) x'⟩ ⟨- Axx * p⟩ ⟨- Ax' * p⟩]
  by (auto simp: n)
then have [simp]: ⟨Ax' = Ax⟩
  using Ax by auto

show ⟨A = 0⟩
  using A ⟨A = - (Ax * p)⟩ ⟨x' ∉ vars (- Ax' * p)⟩ ⟨x' ∉ vars A'⟩ polynom-decomp-alien-var(1) by
force
  then show ⟨q = b⟩
    using q by auto
qed

lemma vars-unE: ⟨x ∈ vars (a * b) ⟹ (x ∈ vars a ⟹ thesis) ⟹ (x ∈ vars b ⟹ thesis) ⟹ thesis⟩
  using vars-mult[of a b] by auto

lemma in-keys-minusI1:
  assumes t ∈ keys p and t ∉ keys q

```

**shows**  $t \in \text{keys } (p - q)$   
**using** *assms unfolding in-keys-iff lookup-minus by simp*

**lemma** *in-keys-minusI2*:  
**fixes**  $t :: \langle 'a \rangle$  **and**  $q :: \langle 'a \Rightarrow_0 'b :: \{\text{cancel-comm-monoid-add, group-add}\} \rangle$   
**assumes**  $t \in \text{keys } q$  **and**  $t \notin \text{keys } p$   
**shows**  $t \in \text{keys } (p - q)$   
**using** *assms unfolding in-keys-iff lookup-minus by simp*

**lemma** *in-vars-addE*:  
 $\langle x \in \text{vars } (p + q) \Rightarrow (x \in \text{vars } p \Rightarrow \text{thesis}) \Rightarrow (x \in \text{vars } q \Rightarrow \text{thesis}) \Rightarrow \text{thesis} \rangle$   
**by** (*meson UnE in-mono vars-add*)

**lemma** *lookup-monomial-If*:  
 $\langle \text{lookup } (\text{monomial } v \ k) = (\lambda k'. \text{ if } k = k' \text{ then } v \text{ else } 0) \rangle$   
**by** (*intro ext*)  
*(auto simp: lookup-single-not-eq lookup-single-eq intro!: ext)*

**lemma** *vars-mult-Var*:  
 $\langle \text{vars } (Var \ x * p) = (\text{if } p = 0 \text{ then } \{\} \text{ else insert } x \ (\text{vars } p)) \rangle$  **for**  $p :: \langle \text{int mpoly} \rangle$   
**apply** (*auto simp: vars-def times-mpoly.rep-eq Var.rep-eq elim!: in-keys-timesE*)  
**apply** (*metis add.right-neutral in-keys-iff lookup-add lookup-single-not-eq*)  
**apply** (*auto simp: keys-def lookup-times-monomial-left Var.rep-eq Var\_0-def adds-def*)  
**apply** (*metis (no-types, hide-lams) One-nat-def ab-semigroup-add-class.add commute add-diff-cancel-right' aux lookup-add lookup-single-eq mapping-of-inject neq0-conv one-neq-zero plus-eq-zero-2 zero-mpoly.rep-eq*)  
**by** (*metis ab-semigroup-add-class.add commute add-diff-cancel-left' add-less-same-cancel1 lookup-add neq0-conv not-less0*)

**lemma** *keys-mult-monomial*:  
 $\langle \text{keys } (\text{monomial } (n :: \text{int}) \ k * \text{mapping-of } a) = (\text{if } n = 0 \text{ then } \{\} \text{ else } ((+) \ k) \ ' \text{keys } (\text{mapping-of } a)) \rangle$   
**proof** –  
**have** [*simp*]:  $\langle (\sum aa. (\text{if } k = aa \text{ then } n \text{ else } 0) * (\sum q. \text{lookup } (\text{mapping-of } a) \ q \text{ when } k + xa = aa + q)) = (\sum aa. (\text{if } k = aa \text{ then } n * (\sum q. \text{lookup } (\text{mapping-of } a) \ q \text{ when } k + xa = aa + q) \text{ else } 0)) \rangle$   
**for**  $xa$   
**by** (*smt Sum-any.cong mult-not-zero*)  
**show** *?thesis*  
**apply** (*auto simp: vars-def times-mpoly.rep-eq Const.rep-eq times-poly-mapping.rep-eq Const\_0-def elim!: in-keys-timesE split: if-splits*)  
**apply** (*auto simp: lookup-monomial-If prod-fun-def keys-def times-poly-mapping.rep-eq*)  
**done**  
**qed**

**lemma** *vars-mult-Const*:  
 $\langle \text{vars } (Const \ n * a) = (\text{if } n = 0 \text{ then } \{\} \text{ else vars } a) \rangle$  **for**  $a :: \langle \text{int mpoly} \rangle$   
**by** (*auto simp: vars-def times-mpoly.rep-eq Const.rep-eq keys-mult-monomial Const\_0-def elim!: in-keys-timesE split: if-splits*)

**lemma** *coeff-minus*:  $\text{coeff } p \ m - \text{coeff } q \ m = \text{coeff } (p - q) \ m$   
**by** (*simp add: coeff-def lookup-minus minus-mpoly.rep-eq*)

**lemma** *Const-1-eq-1*:  $\langle \text{Const } (1 :: \text{int}) = (1 :: \text{int mpoly}) \rangle$   
**by** (*simp add: Const.abs-eq Const<sub>0</sub>-one one-mpoly.abs-eq*)

**lemma** [*simp*]:  
 $\langle \text{vars } (1 :: \text{int mpoly}) = \{\} \rangle$   
**by** (*auto simp: vars-def one-mpoly.rep-eq Const-1-eq-1*)

## 2.2 More Ideals

**lemma**  
**fixes**  $A :: \langle ('x \Rightarrow_0 \text{nat}) \Rightarrow_0 'a :: \text{comm-ring-1} \rangle \text{ set}$   
**assumes**  $\langle p \in \text{ideal } A \rangle$   
**shows**  $\langle p * q \in \text{ideal } A \rangle$   
**by** (*metis assms ideal.span-scale semiring-normalization-rules(7)*)

The following theorem is very close to *More-Modules.ideal (insert ?a ?S) = {x.  $\exists k. x - k * ?a \in \text{More-Modules.ideal } ?S$ }*, except that it is more useful if we need to take an element of *More-Modules.ideal (insert a S)*.

**lemma** *ideal-insert'*:  
 $\langle \text{More-Modules.ideal (insert a S)} = \{y. \exists x k. y = x + k * a \wedge x \in \text{More-Modules.ideal } S\} \rangle$   
**apply** (*auto simp: ideal.span-insert*  
*intro: exI[of -  $\langle x - k * a \rangle$ ]*)  
**apply** (*rule-tac x =  $\langle x - k * a \rangle$  in exI*)  
**apply** *auto*  
**apply** (*rule-tac x =  $\langle k \rangle$  in exI*)  
**apply** *auto*  
**done**

**lemma** *ideal-mult-right-in*:  
 $\langle a \in \text{ideal } A \implies a * b \in \text{More-Modules.ideal } A \rangle$   
**by** (*metis ideal.span-scale mult.commute*)

**lemma** *ideal-mult-right-in2*:  
 $\langle a \in \text{ideal } A \implies b * a \in \text{More-Modules.ideal } A \rangle$   
**by** (*metis ideal.span-scale*)

**lemma** [*simp*]:  $\langle \text{vars } (\text{Var } x :: 'a :: \{\text{zero-neq-one}\} \text{ mpoly}) = \{x\} \rangle$   
**by** (*auto simp: vars-def Var.rep-eq Var<sub>0</sub>-def*)

**lemma** *vars-minus-Var-subset*:  
 $\langle \text{vars } (p' - \text{Var } x :: 'a :: \{\text{ab-group-add,one,zero-neq-one}\} \text{ mpoly}) \subseteq \mathcal{V} \implies \text{vars } p' \subseteq \text{insert } x \mathcal{V} \rangle$   
**using** *vars-add[of  $\langle p' - \text{Var } x \rangle \langle \text{Var } x \rangle$ ]*  
**by** *auto*

**lemma** *vars-add-Var-subset*:  
 $\langle \text{vars } (p' + \text{Var } x :: 'a :: \{\text{ab-group-add,one,zero-neq-one}\} \text{ mpoly}) \subseteq \mathcal{V} \implies \text{vars } p' \subseteq \text{insert } x \mathcal{V} \rangle$   
**using** *vars-add[of  $\langle p' + \text{Var } x \rangle \langle - \text{Var } x \rangle$ ]*  
**by** *auto*

**lemma** *coeff-monomila-in-varsD*:  
 $\langle \text{coeff } p (\text{monomial } (\text{Suc } 0) x) \neq 0 \implies x \in \text{vars } (p :: \text{int mpoly}) \rangle$   
**by** (*auto simp: coeff-def keys-def*  
*intro!: exI[of -  $\langle \text{monomial } (\text{Suc } 0) x \rangle$ ]*)

```

lemma (in -)coeff-MPoly-monomila[simp]:
  ⟨Const (MPoly-Type.coeff (MPoly (monomial a m)) m) = Const a⟩
  by (metis MPoly-Type.coeff-def lookup-single-eq monom.abs-eq monom.rep-eq)

end
theory PAC-Specification
  imports PAC-More-Poly
begin

```

### 3 Specification of the PAC checker

#### 3.1 Ideals

```

type-synonym int-poly = ⟨int mpoly⟩
definition polynom-bool :: ⟨int-poly set⟩ where
  ⟨polynom-bool = (λc. Var c ^ 2 - Var c) ‘ UNIV⟩

```

```

definition pac-ideal where
  ⟨pac-ideal A ≡ ideal (A ∪ polynom-bool)⟩

```

```

lemma X2-X-in-pac-ideal:
  ⟨Var c ^ 2 - Var c ∈ pac-ideal A⟩
  unfolding polynom-bool-def pac-ideal-def
  by (auto intro: ideal.span-base)

```

```

lemma pac-idealI1[intro]:
  ⟨p ∈ A ⟹ p ∈ pac-ideal A⟩
  unfolding pac-ideal-def
  by (auto intro: ideal.span-base)

```

```

lemma pac-idealI2[intro]:
  ⟨p ∈ ideal A ⟹ p ∈ pac-ideal A⟩
  using ideal.span-subspace-induct pac-ideal-def by blast

```

```

lemma pac-idealI3[intro]:
  ⟨p ∈ ideal A ⟹ p*q ∈ pac-ideal A⟩
  by (metis ideal.span-scale mult.commute pac-idealI2)

```

```

lemma pac-ideal-Xsq2-iff:
  ⟨Var c ^ 2 ∈ pac-ideal A ⟷ Var c ∈ pac-ideal A⟩
  unfolding pac-ideal-def
  apply (subst (2) ideal.span-add-eq[symmetric, OF X2-X-in-pac-ideal[of c, unfolded pac-ideal-def]])
  apply auto
  done

```

```

lemma diff-in-polynom-bool-pac-idealI:
  assumes a1: p ∈ pac-ideal A
  assumes a2: p - p' ∈ More-Modules.ideal polynom-bool
  shows ⟨p' ∈ pac-ideal A⟩
proof -
  have insert p polynom-bool ⊆ pac-ideal A
    using a1 unfolding pac-ideal-def by (meson ideal.span-superset insert-subset le-sup-iff)
  then show ?thesis
    using a2 unfolding pac-ideal-def by (metis (no-types) ideal.eq-span-insert-eq ideal.span-subset-spanI)

```

*ideal.span-superset insert-subset subsetD*)  
**qed**

**lemma** *diff-in-polynom-bool-pac-idealI2*:  
**assumes** *a1*:  $p \in A$   
**assumes** *a2*:  $p - p' \in \text{More-Modules.ideal polynom-bool}$   
**shows**  $\langle p' \in \text{pac-ideal } A \rangle$   
**using** *diff-in-polynom-bool-pac-idealI*[*OF* - *assms*(2), *of* *A*] *assms*(1)  
**by** (*auto simp: ideal.span-base*)

**lemma** *pac-ideal-alt-def*:  
 $\langle \text{pac-ideal } A = \text{ideal } (A \cup \text{ideal polynom-bool}) \rangle$   
**unfolding** *pac-ideal-def*  
**by** (*meson ideal.span-eq ideal.span-mono ideal.span-superset le-sup-iff subset-trans sup-ge2*)

The equality on ideals is restricted to polynoms whose variable appear in the set of ideals. The function restrict sets:

**definition** *restricted-ideal-to* **where**  
 $\langle \text{restricted-ideal-to } B \ A = \{p \in A. \text{vars } p \subseteq B\} \rangle$

**abbreviation** *restricted-ideal-to<sub>I</sub>* **where**  
 $\langle \text{restricted-ideal-to}_I \ B \ A \equiv \text{restricted-ideal-to } B \ (\text{pac-ideal } (\text{set-mset } A)) \rangle$

**abbreviation** *restricted-ideal-to<sub>V</sub>* **where**  
 $\langle \text{restricted-ideal-to}_V \ B \equiv \text{restricted-ideal-to } (\bigcup (\text{vars } \text{'set-mset } B)) \rangle$

**abbreviation** *restricted-ideal-to<sub>V I</sub>* **where**  
 $\langle \text{restricted-ideal-to}_{V I} \ B \ A \equiv \text{restricted-ideal-to } (\bigcup (\text{vars } \text{'set-mset } B)) \ (\text{pac-ideal } (\text{set-mset } A)) \rangle$

**lemma** *restricted-idealI*:  
 $\langle p \in \text{pac-ideal } (\text{set-mset } A) \implies \text{vars } p \subseteq C \implies p \in \text{restricted-ideal-to}_I \ C \ A \rangle$   
**unfolding** *restricted-ideal-to-def*  
**by** *auto*

**lemma** *pac-ideal-insert-already-in*:  
 $\langle pq \in \text{pac-ideal } (\text{set-mset } A) \implies \text{pac-ideal } (\text{insert } pq \ (\text{set-mset } A)) = \text{pac-ideal } (\text{set-mset } A) \rangle$   
**by** (*auto simp: pac-ideal-alt-def ideal.span-insert-idI*)

**lemma** *pac-ideal-add*:  
 $\langle p \in \# A \implies q \in \# A \implies p + q \in \text{pac-ideal } (\text{set-mset } A) \rangle$   
**by** (*simp add: ideal.span-add ideal.span-base pac-ideal-def*)

**lemma** *pac-ideal-mult*:  
 $\langle p \in \# A \implies p * q \in \text{pac-ideal } (\text{set-mset } A) \rangle$   
**by** (*simp add: ideal.span-base pac-idealI3*)

**lemma** *pac-ideal-mono*:  
 $\langle A \subseteq B \implies \text{pac-ideal } A \subseteq \text{pac-ideal } B \rangle$   
**using** *ideal.span-mono*[*of*  $\langle A \cup \cdot \rangle \langle B \cup \cdot \rangle$ ]  
**by** (*auto simp: pac-ideal-def intro: ideal.span-mono*)

## 3.2 PAC Format

The PAC format contains three kind of steps:

- add that adds up two polynoms that are known.
- mult that multiply a known polynom with another one.
- del that removes a polynom that cannot be reused anymore.

To model the simplification that happens, we add the  $p - p' \in \text{polynom-bool}$  stating that  $p$  and  $p'$  are equivalent.

**type-synonym**  $\text{pac-st} = \langle (\text{nat set} \times \text{int-poly multiset}) \rangle$

**inductive**  $\text{PAC-Format} :: \langle \text{pac-st} \Rightarrow \text{pac-st} \Rightarrow \text{bool} \rangle$  **where**

**add:**

$\langle \text{PAC-Format } (\mathcal{V}, A) (\mathcal{V}, \text{add-mset } p' A) \rangle$

**if**

$\langle p \in \# A \rangle \langle q \in \# A \rangle$   
 $\langle p + q - p' \in \text{ideal polynom-bool} \rangle$   
 $\langle \text{vars } p' \subseteq \mathcal{V} \rangle \mid$

**mult:**

$\langle \text{PAC-Format } (\mathcal{V}, A) (\mathcal{V}, \text{add-mset } p' A) \rangle$

**if**

$\langle p \in \# A \rangle$   
 $\langle p * q - p' \in \text{ideal polynom-bool} \rangle$   
 $\langle \text{vars } p' \subseteq \mathcal{V} \rangle$   
 $\langle \text{vars } q \subseteq \mathcal{V} \rangle \mid$

**del:**

$\langle p \in \# A \implies \text{PAC-Format } (\mathcal{V}, A) (\mathcal{V}, A - \{\#p\# \}) \rangle \mid$

**extend-pos:**

$\langle \text{PAC-Format } (\mathcal{V}, A) (\mathcal{V} \cup \{x' \in \text{vars } (-\text{Var } x + p'). x' \notin \mathcal{V}\}, \text{add-mset } (-\text{Var } x + p') A) \rangle$

**if**

$\langle (p')^2 - p' \in \text{ideal polynom-bool} \rangle$   
 $\langle \text{vars } p' \subseteq \mathcal{V} \rangle$   
 $\langle x \notin \mathcal{V} \rangle$

In the PAC format above, we have a technical condition on the normalisation:  $\text{vars } p' \subseteq \text{vars } (p + q)$  is here to ensure that we don't normalise  $0$  to  $(\text{Var } x)^2 - \text{Var } x$  for a new variable  $x$ . This is completely obvious for the normalisation processe we have in mind when we write the specification, but we must add it explicetely because we are too general.

**lemmas**  $\text{PAC-Format-induct-split} =$

$\text{PAC-Format.induct}[\text{split-format}(\text{complete}), \text{of } V A V' A' \text{ for } V A V' A']$

**lemma**  $\text{PAC-Format-induct}[\text{consumes } 1, \text{case-names add mult del ext}]$ :

**assumes**

$\langle \text{PAC-Format } (\mathcal{V}, A) (\mathcal{V}', A') \rangle$  **and**

**cases:**

$\langle \bigwedge p q p' A \mathcal{V}. p \in \# A \implies q \in \# A \implies p + q - p' \in \text{ideal polynom-bool} \implies \text{vars } p' \subseteq \mathcal{V} \implies P \mathcal{V} A \mathcal{V} (\text{add-mset } p' A) \rangle$

$\langle \bigwedge p q p' A \mathcal{V}. p \in \# A \implies p * q - p' \in \text{ideal polynom-bool} \implies \text{vars } p' \subseteq \mathcal{V} \implies \text{vars } q \subseteq \mathcal{V} \implies P \mathcal{V} A \mathcal{V} (\text{add-mset } p' A) \rangle$

$\langle \bigwedge p A \mathcal{V}. p \in \# A \implies P \mathcal{V} A \mathcal{V} (A - \{\#p\# \}) \rangle$

$\langle \bigwedge p' x r.$

$(p')^2 - p' \in \text{ideal polynom-bool} \implies \text{vars } p' \subseteq \mathcal{V} \implies$

$x \notin \mathcal{V} \implies P \mathcal{V} A (\mathcal{V} \cup \{x' \in \text{vars } (p' - \text{Var } x). x' \notin \mathcal{V}\}) (\text{add-mset } (p' - \text{Var } x) A) \rangle$

**shows**

$\langle P \mathcal{V} A \mathcal{V}' A' \rangle$



**using** *assms*(1) **apply** –  
**by** (*induct*  $V \equiv \mathcal{V} \ A \equiv A \ \mathcal{V}' \ A'$  *rule: PAC-Format-induct-split*)  
*(auto intro: assms*(1) *cases)*

The theorem below (based on the proof ideal by Manuel Kauers) is the correctness theorem of extensions. Remark that the assumption  $\text{vars } q \subseteq \mathcal{V}$  is only used to show that  $x' \notin \text{vars } q$ .

**lemma** *extensions-are-safe*:

**assumes**  $\langle x' \in \text{vars } p \rangle$  **and**  
 $x'$ :  $\langle x' \notin \mathcal{V} \rangle$  **and**  
 $\langle \bigcup (\text{vars } \text{'set-mset } A) \subseteq \mathcal{V} \rangle$  **and**  
 $p$ - $x$ -*coeff*:  $\langle \text{coeff } p \ (\text{monomial } (\text{Suc } 0) \ x') = 1 \rangle$  **and**  
 $\text{vars-}q$ :  $\langle \text{vars } q \subseteq \mathcal{V} \rangle$  **and**  
 $q$ :  $\langle q \in \text{More-Modules.ideal } (\text{insert } p \ (\text{set-mset } A \cup \text{polynom-bool})) \rangle$  **and**  
*leading*:  $\langle x' \notin \text{vars } (p - \text{Var } x') \rangle$  **and**  
*diff*:  $\langle (\text{Var } x' - p)^2 - (\text{Var } x' - p) \in \text{More-Modules.ideal } \text{polynom-bool} \rangle$   
**shows**  
 $\langle q \in \text{More-Modules.ideal } (\text{set-mset } A \cup \text{polynom-bool}) \rangle$

**proof** –

**define**  $p'$  **where**  $\langle p' \equiv p - \text{Var } x' \rangle$   
**let**  $?v = \langle \text{Var } x' :: \text{int mpoly} \rangle$   
**have**  $p$ - $p'$ :  $\langle p = ?v + p' \rangle$   
**by** (*auto simp: p'-def*)  
**define**  $q'$  **where**  $\langle q' \equiv \text{Var } x' - p \rangle$   
**have**  $q$ - $q'$ :  $\langle p = ?v - q' \rangle$   
**by** (*auto simp: q'-def*)  
**have** *diff*:  $\langle q'^2 - q' \in \text{More-Modules.ideal } \text{polynom-bool} \rangle$   
**using** *diff* **unfolding**  $q$ - $q'$  **by** *auto*

**have** [*simp*]:  $\langle \text{vars } ((\text{Var } c)^2 - \text{Var } c :: \text{int mpoly}) = \{c\} \rangle$  **for**  $c$   
**apply** (*auto simp: vars-def Var-def Var<sub>0</sub>-def mpoly.MPoly-inverse keys-def lookup-minus-fun lookup-times-monomial-right single.rep-eq split: if-splits*)  
**apply** (*auto simp: vars-def Var-def Var<sub>0</sub>-def mpoly.MPoly-inverse keys-def lookup-minus-fun lookup-times-monomial-right single.rep-eq when-def ac-simps adds-def lookup-plus-fun power2-eq-square times-mpoly.rep-eq minus-mpoly.rep-eq split: if-splits*)  
**apply** (*rule-tac*  $x = \langle (2 :: \text{nat} \Rightarrow_0 \text{nat}) * \text{monomial } (\text{Suc } 0) \ c \rangle$  **in** *exI*)  
**apply** (*auto dest: monomial-0D simp: plus-eq-zero-2 lookup-plus-fun mult-2*)  
**by** (*meson Suc-neq-Zero monomial-0D plus-eq-zero-2*)

**have** *eq*:  $\langle \text{More-Modules.ideal } (\text{insert } p \ (\text{set-mset } A \cup \text{polynom-bool})) = \text{More-Modules.ideal } (\text{insert } p \ (\text{set-mset } A \cup (\lambda c. \text{Var } c^2 - \text{Var } c) \text{' } \{c. c \neq x'\})) \rangle$   
**(is**  $\langle ?A = ?B \rangle$  **is**  $\langle - = \text{More-Modules.ideal } ?\text{trimmed} \rangle$ )

**proof** –

**let**  $?C = \langle \text{insert } p \ (\text{set-mset } A \cup (\lambda c. \text{Var } c^2 - \text{Var } c) \text{' } \{c. c \neq x'\}) \rangle$   
**let**  $?D = \langle (\lambda c. \text{Var } c^2 - \text{Var } c) \text{' } \{c. c \neq x'\} \rangle$   
**have** *diff*:  $\langle q'^2 - q' \in \text{More-Modules.ideal } ?D \rangle$  **(is**  $\langle ?q \in - \rangle$ )

**proof** –

**obtain**  $r \ t$  **where**  
 $q$ :  $\langle ?q = (\sum a \in t. r \ a * a) \rangle$  **and**  
 $\text{fin-}t$ :  $\langle \text{finite } t \rangle$  **and**  
 $t$ :  $\langle t \subseteq \text{polynom-bool} \rangle$   
**using** *diff* **unfolding** *ideal.span-explicit*  
**by** *auto*  
**show** *?thesis*  
**proof** (*cases*  $\langle ?v^2 - ?v \notin t \rangle$ )

```

case True
then show  $\langle ?thesis \rangle$ 
  using q fin-t t unfolding ideal.span-explicit
  by (auto intro!: exI[of -  $\langle t - \{ ?v^{\wedge}2 - ?v \} \rangle$ ] exI[of - r]
    simp: polynom-bool-def sum-diff1)
next
case False
define t' where  $\langle t' = t - \{ ?v^{\wedge}2 - ?v \} \rangle$ 
have t-t':  $\langle t = \text{insert } (?v^{\wedge}2 - ?v) \ t' \rangle$  and
  notin:  $\langle ?v^{\wedge}2 - ?v \notin t' \rangle$  and
   $\langle t' \subseteq (\lambda c. \text{Var } c^{\wedge}2 - \text{Var } c) \text{ ' } \{ c. c \neq x' \} \rangle$ 
  using False t unfolding t'-def polynom-bool-def by auto
have mon:  $\langle \text{monom } (\text{monomial } (\text{Suc } 0) \ x') \ 1 = \text{Var } x' \rangle$ 
  by (auto simp: coeff-def minus-mpoly.rep-eq Var-def Var0-def monom-def
    times-mpoly.rep-eq lookup-minus lookup-times-monomial-right mpoly.MPoly-inverse)
then have  $\langle \forall a. \exists g \ h. r \ a = ?v * g + h \wedge x' \notin \text{vars } h \rangle$ 
  using polynom-split-on-var[of  $\langle r \rightarrow x' \rangle$ ]
  by metis
then obtain g h where
  r:  $\langle r \ a = ?v * g \ a + h \ a \rangle$  and
  x'-h:  $\langle x' \notin \text{vars } (h \ a) \rangle$  for a
  using polynom-split-on-var[of  $\langle r \ a \rangle x'$ ]
  by metis
have  $\langle ?q = ((\sum a \in t'. g \ a * a) + r \ (?v^{\wedge}2 - ?v) * (?v - 1)) * ?v + (\sum a \in t'. h \ a * a) \rangle$ 
  using fin-t notin unfolding t-t' q r
  by (auto simp: field-simps comm-monoid-add-class.sum.distrib
    power2-eq-square ideal.scale-left-commute sum-distrib-left)
moreover have  $\langle x' \notin \text{vars } ?q \rangle$ 
  by (metis (no-types, hide-lams) Groups.add-ac(2) Un-iff add-diff-cancel-left'
    diff-minus-eq-add in-mono leading q'-def semiring-normalization-rules(29)
    vars-in-right-only vars-mult)
moreover {
  have  $\langle x' \notin (\bigcup m \in t' - \{ ?v^{\wedge}2 - ?v \}. \text{vars } (h \ m * m)) \rangle$ 
    using fin-t x'-h vars-mult[of  $\langle h \rightarrow \rangle$ ]  $\langle t \subseteq \text{polynom-bool} \rangle$ 
    by (auto simp: polynom-bool-def t-t' elim!: vars-unE)
  then have  $\langle x' \notin \text{vars } (\sum a \in t'. h \ a * a) \rangle$ 
    using vars-setsum[of  $\langle t' \rangle \langle \lambda a. h \ a * a \rangle$ ] fin-t x'-h t notin
    by (auto simp: t-t')
}
ultimately have  $\langle ?q = (\sum a \in t'. h \ a * a) \rangle$ 
  unfolding mon[symmetric]
  by (rule polynom-decomp-alien-var(2)[unfolded])
then show  $\langle ?thesis \rangle$ 
  using t fin-t  $\langle t' \subseteq (\lambda c. \text{Var } c^{\wedge}2 - \text{Var } c) \text{ ' } \{ c. c \neq x' \} \rangle$ 
  unfolding ideal.span-explicit t-t'
  by auto
qed
qed
have eq1:  $\langle \text{More-Modules.ideal } (\text{insert } p \ (\text{set-mset } A \cup \text{polynom-bool})) =$ 
  More-Modules.ideal } (\text{insert } (?v^{\wedge}2 - ?v) \ ?C) \rangle
  (is  $\langle \text{More-Modules.ideal } - = \text{More-Modules.ideal } (\text{insert } - \ ?C) \rangle$ )
  by (rule arg-cong[of - - More-Modules.ideal])
  (auto simp: polynom-bool-def)
moreover have  $\langle ?v^{\wedge}2 - ?v \in \text{More-Modules.ideal } ?C \rangle$ 
proof -

```

```

have ⟨?v - q' ∈ More-Modules.ideal ?C⟩
  by (auto simp: q-q' ideal.span-base)
from ideal.span-scale[OF this, of ⟨?v + q' - 1⟩] have ⟨(?v - q') * (?v + q' - 1) ∈ More-Modules.ideal
?C⟩
  by (auto simp: field-simps)
moreover have ⟨q'^2 - q' ∈ More-Modules.ideal ?C⟩
  using diff by (smt Un-insert-right ideal.span-mono insert-subset subsetD sup-ge2)
ultimately have ⟨(?v - q') * (?v + q' - 1) + (q'^2 - q') ∈ More-Modules.ideal ?C⟩
  by (rule ideal.span-add)
moreover have ⟨?v^2 - ?v = (?v - q') * (?v + q' - 1) + (q'^2 - q')⟩
  by (auto simp: p'-def q-q' field-simps power2-eq-square)
ultimately show ?thesis by simp
qed
ultimately show ?thesis
  using ideal.span-insert-idI by blast
qed

```

have ⟨ $n < m \implies n > 0 \implies \exists q. ?v^n = ?v + q * (?v^2 - ?v)$ ⟩ for  $n\ m :: nat$

proof (induction m arbitrary: n)

case 0

then show ?case by auto

next

case (Suc m n) note IH = this(1-)

consider

⟨ $n < m$ ⟩ |

⟨ $m = n$ ⟩ ⟨ $n > 1$ ⟩ |

⟨ $n = 1$ ⟩

using IH

by (cases ⟨ $n < m$ ⟩; cases n) auto

then show ?case

proof cases

case 1

then show ?thesis using IH by auto

next

case 2

have eq: ⟨ $?v^n = ((?v :: int\ mpoly) ^ (n-2)) * (?v^2 - ?v) + ?v^{n-1}$ ⟩

using 2 by (auto simp: field-simps power-eq-if  
ideal.scale-right-diff-distrib)

obtain q where

$q: ?v^{n-1} = ?v + q * (?v^2 - ?v)$

using IH(1)[of ⟨ $n-1$ ⟩] 2

by auto

show ?thesis

using q unfolding eq

by (auto intro!: exI[of - ⟨ $\text{Var } x' ^ (n-2) + q$ ⟩] simp: distrib-right)

next

case 3

then show ⟨?thesis⟩

by auto

qed

qed

obtain r t where

$q: q = (\sum_{a \in t. r\ a * a})$  and

fin-t: ⟨finite t⟩ and

```

t: ⟨t ⊆ ?trimmed⟩
using q unfolding eq unfolding ideal.span-explicit
by auto

define t' where ⟨t' ≡ t - {p}⟩
have t': ⟨t = (if p ∈ t then insert p t' else t')⟩ and
  t''[simp]: ⟨p ∉ t'⟩
  unfolding t'-def by auto
show ?thesis
proof (cases ⟨r p = 0 ∨ p ∉ t⟩)
case True
have
  q: ⟨q = (∑ a ∈ t'. r a * a)⟩ and
  fin-t: ⟨finite t'⟩ and
  t: ⟨t' ⊆ set-mset A ∪ polynom-bool⟩
  using q fin-t t True t''
  apply (subst (asm) t')
  apply (auto intro: sum.cong simp: sum.insert-remove t'-def)
  using q fin-t t True t''
  apply (auto intro: sum.cong simp: sum.insert-remove t'-def polynom-bool-def)
  done
then show ?thesis
  by (auto simp: ideal.span-explicit)
next
case False
then have ⟨r p ≠ 0⟩ and ⟨p ∈ t⟩
  by auto
then have t: ⟨t = insert p t'⟩
  by (auto simp: t'-def)

have ⟨x' ∉ vars (- p')⟩
  using leading p'-def vars-in-right-only by fastforce
have mon: ⟨monom (monomial (Suc 0) x') 1 = Var x'⟩
  by (auto simp: coeff-def minus-mpoly.rep-eq Var-def Var_0-def monom-def
    times-mpoly.rep-eq lookup-minus lookup-times-monomial-right mpoly.MPoly-inverse)
then have ⟨∀ a. ∃ g h. r a = (?v + p') * g + h ∧ x' ∉ vars h⟩
  using polynom-split-on-var2[of x' (-p') ⟨r -⟩] ⟨x' ∉ vars (- p')⟩
  by (metis diff-minus-eq-add)
then obtain g h where
  r: ⟨r a = p * g a + h a⟩ and
  x'-h: ⟨x' ∉ vars (h a)⟩ for a
  using polynom-split-on-var2[of x' p' ⟨r a⟩] unfolding p-p'[symmetric]
  by metis

have ISABLLE-come-on: ⟨a * (p * g a) = p * (a * g a)⟩ for a
  by auto
have q1: ⟨q = p * (∑ a ∈ t'. g a * a) + (∑ a ∈ t'. h a * a) + p * r p⟩
  (is ⟨- = - + ?NOx' + -⟩)
  using fin-t t'' unfolding q t ISABLLE-come-on r
  apply (subst semiring-class.distrib-right)+
  apply (auto simp: comm-monoid-add-class.sum.distrib semigroup-mult-class.mult.assoc
    ISABLLE-come-on simp flip: semiring-0-class.sum-distrib-right
    semiring-0-class.sum-distrib-left)

```

```

  by (auto simp: field-simps)
also have  $\langle \dots = ((\sum a \in t'. g \ a * a) + r \ p) * p + (\sum a \in t'. h \ a * a) \rangle$ 
  by (auto simp: field-simps)
finally have  $q\text{-decomp}: \langle q = ((\sum a \in t'. g \ a * a) + r \ p) * p + (\sum a \in t'. h \ a * a) \rangle$ 
  (is  $\langle q = ?X * p + ?NOx' \rangle$ ).

have [iff]:  $\langle \text{monomial } (Suc \ 0) \ c = 0 \ - \ \text{monomial } (Suc \ 0) \ c = False \rangle$  for  $c$ 
  by (metis One-nat-def diff-is-0-eq' le-eq-less-or-eq less-Suc-eq-le monomial-0-iff single-diff zero-neq-one)
have  $\langle x \in t' \implies x' \in \text{vars } x \implies False \rangle$  for  $x$ 
  using  $\langle t \subseteq ?trimmed \rangle$  t assms(2,3)
  apply (auto simp: polynom-bool-def dest!: multi-member-split)
  apply (frule set-rev-mp)
  apply assumption
  apply (auto dest!: multi-member-split)
  done
then have  $\langle x' \notin (\bigcup m \in t'. \text{vars } (h \ m * m)) \rangle$ 
  using fin-t x'-h vars-mult[of  $\langle h \ - \rangle$ ]
  by (auto simp: t elim!: vars-unE)
then have  $\langle x' \notin \text{vars } ?NOx' \rangle$ 
  using vars-setsum[of  $\langle t' \rangle$   $\langle \lambda a. h \ a * a \rangle$ ] fin-t x'-h
  by (auto simp: t)

moreover {
  have  $\langle x' \notin \text{vars } p' \rangle$ 
    using assms(7)
    unfolding p'-def
    by auto
  then have  $\langle x' \notin \text{vars } (h \ p * p') \rangle$ 
    using vars-mult[of  $\langle h \ p \rangle$   $p'$ ] x'-h
    by auto
}
ultimately have
   $\langle x' \notin \text{vars } q \rangle$ 
   $\langle x' \notin \text{vars } ?NOx' \rangle$ 
   $\langle x' \notin \text{vars } p' \rangle$ 
  using  $x' \text{ vars-} q \text{ vars-add}$ [of  $\langle h \ p * p' \rangle$   $\langle \sum a \in t'. h \ a * a \rangle$ ] x'-h
  leading p'-def
  by auto
then have  $\langle ?X = 0 \rangle$  and  $q\text{-decomp}: \langle q = ?NOx' \rangle$ 
  unfolding mon[symmetric] p-p'
  using polynom-decomp-alien-var2[OF  $q\text{-decomp}$ [unfolded p-p' mon[symmetric]]]
  by auto

then have  $\langle r \ p = (\sum a \in t'. (- \ g \ a) * a) \rangle$ 
  (is  $\langle - = ?CL \rangle$ )
  unfolding add.assoc add-eq-0-iff equation-minus-iff
  by (auto simp: sum-negf ac-simps)

then have  $q2: \langle q = (\sum a \in t'. a * (r \ a - p * g \ a)) \rangle$ 
  using fin-t unfolding q
  apply (auto simp: t r q
    comm-monoid-add-class.sum.distrib[symmetric]
    sum-distrib-left

```

```

      sum-distrib-right
      left-diff-distrib
      intro!: sum.cong)
    apply (auto simp: field-simps)
  done
then show ⟨?thesis⟩
  using t fin-t ⟨ $t \subseteq ?trimmed$ ⟩ unfolding ideal.span-explicit
  by (auto intro!: exI[of - t] exI[of - ⟨ $\lambda a. r\ a - p * g\ a$ ⟩]
      simp: field-simps polynom-bool-def)
qed
qed

lemma extensions-are-safe-uminus:
  assumes ⟨ $x' \in \text{vars } p$ ⟩ and
     $x'$ : ⟨ $x' \notin \mathcal{V}$ ⟩ and
    ⟨ $\bigcup (\text{vars } \text{set-mset } A) \subseteq \mathcal{V}$ ⟩ and
     $p$ -x-coeff: ⟨ $\text{coeff } p (\text{monomial } (\text{Suc } 0) x') = -1$ ⟩ and
    vars-q: ⟨ $\text{vars } q \subseteq \mathcal{V}$ ⟩ and
    q: ⟨ $q \in \text{More-Modules.ideal } (\text{insert } p (\text{set-mset } A \cup \text{polynom-bool}))$ ⟩ and
    leading: ⟨ $x' \notin \text{vars } (p + \text{Var } x')$ ⟩ and
    diff: ⟨ $(\text{Var } x' + p)^2 - (\text{Var } x' + p) \in \text{More-Modules.ideal polynom-bool}$ ⟩
  shows
    ⟨ $q \in \text{More-Modules.ideal } (\text{set-mset } A \cup \text{polynom-bool})$ ⟩
proof -
  have ⟨ $q \in \text{More-Modules.ideal } (\text{insert } (-\ p) (\text{set-mset } A \cup \text{polynom-bool}))$ ⟩
  by (metis ideal.span-breakdown-eq minus-mult-minus q)

  then show ?thesis
    using extensions-are-safe[of  $x' \langle -p \rangle \mathcal{V} A q$ ] assms
    using vars-in-right-only by force
qed

```

This is the correctness theorem of a PAC step: no polynoms are added to the ideal.

```

lemma vars-subst-in-left-only:
  ⟨ $x \notin \text{vars } p \implies x \in \text{vars } (p - \text{Var } x)$ ⟩ for  $p :: \langle \text{int mpoly} \rangle$ 
  by (metis One-nat-def Var.abs-eq Var_0-def group-eq-aux in-vars-addE monom.abs-eq mult-numeral-1
    polynom-decomp-alien-var(1) zero-neq-numeral)

```

```

lemma vars-subst-in-left-only-diff-iff:
  ⟨ $x \notin \text{vars } p \implies \text{vars } (p - \text{Var } x) = \text{insert } x (\text{vars } p)$ ⟩ for  $p :: \langle \text{int mpoly} \rangle$ 
  apply (auto simp: vars-subst-in-left-only)
  apply (metis (no-types, hide-lams) diff-0-right diff-minus-eq-add empty-iff in-vars-addE insert-iff
    keys-single minus-diff-eq
      monom-one mult.right-neutral one-neq-zero single-zero vars-monom-keys vars-mult-Var vars-uminus)
  by (metis add.inverse-inverse diff-minus-eq-add empty-iff insert-iff keys-single minus-diff-eq monom-one
    mult.right-neutral
      one-neq-zero single-zero vars-in-right-only vars-monom-keys vars-mult-Var vars-uminus)

```

```

lemma vars-subst-in-left-only-iff:
  ⟨ $x \notin \text{vars } p \implies \text{vars } (p + \text{Var } x) = \text{insert } x (\text{vars } p)$ ⟩ for  $p :: \langle \text{int mpoly} \rangle$ 
  using vars-subst-in-left-only-diff-iff[of  $x \langle -p \rangle$ ]
  by (metis diff-0 diff-diff-add vars-uminus)

```

```

lemma coeff-add-right-notin:
  ⟨ $x \notin \text{vars } p \implies \text{MPoly-Type.coeff } (\text{Var } x - p) (\text{monomial } (\text{Suc } 0) x) = 1$ ⟩

```

**apply** (*auto simp flip: coeff-minus simp: not-in-vars-coeff0*)  
**by** (*simp add: MPoly-Type.coeff-def Var.rep-eq Var<sub>0</sub>-def*)

**lemma** *coeff-add-left-notin*:

$\langle x \notin \text{vars } p \implies \text{MPoly-Type.coeff } (p - \text{Var } x) (\text{monomial } (\text{Suc } 0) x) = -1 \rangle$  **for**  $p :: \langle \text{int mpoly} \rangle$   
**apply** (*auto simp flip: coeff-minus simp: not-in-vars-coeff0*)  
**by** (*simp add: MPoly-Type.coeff-def Var.rep-eq Var<sub>0</sub>-def*)

**lemma** *ideal-insert-polynom-bool-swap*:  $\langle r - s \in \text{ideal polynom-bool} \implies$

$\text{More-Modules.ideal } (\text{insert } r (A \cup \text{polynom-bool})) = \text{More-Modules.ideal } (\text{insert } s (A \cup \text{polynom-bool})) \rangle$   
**apply** *auto*  
**using** *ideal.eq-span-insert-eq ideal.span-mono sup-ge2* **apply** *blast+*  
**done**

**lemma** *PAC-Format-subset-ideal*:

$\langle \text{PAC-Format } (\mathcal{V}, A) (\mathcal{V}', B) \implies \bigcup (\text{vars } \langle \text{set-mset } A \rangle \subseteq \mathcal{V} \implies$   
 $\text{restricted-ideal-to}_I \mathcal{V} B \subseteq \text{restricted-ideal-to}_I \mathcal{V} A \wedge \mathcal{V} \subseteq \mathcal{V}' \wedge \bigcup (\text{vars } \langle \text{set-mset } B \rangle \subseteq \mathcal{V}') \rangle$

**unfolding** *restricted-ideal-to-def*

**apply** (*induction rule: PAC-Format-induct*)

**subgoal for**  $p \ q \ pq \ A \ \mathcal{V}$

**using** *vars-add*

**by** (*force simp: ideal.span-add-eq ideal.span-base pac-ideal-insert-already-in[OF diff-in-polynom-bool-pac-idealI] of*  
 $\langle p + q \rangle \langle \cdot \rangle pq]$

*pac-ideal-add*

*intro!: diff-in-polynom-bool-pac-idealI[of  $\langle p + q \rangle \langle \cdot \rangle pq]$ )*

**subgoal for**  $p \ q \ pq$

**using** *vars-mult[of  $p \ q$ ]*

**by** (*force simp: ideal.span-add-eq ideal.span-base pac-ideal-mult*

*pac-ideal-insert-already-in[OF diff-in-polynom-bool-pac-idealI[of  $\langle p * q \rangle \langle \cdot \rangle pq]$ ])*

**subgoal for**  $p \ A$

**using** *pac-ideal-mono[of  $\langle \text{set-mset } (A - \{\#p\# \}) \rangle \langle \text{set-mset } A \rangle]$*

**by** (*auto dest: in-diffD*)

**subgoal for**  $p \ x' \ r'$

**apply** (*subgoal-tac  $\langle x' \notin \text{vars } p \rangle$* )

**using** *extensions-are-safe-uminus[of  $x' \leftarrow \text{Var } x' + p \rangle \mathcal{V} \ A]$*  **unfolding** *pac-ideal-def*

**apply** (*auto simp: vars-subst-in-left-only coeff-add-left-notin*)

**done**

**done**

In general, if deletions are disallowed, then the stronger  $B = \text{pac-ideal } A$  holds.

**lemma** *restricted-ideal-to-restricted-ideal-to<sub>I</sub>D*:

$\langle \text{restricted-ideal-to } \mathcal{V} (\text{set-mset } A) \subseteq \text{restricted-ideal-to}_I \mathcal{V} A \rangle$

**by** (*auto simp add: Collect-disj-eq pac-idealI1 restricted-ideal-to-def*)

**lemma** *rtrancp-PAC-Format-subset-ideal*:

$\langle \text{rtrancp PAC-Format } (\mathcal{V}, A) (\mathcal{V}', B) \implies \bigcup (\text{vars } \langle \text{set-mset } A \rangle \subseteq \mathcal{V} \implies$

$\text{restricted-ideal-to}_I \mathcal{V} B \subseteq \text{restricted-ideal-to}_I \mathcal{V} A \wedge \mathcal{V} \subseteq \mathcal{V}' \wedge \bigcup (\text{vars } \langle \text{set-mset } B \rangle \subseteq \mathcal{V}') \rangle$

**apply** (*induction rule: rtrancp-induct[of PAC-Format  $\langle \cdot, \cdot \rangle \langle \cdot, \cdot \rangle$ , split-format(complete)]*)

**subgoal**

**by** (*simp add: restricted-ideal-to-restricted-ideal-to<sub>I</sub>D*)

**subgoal**

**apply** (*drule PAC-Format-subset-ideal*)

**apply** *simp-all*

**apply** *auto*

by (smt Collect-mono-iff mem-Collect-eq restricted-ideal-to-def subset-trans)  
done

end  
theory *Finite-Map-Multiset*  
imports *HOL-Library.Finite-Map Duplicate-Free-Multiset*  
begin

notation *image-mset* (infixr '# 90)

## 4 Finite maps and multisets

### 4.1 Finite sets and multisets

**abbreviation** *mset-fset* ::  $\langle 'a \text{ fset} \Rightarrow 'a \text{ multiset} \rangle$  **where**  
 $\langle \text{mset-fset } N \equiv \text{mset-set } (\text{fset } N) \rangle$

**definition** *fset-mset* ::  $\langle 'a \text{ multiset} \Rightarrow 'a \text{ fset} \rangle$  **where**  
 $\langle \text{fset-mset } N \equiv \text{Abs-fset } (\text{set-mset } N) \rangle$

**lemma** *fset-mset-mset-fset*:  $\langle \text{fset-mset } (\text{mset-fset } N) = N \rangle$   
by (auto simp: *fset.fset-inverse fset-mset-def*)

**lemma** *mset-fset-fset-mset[simp]*:  
 $\langle \text{mset-fset } (\text{fset-mset } N) = \text{remdups-mset } N \rangle$   
by (auto simp: *fset.fset-inverse fset-mset-def Abs-fset-inverse remdups-mset-def*)

**lemma** *in-mset-fset-fmmember[simp]*:  $\langle x \in\# \text{mset-fset } N \longleftrightarrow x \in | N \rangle$   
by (auto simp: *fmmember.rep-eq*)

**lemma** *in-fset-mset-mset[simp]*:  $\langle x \in | \text{fset-mset } N \longleftrightarrow x \in\# N \rangle$   
by (auto simp: *fmmember.rep-eq fset-mset-def Abs-fset-inverse*)

### 4.2 Finite map and multisets

Roughly the same as *ran* and *dom*, but with duplication in the content (unlike their finite sets counterpart) while still working on finite domains (unlike a function mapping). Remark that *dom-m* (the keys) does not contain duplicates, but we keep for symmetry (and for easier use of multiset operators as in the definition of *ran-m*).

**definition** *dom-m* **where**  
 $\langle \text{dom-m } N = \text{mset-fset } (\text{fmdom } N) \rangle$

**definition** *ran-m* **where**  
 $\langle \text{ran-m } N = \text{the } \langle \# \text{fmlookup } N \rangle \# \text{dom-m } N \rangle$

**lemma** *dom-m-fmdrop[simp]*:  $\langle \text{dom-m } (\text{fmdrop } C \ N) = \text{remove1-mset } C \ (\text{dom-m } N) \rangle$   
**unfolding** *dom-m-def*  
by (cases  $\langle C \in | \text{fmdom } N \rangle$ )  
(auto simp: *mset-set.remove fmmember.rep-eq*)

**lemma** *dom-m-fmdrop-All*:  $\langle \text{dom-m } (\text{fmdrop } C \ N) = \text{removeAll-mset } C \ (\text{dom-m } N) \rangle$   
**unfolding** *dom-m-def*  
by (cases  $\langle C \in | \text{fmdom } N \rangle$ )



(auto simp: mset-set.remove fmember.rep-eq)

**lemma** dom-m-fmupd[simp]:  $\langle \text{dom-m } (\text{fmupd } k \ C \ N) = \text{add-mset } k \ (\text{remove1-mset } k \ (\text{dom-m } N)) \rangle$   
**unfolding** dom-m-def  
**by** (cases  $\langle k \in | \text{fmdom } N \rangle$ )  
(auto simp: mset-set.remove fmember.rep-eq mset-set.insert-remove)

**lemma** distinct-mset-dom:  $\langle \text{distinct-mset } (\text{dom-m } N) \rangle$   
**by** (simp add: distinct-mset-mset-set dom-m-def)

**lemma** in-dom-m-lookup-iff:  $\langle C \in \# \text{ dom-m } N' \longleftrightarrow \text{fmlookup } N' \ C \neq \text{None} \rangle$   
**by** (auto simp: dom-m-def fmdom.rep-eq fmlookup-dom'-iff)

**lemma** in-dom-in-ran-m[simp]:  $\langle i \in \# \text{ dom-m } N \implies \text{the } (\text{fmlookup } N \ i) \in \# \text{ ran-m } N \rangle$   
**by** (auto simp: ran-m-def)

**lemma** fmupd-same[simp]:  
 $\langle x1 \in \# \text{ dom-m } x1aa \implies \text{fmupd } x1 \ (\text{the } (\text{fmlookup } x1aa \ x1)) \ x1aa = x1aa \rangle$   
**by** (metis fmap-ext fmupd-lookup in-dom-m-lookup-iff option.collapse)

**lemma** ran-m-fmempty[simp]:  $\langle \text{ran-m } \text{fmempty} = \{\#\} \rangle$  **and**  
 $\text{dom-m-fmempty[simp]: } \langle \text{dom-m } \text{fmempty} = \{\#\} \rangle$   
**by** (auto simp: ran-m-def dom-m-def)

**lemma** fmrestrict-set-fmupd:  
 $\langle a \in xs \implies \text{fmrestrict-set } xs \ (\text{fmupd } a \ C \ N) = \text{fmupd } a \ C \ (\text{fmrestrict-set } xs \ N) \rangle$   
 $\langle a \notin xs \implies \text{fmrestrict-set } xs \ (\text{fmupd } a \ C \ N) = \text{fmrestrict-set } xs \ N \rangle$   
**by** (auto simp: fmfilter-alt-defs)

**lemma** fset-fmdom-fmrestrict-set:  
 $\langle \text{fset } (\text{fmdom } (\text{fmrestrict-set } xs \ N)) = \text{fset } (\text{fmdom } N) \cap xs \rangle$   
**by** (auto simp: fmfilter-alt-defs)

**lemma** dom-m-fmrestrict-set:  $\langle \text{dom-m } (\text{fmrestrict-set } (\text{set } xs) \ N) = \text{mset } xs \cap \# \text{ dom-m } N \rangle$   
**using** fset-fmdom-fmrestrict-set[of  $\langle \text{set } xs \rangle \ N$ ] distinct-mset-dom[of  $N$ ]  
distinct-mset-inter-remdups-mset[of  $\langle \text{mset-fset } (\text{fmdom } N) \rangle \ \langle \text{mset } xs \rangle$ ]  
**by** (auto simp: dom-m-def fset-mset-mset-fset finite-mset-set-inter multiset-inter-commute  
remdups-mset-def)

**lemma** dom-m-fmrestrict-set':  $\langle \text{dom-m } (\text{fmrestrict-set } xs \ N) = \text{mset-set } (xs \cap \text{set-mset } (\text{dom-m } N)) \rangle$   
**using** fset-fmdom-fmrestrict-set[of  $\langle xs \rangle \ N$ ] distinct-mset-dom[of  $N$ ]  
**by** (auto simp: dom-m-def fset-mset-mset-fset finite-mset-set-inter multiset-inter-commute  
remdups-mset-def)

**lemma** indom-mI:  $\langle \text{fmlookup } m \ x = \text{Some } y \implies x \in \# \text{ dom-m } m \rangle$   
**by** (drule fmdomI) (auto simp: dom-m-def fmember.rep-eq)

**lemma** fmupd-fmdrop-id:  
**assumes**  $\langle k \in | \text{fmdom } N' \rangle$   
**shows**  $\langle \text{fmupd } k \ (\text{the } (\text{fmlookup } N' \ k)) \ (\text{fmdrop } k \ N') = N' \rangle$   
**proof** –  
**have** [simp]:  $\langle \text{map-upd } k \ (\text{the } (\text{fmlookup } N' \ k)) \$   
 $(\lambda x. \text{if } x \neq k \text{ then } \text{fmlookup } N' \ x \text{ else None}) =$   
 $\text{map-upd } k \ (\text{the } (\text{fmlookup } N' \ k))$   
 $(\text{fmlookup } N') \rangle$

```

  by (auto intro!: ext simp: map-upd-def)
have [simp]:  $\langle \text{map-upd } k \text{ (the (fmlookup } N' \text{ } k)) \text{ (fmlookup } N') = \text{fmlookup } N' \rangle$ 
  using assms
  by (auto intro!: ext simp: map-upd-def)
have [simp]:  $\langle \text{finite (dom } (\lambda x. \text{ if } x = k \text{ then None else fmlookup } N' \text{ } x)) \rangle$ 
  by (subst dom-if) auto
show ?thesis
  apply (auto simp: fmupd-def fmupd.abs-eq[symmetric])
  unfolding fmlookup-drop
  apply (simp add: fmlookup-inverse)
  done
qed

```

**lemma** *fm-member-split*:  $\langle k \in \text{fmdom } N' \implies \exists N'' v. N' = \text{fmupd } k \text{ } v \text{ } N'' \wedge \text{the (fmlookup } N' \text{ } k) = v \wedge k \notin \text{fmdom } N'' \rangle$   
 by (rule exI[of -  $\langle \text{fmdrop } k \text{ } N' \rangle$ ])  
 (auto simp: fmupd-fmdrop-id)

**lemma**  $\langle \text{fmdrop } k \text{ (fmupd } k \text{ } v \text{ } N'') = \text{fmdrop } k \text{ } N'' \rangle$   
 by (simp add: fmap-ext)

**lemma** *fmap-ext-fmdom*:  
 $\langle \text{fmdom } N = \text{fmdom } N' \implies (\bigwedge x. x \in \text{fmdom } N \implies \text{fmlookup } N \text{ } x = \text{fmlookup } N' \text{ } x) \implies N = N' \rangle$   
 by (rule fmap-ext)  
 (case-tac  $\langle x \in \text{fmdom } N \rangle$ , auto simp: fmdom-notD)

**lemma** *fmrestrict-set-insert-in*:  
 $\langle xa \in \text{fset (fmdom } N) \implies \text{fmrestrict-set (insert } xa \text{ } l1) \text{ } N = \text{fmupd } xa \text{ (the (fmlookup } N \text{ } xa)) (fmrestrict-set } l1 \text{ } N) \rangle$   
 apply (rule fmap-ext-fmdom)  
 apply (auto simp: fset-fmdom-fmrestrict-set fmrestrict-set.rep-eq notin-fset; fail)[]  
 apply (auto simp: fmlookup-dom-iff; fail)  
 done

**lemma** *fmrestrict-set-insert-notin*:  
 $\langle xa \notin \text{fset (fmdom } N) \implies \text{fmrestrict-set (insert } xa \text{ } l1) \text{ } N = \text{fmrestrict-set } l1 \text{ } N \rangle$   
 by (rule fmap-ext-fmdom)  
 (auto simp: fset-fmdom-fmrestrict-set fmrestrict-set.rep-eq notin-fset)

**lemma** *fmrestrict-set-insert-in-dom-m*[simp]:  
 $\langle xa \in \# \text{dom-m } N \implies \text{fmrestrict-set (insert } xa \text{ } l1) \text{ } N = \text{fmupd } xa \text{ (the (fmlookup } N \text{ } xa)) (fmrestrict-set } l1 \text{ } N) \rangle$   
 by (simp add: fmrestrict-set-insert-in dom-m-def)

**lemma** *fmrestrict-set-insert-notin-dom-m*[simp]:  
 $\langle xa \notin \# \text{dom-m } N \implies \text{fmrestrict-set (insert } xa \text{ } l1) \text{ } N = \text{fmrestrict-set } l1 \text{ } N \rangle$   
 by (simp add: fmrestrict-set-insert-notin dom-m-def)

**lemma** *fmlookup-restrict-set-id*:  $\langle \text{fset (fmdom } N) \subseteq A \implies \text{fmrestrict-set } A \text{ } N = N \rangle$   
 by (metis fmap-ext fmdom'-alt-def fmdom'-notD fmlookup-restrict-set subset-iff)

**lemma** *fmlookup-restrict-set-id*:  $\langle \text{set-mset } (\text{dom-m } N) \subseteq A \implies \text{fmrestrict-set } A \ N = N \rangle$   
**by** (*rule fmlookup-restrict-set-id*)  
(auto simp: *dom-m-def*)

**lemma** *ran-m-mapsto-upd*:

**assumes**

*NC*:  $\langle C \in \# \text{ dom-m } N \rangle$

**shows**  $\langle \text{ran-m } (\text{fmupd } C \ C' \ N) = \text{add-mset } C' \ (\text{remove1-mset } (\text{the } (\text{fmlookup } N \ C)) \ (\text{ran-m } N)) \rangle$

**proof** –

**define** *N'* **where**

$\langle N' = \text{fmdrop } C \ N \rangle$

**have** *N-N'*:  $\langle \text{dom-m } N = \text{add-mset } C \ (\text{dom-m } N') \rangle$

**using** *NC* **unfolding** *N'-def* **by** *auto*

**have**  $\langle C \notin \# \text{ dom-m } N' \rangle$

**using** *NC* *distinct-mset-dom*[*of N*] **unfolding** *N-N'* **by** *auto*

**then show** *?thesis*

**by** (*auto simp: N-N' ran-m-def mset-set.insert-remove image-mset-remove1-mset-if*  
*intro!: image-mset-cong*)

**qed**

**lemma** *ran-m-mapsto-upd-notin*:

**assumes** *NC*:  $\langle C \notin \# \text{ dom-m } N \rangle$

**shows**  $\langle \text{ran-m } (\text{fmupd } C \ C' \ N) = \text{add-mset } C' \ (\text{ran-m } N) \rangle$

**using** *NC*

**by** (*auto simp: ran-m-def mset-set.insert-remove image-mset-remove1-mset-if*  
*intro!: image-mset-cong split: if-splits*)

**lemma** *image-mset-If-eq-notin*:

$\langle C \notin \# A \implies \{ \# f \ (\text{if } x = C \ \text{then } a \ x \ \text{else } b \ x). \ x \in \# A \# \} = \{ \# f(b \ x). \ x \in \# A \# \} \rangle$

**by** (*induction A*) *auto*

**lemma** *filter-mset-cong2*:

$(\bigwedge x. x \in \# M \implies f \ x = g \ x) \implies M = N \implies \text{filter-mset } f \ M = \text{filter-mset } g \ N$

**by** (*hypsubst, rule filter-mset-cong, simp*)

**lemma** *ran-m-fmdrop*:

$\langle C \in \# \text{ dom-m } N \implies \text{ran-m } (\text{fmdrop } C \ N) = \text{remove1-mset } (\text{the } (\text{fmlookup } N \ C)) \ (\text{ran-m } N) \rangle$

**using** *distinct-mset-dom*[*of N*]

**by** (*cases (fmlookup N C)*)

(*auto simp: ran-m-def image-mset-If-eq-notin*[*of C - (λx. fst (the x))*])

*dest!:* *multi-member-split*

*intro!:* *filter-mset-cong2 image-mset-cong2*)

**lemma** *ran-m-fmdrop-notin*:

$\langle C \notin \# \text{ dom-m } N \implies \text{ran-m } (\text{fmdrop } C \ N) = \text{ran-m } N \rangle$

**using** *distinct-mset-dom*[*of N*]

**by** (*auto simp: ran-m-def image-mset-If-eq-notin*[*of C - (λx. fst (the x))*])

*dest!:* *multi-member-split*

*intro!:* *filter-mset-cong2 image-mset-cong2*)

**lemma** *ran-m-fmdrop-If*:

$\langle \text{ran-m } (\text{fmdrop } C \ N) = (\text{if } C \in \# \text{ dom-m } N \ \text{then } \text{remove1-mset } (\text{the } (\text{fmlookup } N \ C)) \ (\text{ran-m } N) \ \text{else } \text{ran-m } N) \rangle$

**using** *distinct-mset-dom*[*of N*]

```

by (auto simp: ran-m-def image-mset-If-eq-notin[of C - ⟨λx. fst (the x)⟩]
    dest!: multi-member-split
    intro!: filter-mset-cong2 image-mset-cong2)

```

```

lemma dom-m-empty-iff[iff]:
  ⟨dom-m NU = {#}⟩ ⟷ NU = fmempty
by (cases NU) (auto simp: dom-m-def mset-set.insert-remove)

```

```

end
theory PAC-Map-Rel
imports
  Refine-Imperative-HOL.IICF Finite-Map-Multiset
begin

```

## 5 Hash-Map for finite mappings

This function declares hash-maps for  $(\text{'a}, \text{'b})$  *fmap*, that are nicer to use especially here where everything is finite.

```

definition fmap-rel where
  [to-relAPP]:
  fmap-rel K V ≡ {(m1, m2).
    (∀ i j. i ∈ fmdom m2 ⟶ (j, i) ∈ K ⟶ (the (fmlookup m1 j), the (fmlookup m2 i)) ∈ V) ∧
    fset (fmdom m1) ⊆ Domain K ∧ fset (fmdom m2) ⊆ Range K ∧
    (∀ i j. (i, j) ∈ K ⟶ j ∈ fmdom m2 ⟷ i ∈ fmdom m1)}

```

```

lemma fmap-rel-alt-def:
  ⟨K, V⟩fmap-rel ≡
    {(m1, m2).
      (∀ i j. i ∈# dom-m m2 ⟶
        (j, i) ∈ K ⟶ (the (fmlookup m1 j), the (fmlookup m2 i)) ∈ V) ∧
      fset (fmdom m1) ⊆ Domain K ∧
      fset (fmdom m2) ⊆ Range K ∧
      (∀ i j. (i, j) ∈ K ⟶ (j ∈# dom-m m2) = (i ∈# dom-m m1))}
  ,
  unfolding fmap-rel-def dom-m-def fmember.rep-eq
  by auto

```

```

lemma fmap-rel-empty1-simp[simp]:
  (fmempty, m) ∈ ⟨K, V⟩fmap-rel ⟷ m = fmempty
  apply (cases ⟨fmdom m = {||}⟩)
  apply (auto simp: fmap-rel-def)
  apply (metis fmrestrict-fset-dom fmrestrict-fset-null)
  by (meson RangeE notin-fset subsetD)

```

```

lemma fmap-rel-empty2-simp[simp]:
  (m, fmempty) ∈ ⟨K, V⟩fmap-rel ⟷ m = fmempty
  apply (cases ⟨fmdom m = {||}⟩)
  apply (auto simp: fmap-rel-def)
  apply (metis fmrestrict-fset-dom fmrestrict-fset-null)
  by (meson DomainE notin-fset subset-iff)

```

```

sempref-decl-intf ('k, 'v) f-map is ('k, 'v) fmap

```

**lemma** *[synth-rules]*:  $\llbracket \text{INTF-OF-REL } K \text{ TYPE}('k); \text{INTF-OF-REL } V \text{ TYPE}('v) \rrbracket$   
 $\implies \text{INTF-OF-REL } (\langle K, V \rangle \text{fmap-rel}) \text{ TYPE}((\langle k, 'v \rangle) \text{f-map})$  **by** *simp*

## 5.1 Operations

**sempref-decl-op** *fmap-empty*:  $\text{fmempty} :: \langle K, V \rangle \text{fmap-rel} .$

**sempref-decl-op** *fmap-is-empty*:  $(=) \text{fmempty} :: \langle K, V \rangle \text{fmap-rel} \rightarrow \text{bool-rel}$   
**apply** *(rule fref-ncI)*  
**apply** *parametricity*  
**apply** *(rule fun-relI; auto)*  
**done**

**lemma** *fmap-rel-fmupd-fmap-rel*:  
 $\langle (A, B) \in \langle K, R \rangle \text{fmap-rel} \implies (p, p') \in K \implies (q, q') \in R \implies$   
 $(\text{fmupd } p \ q \ A, \text{fmupd } p' \ q' \ B) \in \langle K, R \rangle \text{fmap-rel} \rangle$   
**if** *single-valued K single-valued ( $K^{-1}$ )*  
**using** *that*  
**unfolding** *fmap-rel-alt-def*  
**apply** *(case-tac  $\langle p' \in \# \text{dom-m } B \rangle$ )*  
**apply** *(auto simp add: all-conj-distrib IS-RIGHT-UNIQUED dest!: multi-member-split)*  
**done**

**sempref-decl-op** *fmap-update*:  $\text{fmupd} :: K \rightarrow V \rightarrow \langle K, V \rangle \text{fmap-rel} \rightarrow \langle K, V \rangle \text{fmap-rel}$   
**where** *single-valued K single-valued ( $K^{-1}$ )*  
**apply** *(rule fref-ncI)*  
**apply** *parametricity*  
**apply** *(intro fun-relI)*  
**by** *(rule fmap-rel-fmupd-fmap-rel)*

**lemma** *fmap-rel-fmdrop-fmap-rel*:  
 $\langle (A, B) \in \langle K, R \rangle \text{fmap-rel} \implies (p, p') \in K \implies$   
 $(\text{fmdrop } p \ A, \text{fmdrop } p' \ B) \in \langle K, R \rangle \text{fmap-rel} \rangle$   
**if** *single-valued K single-valued ( $K^{-1}$ )*  
**using** *that*  
**unfolding** *fmap-rel-alt-def*  
**apply** *(auto simp add: all-conj-distrib IS-RIGHT-UNIQUED dest!: multi-member-split)*  
**apply** *(metis dom-m-fmdrop fmlookup-drop in-dom-m-lookup-iff union-single-eq-member)*  
**apply** *(metis dom-m-fmdrop fmlookup-drop in-dom-m-lookup-iff union-single-eq-member)*  
**by** *(metis IS-RIGHT-UNIQUED converse.intros dom-m-fmdrop fmlookup-drop in-dom-m-lookup-iff union-single-eq-member)+*

**sempref-decl-op** *fmap-delete*:  $\text{fmdrop} :: K \rightarrow \langle K, V \rangle \text{fmap-rel} \rightarrow \langle K, V \rangle \text{fmap-rel}$   
**where** *single-valued K single-valued ( $K^{-1}$ )*  
**apply** *(rule fref-ncI)*  
**apply** *parametricity*  
**by** *(auto simp add: fmap-rel-fmdrop-fmap-rel)*

**lemma** *fmap-rel-nat-the-fmlookup[intro]*:  
 $\langle (A, B) \in \langle S, R \rangle \text{fmap-rel} \implies (p, p') \in S \implies p' \in \# \text{dom-m } B \implies$   
 $(\text{the } (\text{fmlookup } A \ p), \text{the } (\text{fmlookup } B \ p')) \in R \rangle$   
**by** *(auto simp: fmap-rel-alt-def distinct-mset-dom)*

**lemma** *fmap-rel-in-dom-iff*:  
 $\langle (aa, a'a) \in \langle K, V \rangle \text{fmap-rel} \implies$   
 $(a, a') \in K \implies$   
 $a' \in \# \text{ dom-m } a'a \longleftrightarrow$   
 $a \in \# \text{ dom-m } aa$   
**unfolding** *fmap-rel-alt-def*  
**by** *auto*

**lemma** *fmap-rel-fmlookup-rel*:  
 $\langle (a, a') \in K \implies (aa, a'a) \in \langle K, V \rangle \text{fmap-rel} \implies$   
 $(\text{fmlookup } aa \ a, \text{fmlookup } a'a \ a') \in \langle V \rangle \text{option-rel}$   
**using** *fmap-rel-nat-the-fmlookup*[*of aa a'a K V a a'*]  
*fmap-rel-in-dom-iff*[*of aa a'a K V a a'*]  
*in-dom-m-lookup-iff*[*of a' a'a*]  
*in-dom-m-lookup-iff*[*of a aa*]  
**by** (*cases*  $\langle a' \in \# \text{ dom-m } a'a \rangle$ )  
*(auto simp del: fmap-rel-nat-the-fmlookup)*

**sempref-decl-op** *fmap-lookup*:  $\text{fmlookup} :: \langle K, V \rangle \text{fmap-rel} \rightarrow K \rightarrow \langle V \rangle \text{option-rel}$   
**apply** (*rule fref-ncI*)  
**apply** *parametricity*  
**apply** (*intro fun-relI*)  
**apply** (*rule fmap-rel-fmlookup-rel; assumption*)  
**done**

**lemma** *in-fdom-alt*:  $k \in \# \text{ dom-m } m \longleftrightarrow \neg \text{is-None } (\text{fmlookup } m \ k)$   
**apply** (*auto split: option.split intro: fmdom-notI simp: dom-m-def fmlookup.rep-eq*)  
**apply** (*meson fmdom-notI notin-fset*)  
**using** *notin-fset* **by** *fastforce*

**sempref-decl-op** *fmap-contains-key*:  $\lambda k \ m. k \in \# \text{ dom-m } m :: K \rightarrow \langle K, V \rangle \text{fmap-rel} \rightarrow \text{bool-rel}$   
**unfolding** *in-fdom-alt*  
**apply** (*rule fref-ncI*)  
**apply** *parametricity*  
**apply** (*rule fmap-rel-fmlookup-rel; assumption*)  
**done**

## 5.2 Patterns

**lemma** *pat-fmap-empty*[*pat-rules*]:  $\text{fmempty} \equiv \text{op-fmap-empty}$  **by** *simp*

**lemma** *pat-map-is-empty*[*pat-rules*]:  
 $(=) \ \$m\$ \text{fmempty} \equiv \text{op-fmap-is-empty} \$m$   
 $(=) \ \$ \text{fmempty} \$m \equiv \text{op-fmap-is-empty} \$m$   
 $(=) \ \$ (\text{dom-m} \$m) \$ \{ \# \} \equiv \text{op-fmap-is-empty} \$m$   
 $(=) \ \$ \{ \# \} \$ (\text{dom-m} \$m) \equiv \text{op-fmap-is-empty} \$m$   
**unfolding** *atomize-eq*  
**by** (*auto dest: sym*)

**lemma** *op-map-contains-key*[*pat-rules*]:  
 $(\in \#) \ \$ \ k \ \$ (\text{dom-m} \$m) \equiv \text{op-fmap-contains-key} \$'k \$'m$   
**by** (*auto intro!: eq-reflection*)

### 5.3 Mapping to Normal Hashmaps

**abbreviation** *map-of-fmap* ::  $\langle ('k \Rightarrow 'v \text{ option}) \Rightarrow ('k, 'v) \text{ fmap} \rangle$  **where**  
 $\langle \text{map-of-fmap } h \equiv \text{Abs-fmap } h \rangle$

**definition** *map-fmap-rel* **where**  
 $\langle \text{map-fmap-rel} = \text{br } \text{map-of-fmap } (\lambda a. \text{finite } (\text{dom } a)) \rangle$

**lemma** *fmdrop-set-None*:  
 $\langle (\text{op-map-delete}, \text{fmdrop}) \in \text{Id} \rightarrow \text{map-fmap-rel} \rightarrow \text{map-fmap-rel} \rangle$   
**apply** (*auto simp*: *map-fmap-rel-def br-def*)  
**apply** (*subst fmdrop.abs-eq*)  
**apply** (*auto simp*: *eq-onp-def fmap.Abs-fmap-inject*  
*map-drop-def map-filter-finite*  
*intro!*: *ext*)  
**apply** (*auto simp*: *map-filter-def*)  
**done**

**lemma** *map-upd-fmupd*:  
 $\langle (\text{op-map-update}, \text{fmupd}) \in \text{Id} \rightarrow \text{Id} \rightarrow \text{map-fmap-rel} \rightarrow \text{map-fmap-rel} \rangle$   
**apply** (*auto simp*: *map-fmap-rel-def br-def*)  
**apply** (*subst fmupd.abs-eq*)  
**apply** (*auto simp*: *eq-onp-def fmap.Abs-fmap-inject*  
*map-drop-def map-filter-finite map-upd-def*  
*intro!*: *ext*)  
**done**

Technically *op-map-lookup* has the arguments in the wrong direction.

**definition** *fmlookup'* **where**  
 $\langle \text{simp} \rangle: \langle \text{fmlookup}' A k = \text{fmlookup } k A \rangle$

**lemma** [*def-pat-rules*]:  
 $\langle ((\in \#) \$ k \$ (\text{dom-m } \$ A)) \equiv \text{Not} \$ (\text{is-None} \$ (\text{fmlookup}' \$ k \$ A)) \rangle$   
**apply** (*auto split*: *option.split simp*: *dom-m-def*)  
**by** (*smt domIff fmdom.rep-eq option.disc-eq-case(1)*)

**lemma** *op-map-lookup-fmlookup*:  
 $\langle (\text{op-map-lookup}, \text{fmlookup}') \in \text{Id} \rightarrow \text{map-fmap-rel} \rightarrow \langle \text{Id} \rangle \text{option-rel} \rangle$   
**by** (*auto simp*: *map-fmap-rel-def br-def fmap.Abs-fmap-inverse*)

**abbreviation** *hm-fmap-assn* **where**  
 $\langle \text{hm-fmap-assn } K V \equiv \text{hr-comp } (\text{hm.assn } K V) \text{ map-fmap-rel} \rangle$

**lemmas** *fmap-delete-hnr* [*sepref-fr-rules*] =  
*hm.delete-hnr*[*FCOMP fmdrop-set-None*]

**lemmas** *fmap-update-hnr* [*sepref-fr-rules*] =  
*hm.update-hnr*[*FCOMP map-upd-fmupd*]

**lemmas** *fmap-lookup-hnr* [*sepref-fr-rules*] =  
*hm.lookup-hnr*[*FCOMP op-map-lookup-fmlookup*]

**lemma** *fmempty-empty*:

$\langle (\text{uncurry0 } (\text{RETURN } \text{op-map-empty}), \text{uncurry0 } (\text{RETURN } \text{fmempty})) \in \text{unit-rel} \rightarrow_f \langle \text{map-fmap-rel} \rangle \text{nres-rel} \rangle$   
**by** (auto simp: map-fmap-rel-def br-def fmempty-def frefI nres-relI)

**lemmas** [sepref-fr-rules] =  
 hm.empty-hnr[FCOMP fmempty-empty, unfolded op-fmap-empty-def[symmetric]]

**abbreviation** iam-fmap-assn **where**  
 $\langle \text{iam-fmap-assn } K \ V \equiv \text{hr-comp } (\text{iam.assn } K \ V) \ \text{map-fmap-rel} \rangle$

**lemmas** iam-fmap-delete-hnr [sepref-fr-rules] =  
 iam.delete-hnr[FCOMP fmdrop-set-None]

**lemmas** iam-ffmap-update-hnr [sepref-fr-rules] =  
 iam.update-hnr[FCOMP map-upd-fmupd]

**lemmas** iam-ffmap-lookup-hnr [sepref-fr-rules] =  
 iam.lookup-hnr[FCOMP op-map-lookup-fmlookup]

**definition** op-iam-fmap-empty **where**  
 $\langle \text{op-iam-fmap-empty} = \text{fmempty} \rangle$

**lemma** iam-fmempty-empty:  
 $\langle (\text{uncurry0 } (\text{RETURN } \text{op-map-empty}), \text{uncurry0 } (\text{RETURN } \text{op-iam-fmap-empty})) \in \text{unit-rel} \rightarrow_f \langle \text{map-fmap-rel} \rangle \text{nres-rel} \rangle$   
**by** (auto simp: map-fmap-rel-def br-def fmempty-def frefI nres-relI op-iam-fmap-empty-def)

**lemmas** [sepref-fr-rules] =  
 iam.empty-hnr[FCOMP fmempty-empty, unfolded op-iam-fmap-empty-def[symmetric]]

**definition** upper-bound-on-dom **where**  
 $\langle \text{upper-bound-on-dom } A = \text{SPEC}(\lambda n. \forall i \in \#(\text{dom-m } A). i < n) \rangle$

**lemma** [sepref-fr-rules]:  
 $\langle ((\text{Array.len}), \text{upper-bound-on-dom}) \in (\text{iam-fmap-assn } \text{nat-assn } V)^k \rightarrow_a \text{nat-assn} \rangle$

**proof** –

**have** [simp]:  $\langle \text{finite } (\text{dom } b) \implies i \in \text{fset } (\text{fmdom } (\text{map-of-fmap } b)) \longleftrightarrow i \in \text{dom } b \rangle$  **for**  $i \ b$   
**by** (subst fmdom.abs-eq)  
 (auto simp: eq-onp-def fset.Abs-fset-inverse)

**have** 2:  $\langle \text{nat-rel} = \text{the-pure } (\text{nat-assn}) \rangle$  **and**  
 3:  $\langle \text{nat-assn} = \text{pure } \text{nat-rel} \rangle$   
**by** auto

**have** [simp]:  $\langle \text{the-pure } (\lambda a \ c :: \text{nat}. \uparrow (c = a)) = \text{nat-rel} \rangle$   
**apply** (subst 2)  
**apply** (subst 3)  
**apply** (subst pure-def)  
**apply** auto  
**done**

**have** [simp]:  $\langle (\text{iam-of-list } l, b) \in \text{the-pure } (\lambda a \ c :: \text{nat}. \uparrow (c = a)) \rightarrow \langle \text{the-pure } V \rangle \text{option-rel} \implies b \ i = \text{Some } y \implies i < \text{length } l \rangle$  **for**  $i \ b \ l \ y$   
**by** (auto dest!: fun-relD[of - - - i i] simp: option-rel-def iam-of-list-def split: if-splits)

**show** ?thesis



by *seprex-to-hoare*  
 (sep-auto simp: upper-bound-on-dom-def hr-comp-def iam.assn-def map-rel-def  
 map-fmap-rel-def is-iam-def br-def dom-m-def)  
 qed

**lemma** *fmap-rel-nat-rel-dom-m*[simp]:  
 $\langle (A, B) \in \langle \text{nat-rel}, R \rangle \text{fmap-rel} \implies \text{dom-m } A = \text{dom-m } B \rangle$   
 by (subst *distinct-set-mset-eq-iff*[symmetric])  
 (auto simp: *fmap-rel-alt-def* *distinct-mset-dom*  
*simp del: fmap-rel-nat-the-fmlookup*)

**lemma** *ref-two-step'*:  
 $\langle A \leq B \implies \Downarrow R A \leq \Downarrow R B \rangle$   
 using *ref-two-step* by auto

end  
**theory** *PAC-Checker-Specification*  
 imports *PAC-Specification*  
         *Refine-Imperative-HOL.IICF*  
         *Finite-Map-Multiset*  
 begin

## 6 Checker Algorithm

In this level of refinement, we define the first level of the implementation of the checker, both with the specification as on ideals and the first version of the loop.

### 6.1 Specification

**datatype** *status* =  
*is-failed*: *FAILED* |  
*is-success*: *SUCCESS* |  
*is-found*: *FOUND*

**lemma** *is-success-alt-def*:  
 $\langle \text{is-success } a \longleftrightarrow a = \text{SUCCESS} \rangle$   
 by (cases *a*) auto

**datatype** ('a, 'b, 'lbls) *pac-step* =  
*Add* (pac-src1: 'lbls) (pac-src2: 'lbls) (new-id: 'lbls) (pac-res: 'a) |  
*Mult* (pac-src1: 'lbls) (pac-mult: 'a) (new-id: 'lbls) (pac-res: 'a) |  
*Extension* (new-id: 'lbls) (new-var: 'b) (pac-res: 'a) |  
*Del* (pac-src1: 'lbls)

**type-synonym** *pac-state* =  $\langle (\text{nat set} \times \text{int-poly multiset}) \rangle$

**definition** *PAC-checker-specification*

$:: \langle \text{int-poly} \Rightarrow \text{int-poly multiset} \Rightarrow (\text{status} \times \text{nat set} \times \text{int-poly multiset}) \text{ nres} \rangle$

**where**

$\langle \text{PAC-checker-specification spec } A = \text{SPEC}(\lambda(b, \mathcal{V}, B).$

$(\neg \text{is-failed } b \longrightarrow \text{restricted-ideal-to}_I (\bigcup (\text{vars } \text{'set-mset } A) \cup \text{vars spec}) B \subseteq \text{restricted-ideal-to}_I$   
 $(\bigcup (\text{vars } \text{'set-mset } A) \cup \text{vars spec}) A) \wedge$

$\langle \text{is-found } b \longrightarrow \text{spec} \in \text{pac-ideal } (\text{set-mset } A) \rangle\rangle$

**definition** *PAC-checker-specification-spec*

$\langle \text{int-poly} \Rightarrow \text{pac-state} \Rightarrow (\text{status} \times \text{pac-state}) \Rightarrow \text{bool} \rangle$

**where**

$\langle \text{PAC-checker-specification-spec } \text{spec} = (\lambda(\mathcal{V}, A) (b, B). (\neg \text{is-failed } b \longrightarrow \bigcup (\text{vars } \text{'set-mset } A) \subseteq \mathcal{V}) \wedge$   
 $(\text{is-success } b \longrightarrow \text{PAC-Format}^{**}(\mathcal{V}, A) B) \wedge$   
 $(\text{is-found } b \longrightarrow \text{PAC-Format}^{**}(\mathcal{V}, A) B \wedge \text{spec} \in \text{pac-ideal } (\text{set-mset } A))) \rangle$

**abbreviation** *PAC-checker-specification2*

$\langle \text{int-poly} \Rightarrow (\text{nat set} \times \text{int-poly multiset}) \Rightarrow (\text{status} \times (\text{nat set} \times \text{int-poly multiset})) \text{ nres} \rangle$

**where**

$\langle \text{PAC-checker-specification2 } \text{spec } A \equiv \text{SPEC}(\text{PAC-checker-specification-spec } \text{spec } A) \rangle$

**definition** *PAC-checker-specification-step-spec*

$\langle \text{pac-state} \Rightarrow \text{int-poly} \Rightarrow \text{pac-state} \Rightarrow (\text{status} \times \text{pac-state}) \Rightarrow \text{bool} \rangle$

**where**

$\langle \text{PAC-checker-specification-step-spec} = (\lambda(\mathcal{V}_0, A_0) \text{spec } (\mathcal{V}, A) (b, B).$   
 $(\text{is-success } b \longrightarrow$   
 $\bigcup (\text{vars } \text{'set-mset } A_0) \subseteq \mathcal{V}_0 \wedge$   
 $\bigcup (\text{vars } \text{'set-mset } A) \subseteq \mathcal{V} \wedge \text{PAC-Format}^{**}(\mathcal{V}_0, A_0) (\mathcal{V}, A) \wedge \text{PAC-Format}^{**}(\mathcal{V}, A) B) \wedge$   
 $(\text{is-found } b \longrightarrow$   
 $\bigcup (\text{vars } \text{'set-mset } A_0) \subseteq \mathcal{V}_0 \wedge$   
 $\bigcup (\text{vars } \text{'set-mset } A) \subseteq \mathcal{V} \wedge \text{PAC-Format}^{**}(\mathcal{V}_0, A_0) (\mathcal{V}, A) \wedge \text{PAC-Format}^{**}(\mathcal{V}, A) B \wedge$   
 $\text{spec} \in \text{pac-ideal } (\text{set-mset } A_0))) \rangle$

**abbreviation** *PAC-checker-specification-step2*

$\langle \text{pac-state} \Rightarrow \text{int-poly} \Rightarrow \text{pac-state} \Rightarrow (\text{status} \times \text{pac-state}) \text{ nres} \rangle$

**where**

$\langle \text{PAC-checker-specification-step2 } A_0 \text{spec } A \equiv \text{SPEC}(\text{PAC-checker-specification-step-spec } A_0 \text{spec } A) \rangle$

**definition** *normalize-poly-spec*  $\langle \cdot \rangle$  **where**

$\langle \text{normalize-poly-spec } p = \text{SPEC } (\lambda r. p - r \in \text{ideal polynom-bool} \wedge \text{vars } r \subseteq \text{vars } p) \rangle$

**lemma** *normalize-poly-spec-alt-def*:

$\langle \text{normalize-poly-spec } p = \text{SPEC } (\lambda r. r - p \in \text{ideal polynom-bool} \wedge \text{vars } r \subseteq \text{vars } p) \rangle$

**unfolding** *normalize-poly-spec-def*

**by** (*auto dest: ideal.span-neg*)

**definition** *mult-poly-spec*  $\langle \text{int mpoly} \Rightarrow \text{int mpoly} \Rightarrow \text{int mpoly nres} \rangle$  **where**

$\langle \text{mult-poly-spec } p \ q = \text{SPEC } (\lambda r. p * q - r \in \text{ideal polynom-bool}) \rangle$

**definition** *check-add*  $\langle (\text{nat}, \text{int mpoly}) \text{ fmap} \Rightarrow \text{nat set} \Rightarrow \text{nat} \Rightarrow \text{nat} \Rightarrow \text{nat} \Rightarrow \text{int mpoly} \Rightarrow \text{bool nres} \rangle$  **where**

$\langle \text{check-add } A \ \mathcal{V} \ p \ q \ i \ r =$   
 $\text{SPEC}(\lambda b. b \longrightarrow p \in \# \text{ dom-m } A \wedge q \in \# \text{ dom-m } A \wedge i \notin \# \text{ dom-m } A \wedge \text{vars } r \subseteq \mathcal{V} \wedge$   
 $\text{the } (\text{fmlookup } A \ p) + \text{the } (\text{fmlookup } A \ q) - r \in \text{ideal polynom-bool}) \rangle$

**definition** *check-mult*  $\langle (\text{nat}, \text{int mpoly}) \text{ fmap} \Rightarrow \text{nat set} \Rightarrow \text{nat} \Rightarrow \text{int mpoly} \Rightarrow \text{nat} \Rightarrow \text{int mpoly} \Rightarrow \text{bool nres} \rangle$  **where**

$\langle \text{check-mult } A \ \mathcal{V} \ p \ q \ i \ r =$   
 $\text{SPEC}(\lambda b. b \longrightarrow p \in \# \text{ dom-m } A \wedge i \notin \# \text{ dom-m } A \wedge \text{vars } q \subseteq \mathcal{V} \wedge \text{vars } r \subseteq \mathcal{V} \wedge$   
 $\text{the } (\text{fmlookup } A \ p) * q - r \in \text{ideal polynom-bool}) \rangle$

**definition** *check-extension* ::  $\langle (nat, int\ mpoly) \text{ fmap} \Rightarrow nat\ set \Rightarrow nat \Rightarrow nat \Rightarrow int\ mpoly \Rightarrow (bool)\ nres \rangle$  **where**

$\langle check-extension\ A\ \mathcal{V}\ i\ v\ p =$   
 $\quad SPEC(\lambda b. b \longrightarrow (i \notin \# dom-m\ A \wedge$   
 $\quad (v \notin \mathcal{V} \wedge$   
 $\quad (p + Var\ v)^2 - (p + Var\ v) \in ideal\ polynom\ bool \wedge$   
 $\quad vars\ (p + Var\ v) \subseteq \mathcal{V})) \rangle$

**fun** *merge-status* **where**

$\langle merge-status\ (FAILED) = FAILED \rangle \mid$   
 $\langle merge-status - (FAILED) = FAILED \rangle \mid$   
 $\langle merge-status\ FOUND = FOUND \rangle \mid$   
 $\langle merge-status - FOUND = FOUND \rangle \mid$   
 $\langle merge-status - - = SUCCESS \rangle$

**type-synonym** *fpac-step* =  $\langle nat\ set \times (nat, int\ poly) \text{ fmap} \rangle$

**definition** *check-del* ::  $\langle (nat, int\ mpoly) \text{ fmap} \Rightarrow nat \Rightarrow bool\ nres \rangle$  **where**

$\langle check-del\ A\ p =$   
 $\quad SPEC(\lambda b. b \longrightarrow True) \rangle$

## 6.2 Algorithm

**definition** *PAC-checker-step*

::  $\langle int\ poly \Rightarrow (status \times fpac\ step) \Rightarrow (int\ poly, nat, nat) \text{ pac}\ step \Rightarrow$   
 $\quad (status \times fpac\ step) \text{ nres} \rangle$

**where**

$\langle PAC-checker-step = (\lambda spec\ (stat, (\mathcal{V}, A))\ st. \text{ case } st \text{ of}$   
 $\quad Add\ -\ -\ -\ - \Rightarrow$   
 $\quad \text{do } \{$   
 $\quad \quad r \leftarrow normalize\ poly\ spec\ (pac\ res\ st);$   
 $\quad \quad eq \leftarrow check\ add\ A\ \mathcal{V}\ (pac\ src1\ st)\ (pac\ src2\ st)\ (new\ id\ st)\ r;$   
 $\quad \quad st' \leftarrow SPEC(\lambda st'. (\neg is\ failed\ st' \wedge is\ found\ st' \longrightarrow r - spec \in ideal\ polynom\ bool));$   
 $\quad \quad \text{if } eq$   
 $\quad \quad \text{then } RETURN\ (merge\ status\ stat\ st',$   
 $\quad \quad \quad \mathcal{V}, \text{ fmu}pd\ (new\ id\ st)\ r\ A)$   
 $\quad \quad \text{else } RETURN\ (FAILED, (\mathcal{V}, A))$   
 $\quad \}$   
 $\mid Del\ - \Rightarrow$   
 $\quad \text{do } \{$   
 $\quad \quad eq \leftarrow check\ del\ A\ (pac\ src1\ st);$   
 $\quad \quad \text{if } eq$   
 $\quad \quad \text{then } RETURN\ (stat, (\mathcal{V}, \text{ fmd}rop\ (pac\ src1\ st)\ A))$   
 $\quad \quad \text{else } RETURN\ (FAILED, (\mathcal{V}, A))$   
 $\quad \}$   
 $\mid Mult\ -\ -\ -\ - \Rightarrow$   
 $\quad \text{do } \{$   
 $\quad \quad r \leftarrow normalize\ poly\ spec\ (pac\ res\ st);$   
 $\quad \quad q \leftarrow normalize\ poly\ spec\ (pac\ mult\ st);$   
 $\quad \quad eq \leftarrow check\ mult\ A\ \mathcal{V}\ (pac\ src1\ st)\ q\ (new\ id\ st)\ r;$   
 $\quad \quad st' \leftarrow SPEC(\lambda st'. (\neg is\ failed\ st' \wedge is\ found\ st' \longrightarrow r - spec \in ideal\ polynom\ bool));$   
 $\quad \quad \text{if } eq$   
 $\quad \quad \text{then } RETURN\ (merge\ status\ stat\ st',$   
 $\quad \quad \quad \mathcal{V}, \text{ fmu}pd\ (new\ id\ st)\ r\ A)$   
 $\quad \quad \text{else } RETURN\ (FAILED, (\mathcal{V}, A))$   
 $\quad \}$

```

}
| Extension - - - =>
  do {
    r ← normalize-poly-spec (pac-res st - Var (new-var st));
    (eq) ← check-extension A V (new-id st) (new-var st) r;
    if eq
    then do {
      RETURN (stat,
        insert (new-var st) V, fmupd (new-id st) (r) A)}
    else RETURN (FAILED, (V, A))
  }
)»

```

**definition** *polys-rel* ::  $\langle (nat, int \text{ mpoly}) \text{ fmap} \times - \rangle \text{ set} \rangle$  **where**  
 $\langle polys-rel = \{(A, B). B = (ran-m A)\} \rangle$

**definition** *polys-rel-full* ::  $\langle (nat \text{ set} \times (nat, int \text{ mpoly}) \text{ fmap}) \times - \rangle \text{ set} \rangle$  **where**  
 $\langle polys-rel-full = \{((V, A), (V', B)). (A, B) \in polys-rel \wedge V = V'\} \rangle$

**lemma** *polys-rel-update-remove*:

```

⟨x13 ∉ #dom-m A ⟹ x11 ∈ # dom-m A ⟹ x12 ∈ # dom-m A ⟹ x11 ≠ x12 ⟹ (A,B) ∈ polys-rel
=>
  (fmupd x13 r (fmdrop x11 (fmdrop x12 A)),
    add-mset r B - {#the (fmlookup A x11), the (fmlookup A x12)#})
  ∈ polys-rel
⟨x13 ∉ #dom-m A ⟹ x11 ∈ # dom-m A ⟹ (A,B) ∈ polys-rel ⟹
  (fmupd x13 r (fmdrop x11 A), add-mset r B - {#the (fmlookup A x11)#})
  ∈ polys-rel
⟨x13 ∉ #dom-m A ⟹ (A,B) ∈ polys-rel ⟹
  (fmupd x13 r A, add-mset r B) ∈ polys-rel
⟨x13 ∈ #dom-m A ⟹ (A,B) ∈ polys-rel ⟹
  (fmdrop x13 A, remove1-mset (the (fmlookup A x13)) B) ∈ polys-rel
using distinct-mset-dom[of A]
apply (auto simp: polys-rel-def ran-m-mapsto-upd ran-m-mapsto-upd-notin
  ran-m-fmdrop)
apply (subst ran-m-mapsto-upd-notin)
apply (auto dest: in-diffD dest!: multi-member-split simp: ran-m-fmdrop ran-m-fmdrop-If distinct-mset-remove1-All
  ran-m-def
    add-mset-eq-add-mset removeAll-notin
    split: if-splits intro!: image-mset-cong)
by (smt count-inI diff-single-trivial fmlookup-drop image-mset-cong2 replicate-mset-0)

```

**lemma** *polys-rel-in-dom-inD*:

```

⟨(A, B) ∈ polys-rel ⟹
  x12 ∈ # dom-m A ⟹
  the (fmlookup A x12) ∈ # B
by (auto simp: polys-rel-def)

```

**lemma** *PAC-Format-add-and-remove*:

```

⟨r - x14 ∈ More-Modules.ideal polynom-bool ⟹
  (A, B) ∈ polys-rel ⟹
  x12 ∈ # dom-m A ⟹
  x13 ∉ # dom-m A ⟹
  vars r ⊆ V ⟹
  2 * the (fmlookup A x12) - r ∈ More-Modules.ideal polynom-bool ⟹

```

$PAC-Format^{**} (\mathcal{V}, B) (\mathcal{V}, \text{remove1-mset } (the (fmlookup A x12)) (add-mset r B)) \rangle$   
 $\langle r - x14 \in \text{More-Modules.ideal polynom-bool} \implies$   
 $(A, B) \in \text{polys-rel} \implies$   
 $the (fmlookup A x11) + the (fmlookup A x12) - r \in \text{More-Modules.ideal polynom-bool} \implies$   
 $x11 \in \# \text{ dom-m } A \implies$   
 $x12 \in \# \text{ dom-m } A \implies$   
 $\text{vars } r \subseteq \mathcal{V} \implies$   
 $PAC-Format^{**} (\mathcal{V}, B) (\mathcal{V}, \text{add-mset } r B) \rangle$   
 $\langle r - x14 \in \text{More-Modules.ideal polynom-bool} \implies$   
 $(A, B) \in \text{polys-rel} \implies$   
 $x11 \in \# \text{ dom-m } A \implies$   
 $x12 \in \# \text{ dom-m } A \implies$   
 $the (fmlookup A x11) + the (fmlookup A x12) - r \in \text{More-Modules.ideal polynom-bool} \implies$   
 $\text{vars } r \subseteq \mathcal{V} \implies$   
 $x11 \neq x12 \implies$   
 $PAC-Format^{**} (\mathcal{V}, B)$   
 $(\mathcal{V}, \text{add-mset } r B - \{\#the (fmlookup A x11), the (fmlookup A x12)\# \}) \rangle$   
 $\langle (A, B) \in \text{polys-rel} \implies$   
 $r - x34 \in \text{More-Modules.ideal polynom-bool} \implies$   
 $x11 \in \# \text{ dom-m } A \implies$   
 $the (fmlookup A x11) * x32 - r \in \text{More-Modules.ideal polynom-bool} \implies$   
 $\text{vars } x32 \subseteq \mathcal{V} \implies$   
 $\text{vars } r \subseteq \mathcal{V} \implies$   
 $PAC-Format^{**} (\mathcal{V}, B) (\mathcal{V}, \text{add-mset } r B) \rangle$   
 $\langle (A, B) \in \text{polys-rel} \implies$   
 $r - x34 \in \text{More-Modules.ideal polynom-bool} \implies$   
 $x11 \in \# \text{ dom-m } A \implies$   
 $the (fmlookup A x11) * x32 - r \in \text{More-Modules.ideal polynom-bool} \implies$   
 $\text{vars } x32 \subseteq \mathcal{V} \implies$   
 $\text{vars } r \subseteq \mathcal{V} \implies$   
 $PAC-Format^{**} (\mathcal{V}, B) (\mathcal{V}, \text{remove1-mset } (the (fmlookup A x11)) (add-mset r B)) \rangle$   
 $\langle (A, B) \in \text{polys-rel} \implies$   
 $x12 \in \# \text{ dom-m } A \implies$   
 $PAC-Format^{**} (\mathcal{V}, B) (\mathcal{V}, \text{remove1-mset } (the (fmlookup A x12)) B) \rangle$   
 $\langle (A, B) \in \text{polys-rel} \implies$   
 $(p' + \text{Var } x)^2 - (p' + \text{Var } x) \in \text{ideal polynom-bool} \implies$   
 $x \notin \mathcal{V} \implies$   
 $x \notin \text{vars}(p' + \text{Var } x) \implies$   
 $\text{vars}(p' + \text{Var } x) \subseteq \mathcal{V} \implies$   
 $PAC-Format^{**} (\mathcal{V}, B)$   
 $(\text{insert } x \mathcal{V}, \text{add-mset } p' B) \rangle$

**subgoal**

**apply** (rule converse-rtrancpl-into-rtrancpl)  
**apply** (rule PAC-Format.add[of  $\langle the (fmlookup A x12) \rangle B \langle the (fmlookup A x12) \rangle$ ])  
**apply** (auto dest: polys-rel-in-dom-inD)  
**apply** (rule converse-rtrancpl-into-rtrancpl)  
**apply** (rule PAC-Format.del[of  $\langle the (fmlookup A x12) \rangle$ ])  
**apply** (auto dest: polys-rel-in-dom-inD)  
**done**

**subgoal H2**

**apply** (rule converse-rtrancpl-into-rtrancpl)  
**apply** (rule PAC-Format.add[of  $\langle the (fmlookup A x11) \rangle B \langle the (fmlookup A x12) \rangle$ ])  
**apply** (auto dest: polys-rel-in-dom-inD)  
**done**

**subgoal**

```

  apply (rule rtrancpl-trans)
  apply (rule H2; assumption)
  apply (rule converse-rtrancpl-into-rtrancpl)
  apply (rule PAC-Format.del[of ⟨the (fmlookup A x12)⟩])
  apply (auto dest: polys-rel-in-dom-inD)
  apply (rule converse-rtrancpl-into-rtrancpl)
  apply (rule PAC-Format.del[of ⟨the (fmlookup A x11)⟩])
  apply (auto dest: polys-rel-in-dom-inD)
  apply (auto simp: polys-rel-def ran-m-def add-mset-eq-add-mset dest!: multi-member-split)
done
subgoal H2
  apply (rule converse-rtrancpl-into-rtrancpl)
  apply (rule PAC-Format.mult[of ⟨the (fmlookup A x11)⟩ B ⟨x32⟩ r])
  apply (auto dest: polys-rel-in-dom-inD)
done
subgoal
  apply (rule rtrancpl-trans)
  apply (rule H2; assumption)
  apply (rule converse-rtrancpl-into-rtrancpl)
  apply (rule PAC-Format.del[of ⟨the (fmlookup A x11)⟩])
  apply (auto dest: polys-rel-in-dom-inD)
done
subgoal
  apply (rule converse-rtrancpl-into-rtrancpl)
  apply (rule PAC-Format.del[of ⟨the (fmlookup A x12)⟩ B])
  apply (auto dest: polys-rel-in-dom-inD)
done
subgoal
  apply (rule converse-rtrancpl-into-rtrancpl)
  apply (rule PAC-Format.extend-pos[of ⟨p' + Var x⟩ - x])
  using coeff-monomila-in-varsD[of ⟨p' - Var x⟩ x]
  apply (auto dest: polys-rel-in-dom-inD simp: vars-in-right-only vars-subst-in-left-only)
  apply (subgoal-tac ⟨V ∪ {x' ∈ vars (p'). x' ∉ V} = insert x V⟩)
  apply simp
  using coeff-monomila-in-varsD[of p' x]
  apply (auto dest: vars-add-Var-subset vars-minus-Var-subset polys-rel-in-dom-inD simp: vars-subst-in-left-only-iff)
  using vars-in-right-only vars-subst-in-left-only by force
done

```

**abbreviation** *status-rel* ::  $\langle (status \times status) \text{ set} \rangle$  **where**  
 $\langle status\text{-rel} \equiv Id \rangle$

**lemma** *is-merge-status*[*simp*]:  
 $\langle is\text{-failed} (merge\text{-status } a \ st') \longleftrightarrow is\text{-failed } a \vee is\text{-failed } st' \rangle$   
 $\langle is\text{-found} (merge\text{-status } a \ st') \longleftrightarrow \neg is\text{-failed } a \wedge \neg is\text{-failed } st' \wedge (is\text{-found } a \vee is\text{-found } st') \rangle$   
 $\langle is\text{-success} (merge\text{-status } a \ st') \longleftrightarrow (is\text{-success } a \wedge is\text{-success } st') \rangle$   
**by** (cases *a*; cases *st'*; auto; fail)+

**lemma** *status-rel-merge-status*:  
 $\langle (merge\text{-status } a \ b, \text{SUCCESS}) \notin status\text{-rel} \longleftrightarrow$   
 $(a = \text{FAILED}) \vee (b = \text{FAILED}) \vee$   
 $a = \text{FOUND} \vee (b = \text{FOUND}) \rangle$   
**by** (cases *a*; cases *b*; auto)

**lemma** *Ex-status-iff*:

⟨(∃ a. P a) ⟷ P SUCCESS ∨ P FOUND ∨ (P (FAILED))⟩  
**apply** *auto*  
**apply** (case-tac a; *auto*)  
**done**

**lemma** *is-failed-alt-def*:

⟨is-failed st' ⟷ ¬is-success st' ∧ ¬is-found st'⟩  
**by** (cases st') *auto*

**lemma** *merge-status-eq-iff*[simp]:

⟨merge-status a SUCCESS = SUCCESS ⟷ a = SUCCESS⟩  
⟨merge-status a SUCCESS = FOUND ⟷ a = FOUND⟩  
⟨merge-status SUCCESS a = SUCCESS ⟷ a = SUCCESS⟩  
⟨merge-status SUCCESS a = FOUND ⟷ a = FOUND⟩  
⟨merge-status SUCCESS a = FAILED ⟷ a = FAILED⟩  
⟨merge-status a SUCCESS = FAILED ⟷ a = FAILED⟩  
⟨merge-status FOUND a = FAILED ⟷ a = FAILED⟩  
⟨merge-status a FOUND = FAILED ⟷ a = FAILED⟩  
⟨merge-status a FOUND = SUCCESS ⟷ False⟩  
⟨merge-status a b = FOUND ⟷ (a = FOUND ∨ b = FOUND) ∧ (a ≠ FAILED ∧ b ≠ FAILED)⟩  
**apply** (cases a; *auto*; *fail*) +  
**apply** (cases a; cases b; *auto*; *fail*) +  
**done**

**lemma** *fmdrop-irrelevant*: ⟨x11 ∉ # dom-m A ⟹ fmdrop x11 A = A⟩

**by** (simp add: fmap-ext in-dom-m-lookup-iff)

**lemma** *PAC-checker-step-PAC-checker-specification2*:

**fixes** a :: ⟨status⟩

**assumes** AB: ⟨(V, A), (V<sub>B</sub>, B)⟩ ∈ polys-rel-full **and**

⟨¬is-failed a⟩ **and**

[simp, intro]: ⟨a = FOUND ⟹ spec ∈ pac-ideal (set-mset A<sub>0</sub>)⟩ **and**

A<sub>0</sub>B: ⟨PAC-Format\*\* (V<sub>0</sub>, A<sub>0</sub>) (V, B)⟩ **and**

spec<sub>0</sub>: ⟨vars spec ⊆ V<sub>0</sub>⟩ **and**

vars-A<sub>0</sub>: ⟨⋃ (vars ' set-mset A<sub>0</sub>) ⊆ V<sub>0</sub>⟩

**shows** ⟨PAC-checker-step spec (a, (V, A)) st ≤ ↓ (status-rel ×<sub>r</sub> polys-rel-full) (PAC-checker-specification-step2 (V<sub>0</sub>, A<sub>0</sub>) spec (V, B))⟩

**proof** –

**have**

⟨V<sub>B</sub> = V⟩ **and**

[simp, intro]: ⟨(A, B) ∈ polys-rel⟩

**using** AB

**by** (auto simp: polys-rel-full-def)

**have** H1: ⟨x12 ∈ # dom-m A ⟹

2 \* the (fmlookup A x12) - r ∈ More-Modules.ideal polynom-bool ⟹

r - spec ∈ More-Modules.ideal polynom-bool ⟹

vars spec ⊆ vars r ⟹

spec ∈ pac-ideal (set-mset B)⟩ **for** x12 r

**using** ⟨(A, B) ∈ polys-rel⟩

ideal.span-add[OF ideal.span-add[OF ideal.span-neg ideal.span-neg,

of (the (fmlookup A x12)) - (the (fmlookup A x12))],

of (set-mset B ∪ polynom-bool) ⟨2 \* the (fmlookup A x12) - r⟩]

**unfolding** polys-rel-def

**apply** (subgoal-tac ⟨r ∈ pac-ideal (set-mset B)⟩)

```

apply (auto dest!: multi-member-split simp: ran-m-def intro: diff-in-polynom-bool-pac-idealI)
by (metis (no-types, lifting) ab-semigroup-mult-class.mult.commute diff-in-polynom-bool-pac-idealI
    ideal.span-base pac-idealI3 set-image-mset set-mset-add-mset-insert union-single-eq-member)

have H2:  $\langle x11 \in \# \text{ dom-}m \ A \implies$ 
   $x12 \in \# \text{ dom-}m \ A \implies$ 
   $\text{the } (fmllookup \ A \ x11) + \text{the } (fmllookup \ A \ x12) - r$ 
   $\in \text{More-Modules.ideal polynom-bool} \implies$ 
   $r - \text{spec} \in \text{More-Modules.ideal polynom-bool} \implies$ 
   $\text{spec} \in \text{pac-ideal } (\text{set-mset } B) \rangle \text{ for } x12 \ r \ x11$ 
using  $\langle (A,B) \in \text{polys-rel} \rangle$ 
   $\text{ideal.span-add}[OF \ \text{ideal.span-add}[OF \ \text{ideal.span-neg } \text{ideal.span-neg},$ 
     $\text{of } (\text{the } (fmllookup \ A \ x11)) - (\text{the } (fmllookup \ A \ x12))],$ 
     $\text{of } (\text{set-mset } B \cup \text{polynom-bool}) \ (\text{the } (fmllookup \ A \ x11) + \text{the } (fmllookup \ A \ x12) - r)]$ 
unfolding polys-rel-def
apply (subgoal-tac  $\langle r \in \text{pac-ideal } (\text{set-mset } B) \rangle$ )
apply (auto dest!: multi-member-split simp: ran-m-def ideal.span-base intro: diff-in-polynom-bool-pac-idealI)
by (metis (mono-tags, lifting) Un-insert-left diff-diff-eq2 diff-in-polynom-bool-pac-idealI diff-zero
    ideal.span-diff ideal.span-neg minus-diff-eq pac-idealI1 pac-ideal-def set-image-mset
    set-mset-add-mset-insert union-single-eq-member)

have H3:  $\langle x12 \in \# \text{ dom-}m \ A \implies$ 
   $\text{the } (fmllookup \ A \ x12) * q - r \in \text{More-Modules.ideal polynom-bool} \implies$ 
   $r - \text{spec} \in \text{More-Modules.ideal polynom-bool} \implies$ 
   $\text{spec} \in \text{pac-ideal } (\text{set-mset } B) \rangle \text{ for } x12 \ r \ q$ 
using  $\langle (A,B) \in \text{polys-rel} \rangle$ 
   $\text{ideal.span-add}[OF \ \text{ideal.span-add}[OF \ \text{ideal.span-neg } \text{ideal.span-neg},$ 
     $\text{of } (\text{the } (fmllookup \ A \ x12)) - (\text{the } (fmllookup \ A \ x12))],$ 
     $\text{of } (\text{set-mset } B \cup \text{polynom-bool}) \ (2 * \text{the } (fmllookup \ A \ x12) - r)]$ 
unfolding polys-rel-def
apply (subgoal-tac  $\langle r \in \text{pac-ideal } (\text{set-mset } B) \rangle$ )
apply (auto dest!: multi-member-split simp: ran-m-def intro: diff-in-polynom-bool-pac-idealI)
by (metis (no-types, lifting) ab-semigroup-mult-class.mult.commute diff-in-polynom-bool-pac-idealI
    ideal.span-base pac-idealI3 set-image-mset set-mset-add-mset-insert union-single-eq-member)

have [intro]:  $\langle \text{spec} \in \text{pac-ideal } (\text{set-mset } B) \implies \text{spec} \in \text{pac-ideal } (\text{set-mset } A_0) \rangle$  and
   $\text{vars-B: } \langle \bigcup (\text{vars } ' \text{ set-mset } B) \subseteq \mathcal{V} \rangle$  and
   $\text{vars-B: } \langle \bigcup (\text{vars } ' \text{ set-mset } (\text{ran-}m \ A)) \subseteq \mathcal{V} \rangle$ 
using rtrancpl-PAC-Format-subset-ideal[OF  $A_0 B \ \text{vars-}A_0$ ] spec0  $\langle (A, B) \in \text{polys-rel} \rangle$  [unfolded
  polys-rel-def, simplified]
by (smt in-mono mem-Collect-eq restricted-ideal-to-def)+

have eq-successI:  $\langle st' \neq \text{FAILED} \implies$ 
   $st' \neq \text{FOUND} \implies st' = \text{SUCCESS} \rangle$  for  $st'$ 
by (cases  $st'$ ) auto
have vars-diff-inv:  $\langle \text{vars } (Var \ x2 - r) = \text{vars } (r - Var \ x2 :: \text{int mpoly}) \rangle$  for  $x2 \ r$ 
using vars-uminus[of  $\langle Var \ x2 - r \rangle$ ]
by (auto simp del: vars-uminus)
have vars-add-inv:  $\langle \text{vars } (Var \ x2 + r) = \text{vars } (r + Var \ x2 :: \text{int mpoly}) \rangle$  for  $x2 \ r$ 
unfolding add.commute[of  $\langle Var \ x2 \rangle \ r$ ] ..

have [iff]:  $\langle a \neq \text{FAILED} \rangle$  and
  [intro]:  $\langle a \neq \text{SUCCESS} \implies a = \text{FOUND} \rangle$  and
  [simp]:  $\langle \text{merge-status } a \ \text{FOUND} = \text{FOUND} \rangle$ 
using assms(2) by (cases  $a$ ; auto)+

```



```

note  $[[goals-limit=1]]$ 
show ?thesis
unfolding PAC-checker-step-def PAC-checker-specification-step-spec-def
normalize-poly-spec-alt-def check-mult-def check-add-def
check-extension-def polys-rel-full-def
apply (cases st)
apply clarsimp-all
subgoal for x11 x12 x13 x14
apply (refine-vcg lhs-step-If)
subgoal for r eqa st'
using assms vars-B apply –
apply (rule RETURN-SPEC-refine)
apply (rule-tac x = (merge-status a st',  $\mathcal{V}$ , add-mset r B) in exI)
by (auto simp: polys-rel-update-remove ran-m-mapsto-upd-notin
intro: PAC-Format-add-and-remove H2 dest: rtranclp-PAC-Format-subset-ideal)
subgoal
by (rule RETURN-SPEC-refine)
(auto simp: Ex-status-iff dest: rtranclp-PAC-Format-subset-ideal)
done
subgoal for x11 x12 x13 x14
apply (refine-vcg lhs-step-If)
subgoal for r q eqa st'
using assms vars-B apply –
apply (rule RETURN-SPEC-refine)
apply (rule-tac x = (merge-status a st',  $\mathcal{V}$ , add-mset r B) in exI)
by (auto intro: polys-rel-update-remove intro: PAC-Format-add-and-remove(3–) H3
dest: rtranclp-PAC-Format-subset-ideal)
subgoal
by (rule RETURN-SPEC-refine)
(auto simp: Ex-status-iff)
done
subgoal for x31 x32 x34
apply (refine-vcg lhs-step-If)
subgoal for r x
using assms vars-B apply –
apply (rule RETURN-SPEC-refine)
apply (rule-tac x = (a, insert x32  $\mathcal{V}$ , add-mset r B) in exI)
apply (auto simp: intro!: polys-rel-update-remove PAC-Format-add-and-remove(5–)
dest: rtranclp-PAC-Format-subset-ideal)
done
subgoal
by (rule RETURN-SPEC-refine)
(auto simp: Ex-status-iff)
done
subgoal for x11
unfolding check-del-def
apply (refine-vcg lhs-step-If)
subgoal for eq
using assms vars-B apply –
apply (rule RETURN-SPEC-refine)
apply (cases (x11  $\in$  # dom-m A))
subgoal
apply (rule-tac x = (a,  $\mathcal{V}$ , remove1-mset (the (fmlookup A x11)) B) in exI)
apply (auto simp: polys-rel-update-remove PAC-Format-add-and-remove
is-failed-def is-success-def is-found-def)

```

```

    dest!: eq-successI
    split: if-splits
    dest: rtrancp-PAC-Format-subset-ideal
    intro: PAC-Format-add-and-remove H3)
  done
subgoal
  apply (rule-tac x = ⟨(a, V, B)⟩ in exI)
  apply (auto simp: fmdrop-irrelevant
    is-failed-def is-success-def is-found-def
    dest!: eq-successI
    split: if-splits
    dest: rtrancp-PAC-Format-subset-ideal
    intro: PAC-Format-add-and-remove)
  done
done
subgoal
  by (rule RETURN-SPEC-refine)
  (auto simp: Ex-status-iff)
done
done
qed

```

**definition** *PAC-checker*

$\langle \text{int-poly} \Rightarrow \text{fpac-step} \Rightarrow \text{status} \Rightarrow (\text{int-poly}, \text{nat}, \text{nat}) \text{ pac-step list} \Rightarrow$   
 $(\text{status} \times \text{fpac-step}) \text{ nres} \rangle$

**where**

$\langle \text{PAC-checker spec } A \text{ } b \text{ } st = \text{do} \{$   
 $(S, -) \leftarrow \text{WHILE}_T$   
 $(\lambda((b :: \text{status}, A :: \text{fpac-step}), st). \neg \text{is-failed } b \wedge st \neq [])$   
 $(\lambda((bA), st). \text{do} \{$   
 $\text{ASSERT}(st \neq []);$   
 $S \leftarrow \text{PAC-checker-step spec } (bA) (\text{hd } st);$   
 $\text{RETURN } (S, \text{tl } st)$   
 $\})$   
 $((b, A), st);$   
 $\text{RETURN } S$   
 $\} \rangle$

**lemma** *PAC-checker-specification-spec-trans:*

$\langle \text{PAC-checker-specification-spec spec } A (st, x2) \Longrightarrow$   
 $\text{PAC-checker-specification-step-spec } A \text{ spec } x2 (st', x1a) \Longrightarrow$   
 $\text{PAC-checker-specification-spec spec } A (st', x1a) \rangle$

**unfolding** *PAC-checker-specification-spec-def*

*PAC-checker-specification-step-spec-def*

**apply** *auto*

**using** *is-failed-alt-def* **apply** *blast+*

**done**

**lemma** *RES-SPEC-eq:*

$\langle \text{RES } \Phi = \text{SPEC}(\lambda P. P \in \Phi) \rangle$

**by** *auto*

**lemma** *is-failed-is-success-completeD:*

$\langle \neg \text{is-failed } x \implies \neg \text{is-success } x \implies \text{is-found } x \rangle$   
**by** (cases  $x$ ) *auto*

**lemma** *PAC-checker-PAC-checker-specification2*:

$\langle (A, B) \in \text{polys-rel-full} \implies$   
 $\neg \text{is-failed } a \implies$   
 $(a = \text{FOUND} \implies \text{spec} \in \text{pac-ideal } (\text{set-mset } (\text{snd } B))) \implies$   
 $\bigcup (\text{vars } \text{'set-mset } (\text{ran-m } (\text{snd } A))) \subseteq \text{fst } B \implies$   
 $\text{vars spec} \subseteq \text{fst } B \implies$   
 $\text{PAC-checker spec } A \text{ a st} \leq \Downarrow (\text{status-rel} \times_r \text{polys-rel-full}) (\text{PAC-checker-specification2 spec } B) \rangle$   
**unfolding** *PAC-checker-def conc-fun-RES*  
**apply** (*subst RES-SPEC-eq*)  
**apply** (*refine-vcg WHILET-rule* [where  
 $I = \langle \lambda((bB), st). bB \in (\text{status-rel} \times_r \text{polys-rel-full})^{-1} \text{' '}$   
 $\text{Collect } (\text{PAC-checker-specification-spec spec } B) \rangle$   
**and**  $R = \langle \text{measure } (\lambda(-, st). \text{Suc } (\text{length } st)) \rangle$ ]  
**subgoal by** *auto*  
**subgoal apply** (*auto simp: PAC-checker-specification-spec-def*)  
**apply** (*cases B; cases A*)  
**apply** (*auto simp: polys-rel-def polys-rel-full-def Image-iff*)  
**done**  
**subgoal by** *auto*  
**subgoal**  
**apply** *auto*  
**apply** (*rule*  
 $\text{PAC-checker-step-PAC-checker-specification2} [\text{of } \text{' - - - - - } \langle \text{fst } B \rangle, \text{ THEN } \text{order-trans}]$ )  
**apply** *assumption*  
**apply** *assumption*  
**apply** (*auto intro: PAC-checker-specification-spec-trans simp: conc-fun-RES*)  
**apply** (*auto simp: PAC-checker-specification-spec-def polys-rel-full-def polys-rel-def*  
 $\text{dest: PAC-Format-subset-ideal}$   
 $\text{dest: is-failed-is-success-completeD; fail})+$   
**apply** (*auto simp: Image-iff intro: PAC-checker-specification-spec-trans*)  
**by** (*metis (mono-tags, lifting) PAC-checker-specification-spec-trans mem-Collect-eq old.prod.case*  
 $\text{polys-rel-def polys-rel-full-def prod.collapse}$ )  
**subgoal**  
**by** *auto*  
**done**

**definition** *remap-polys-polynom-bool* ::  $\langle \text{int mpolys} \Rightarrow \text{nat set} \Rightarrow (\text{nat}, \text{int-poly}) \text{ fmap} \Rightarrow (\text{status} \times \text{fpac-step}) \text{ nres} \rangle$  **where**

$\langle \text{remap-polys-polynom-bool spec} = (\lambda \mathcal{V} A.$   
 $\text{SPEC}(\lambda(st, \mathcal{V}', A'). (\neg \text{is-failed } st \longrightarrow$   
 $\text{dom-m } A = \text{dom-m } A' \wedge$   
 $(\forall i \in \# \text{dom-m } A. \text{the } (\text{fmlookup } A \ i) - \text{the } (\text{fmlookup } A' \ i) \in \text{ideal polynom-bool}) \wedge$   
 $\bigcup (\text{vars } \text{'set-mset } (\text{ran-m } A)) \subseteq \mathcal{V}' \wedge$   
 $\bigcup (\text{vars } \text{'set-mset } (\text{ran-m } A')) \subseteq \mathcal{V}') \wedge$   
 $(st = \text{FOUND} \longrightarrow \text{spec} \in \# \text{ran-m } A')) \rangle$

**definition** *remap-polys-change-all* ::  $\langle \text{int mpolys} \Rightarrow \text{nat set} \Rightarrow (\text{nat}, \text{int-poly}) \text{ fmap} \Rightarrow (\text{status} \times \text{fpac-step}) \text{ nres} \rangle$  **where**

$\langle \text{remap-polys-change-all spec} = (\lambda \mathcal{V} A. \text{SPEC } (\lambda(st, \mathcal{V}', A').$   
 $(\neg \text{is-failed } st \longrightarrow$   
 $\text{pac-ideal } (\text{set-mset } (\text{ran-m } A)) = \text{pac-ideal } (\text{set-mset } (\text{ran-m } A')) \wedge$   
 $\bigcup (\text{vars } \text{'set-mset } (\text{ran-m } A)) \subseteq \mathcal{V}' \wedge$

$$\bigcup (vars \text{ ' set-mset (ran-m } A') \subseteq \mathcal{V}' ) \wedge \\ (st = FOUND \longrightarrow spec \in \# \text{ ran-m } A')) \rangle$$

**lemma** *fmap-eq-dom-iff*:

$\langle A = A' \longleftrightarrow dom-m \ A = dom-m \ A' \wedge (\forall i \in \# \text{ dom-m } A. \text{ the (fmllookup } A \ i) = \text{ the (fmllookup } A' \ i)) \rangle$   
**apply** *auto*  
**by** (*metis fmap-ext in-dom-m-lookup-iff option.collapse*)

**lemma** *ideal-remap-incl*:

$\langle \text{finite } A' \implies (\forall a' \in A'. \exists a \in A. a - a' \in B) \implies ideal \ (A' \cup B) \subseteq ideal \ (A \cup B) \rangle$   
**apply** (*induction A' rule: finite-induct*)  
**apply** (*auto intro: ideal.span-mono*)  
**using** *ideal.span-mono sup-ge2* **apply** *blast*  
**proof** –  
**fix** *x :: 'a and F :: 'a set and xa :: 'a and a :: 'a*  
**assume** *a1: a ∈ A*  
**assume** *a2: a – x ∈ B*  
**assume** *a3: xa ∈ More-Modules.ideal (insert x (F ∪ B))*  
**assume** *a4: More-Modules.ideal (F ∪ B) ⊆ More-Modules.ideal (A ∪ B)*  
**have** *x ∈ More-Modules.ideal (A ∪ B)*  
**using** *a2 a1 by (metis (no-types, lifting) Un-upper1 Un-upper2 add-diff-cancel-left' diff-add-cancel ideal.module-axioms ideal.span-diff in-mono module.span-superset)*  
**then show** *xa ∈ More-Modules.ideal (A ∪ B)*  
**using** *a4 a3 ideal.span-insert-subset by blast*  
**qed**

**lemma** *pac-ideal-remap-eq*:

$\langle dom-m \ b = dom-m \ ba \implies$   
 $\forall i \in \# dom-m \ ba.$   
 $\text{the (fmllookup } b \ i) - \text{the (fmllookup } ba \ i)$   
 $\in More-Modules.ideal \ polynom\text{-bool} \implies$   
 $pac\text{-ideal} \ ((\lambda x. \text{the (fmllookup } b \ x)) \text{ ' set-mset (dom-m } ba)) = pac\text{-ideal} \ ((\lambda x. \text{the (fmllookup } ba \ x)) \text{ ' set-mset (dom-m } ba)) \rangle$   
**unfolding** *pac-ideal-alt-def*  
**apply** *standard*  
**subgoal**  
**apply** (*rule ideal-remap-incl*)  
**apply** (*auto dest!: multi-member-split*  
*dest: ideal.span-neg*)  
**apply** (*drule ideal.span-neg*)  
**apply** *auto*  
**done**  
**subgoal**  
**by** (*rule ideal-remap-incl*)  
*(auto dest!: multi-member-split)*  
**done**

**lemma** *remap-polys-polynom-bool-remap-polys-change-all*:

$\langle \text{remap-polys-polynom-bool spec } \mathcal{V} \ A \leq \text{remap-polys-change-all spec } \mathcal{V} \ A \rangle$   
**unfolding** *remap-polys-polynom-bool-def remap-polys-change-all-def*  
**apply** (*simp add: ideal.span-zero fmap-eq-dom-iff ideal.span-eq*)  
**apply** (*auto dest: multi-member-split simp: ran-m-def ideal.span-base pac-ideal-remap-eq add-mset-eq-add-mset eq-commute[of 'add-mset - -> 'dom-m (A :: (nat, int mpoly)fmap)] for A*)  
**done**

**definition** *remap-polys* ::  $\langle \text{int mpolys} \Rightarrow \text{nat set} \Rightarrow (\text{nat}, \text{int-poly}) \text{ fmap} \Rightarrow (\text{status} \times \text{fpac-step}) \text{ nres} \rangle$   
**where**

```

  remap-polys spec = ( $\lambda \mathcal{V} A. \text{do}\{$ 
     $\text{dom} \leftarrow \text{SPEC}(\lambda \text{dom}. \text{set-mset} (\text{dom-m } A) \subseteq \text{dom} \wedge \text{finite dom});$ 

     $\text{failed} \leftarrow \text{SPEC}(\lambda :: \text{bool}. \text{True});$ 
     $\text{if failed}$ 
     $\text{then do}\{$ 
       $\text{RETURN} (\text{FAILED}, \mathcal{V}, \text{fmempty})$ 
     $\}$ 
     $\text{else do}\{$ 
       $(b, N) \leftarrow \text{FOREACH dom}$ 
       $(\lambda i (b, \mathcal{V}, A').$ 
         $\text{if } i \in \# \text{ dom-m } A$ 
         $\text{then do}\{$ 
           $p \leftarrow \text{SPEC}(\lambda p. \text{the } (\text{fmlookup } A \ i) - p \in \text{ideal polynom-bool} \wedge \text{vars } p \subseteq \text{vars } (\text{the } (\text{fmlookup } A \ i))));$ 
           $\text{eq} \leftarrow \text{SPEC}(\lambda \text{eq}. \text{eq} \longrightarrow p = \text{spec});$ 
           $\mathcal{V} \leftarrow \text{SPEC}(\lambda \mathcal{V}'. \mathcal{V} \cup \text{vars } (\text{the } (\text{fmlookup } A \ i)) \subseteq \mathcal{V}');$ 
           $\text{RETURN}(b \vee \text{eq}, \mathcal{V}, \text{fmupd } i \ p \ A')$ 
         $\}$ 
         $\text{else RETURN } (b, \mathcal{V}, A')$ 
       $(\text{False}, \mathcal{V}, \text{fmempty});$ 
       $\text{RETURN } (\text{if } b \text{ then FOUND else SUCCESS}, N)$ 
     $\}$ 
   $\})$ 

```

**lemma** *remap-polys-spec*:

```

  remap-polys spec  $\mathcal{V} A \leq \text{remap-polys-polynom-bool spec } \mathcal{V} A$ 
unfolding remap-polys-def remap-polys-polynom-bool-def
apply (refine-vcg FOREACH-rule[where
   $I = (\lambda \text{dom } (b, \mathcal{V}, A').$ 
     $\text{set-mset} (\text{dom-m } A') = \text{set-mset} (\text{dom-m } A) - \text{dom} \wedge$ 
     $(\forall i \in \text{set-mset} (\text{dom-m } A) - \text{dom}. \text{the } (\text{fmlookup } A \ i) - \text{the } (\text{fmlookup } A' \ i) \in \text{ideal polynom-bool})$ 
   $\wedge$ 
     $\bigcup (\text{vars } ' \text{set-mset } (\text{ran-m } (\text{fmrestrict-set } (\text{set-mset } (\text{dom-m } A')) \ A))) \subseteq \mathcal{V} \wedge$ 
     $\bigcup (\text{vars } ' \text{set-mset } (\text{ran-m } A')) \subseteq \mathcal{V} \wedge$ 
     $(b \longrightarrow \text{spec} \in \# \text{ ran-m } A'))]$ 
subgoal by auto
subgoal by auto
subgoal by auto
subgoal by auto
subgoal by auto
subgoal by auto
subgoal by auto
subgoal
  by auto
subgoal by auto
subgoal using ideal.span-add by auto
subgoal by auto
subgoal by auto
subgoal by clarsimp auto
subgoal
  supply[[goals-limit=1]]

```

```

  by (auto simp add: ran-m-mapsto-upd-notin dom-m-fmrestrict-set' subset-eq)
subgoal
  supply[[goals-limit=1]]
  by (auto simp add: ran-m-mapsto-upd-notin dom-m-fmrestrict-set' subset-eq)
subgoal
  by (auto simp: ran-m-mapsto-upd-notin)
subgoal
  by auto
subgoal
  by auto
subgoal
  by (auto simp add: ran-m-mapsto-upd-notin dom-m-fmrestrict-set' subset-eq)
subgoal
  by (auto simp add: ran-m-mapsto-upd-notin dom-m-fmrestrict-set' subset-eq)
subgoal
  by auto
subgoal
  by (auto simp: distinct-set-mset-eq-iff[symmetric] distinct-mset-dom)
subgoal
  by (auto simp: distinct-set-mset-eq-iff[symmetric] distinct-mset-dom)
subgoal
  by (auto simp add: ran-m-mapsto-upd-notin dom-m-fmrestrict-set' subset-eq
    fmlookup-restrict-set-id')
subgoal
  by (auto simp add: ran-m-mapsto-upd-notin dom-m-fmrestrict-set' subset-eq)
subgoal
  by (auto simp add: ran-m-mapsto-upd-notin dom-m-fmrestrict-set' subset-eq
    fmlookup-restrict-set-id')
done

```

### 6.3 Full Checker

**definition** *full-checker*

```

:: (int-poly  $\Rightarrow$  (nat, int-poly) fmap  $\Rightarrow$  (int-poly, nat,nat) pac-step list  $\Rightarrow$  (status  $\times$  -) nres)
where
  (full-checker spec0 A pac = do {
    spec  $\leftarrow$  normalize-poly-spec spec0;
    (st,  $\mathcal{V}$ , A)  $\leftarrow$  remap-polys-change-all spec {} A;
    if is-failed st then
      RETURN (st,  $\mathcal{V}$ , A)
    else do {
       $\mathcal{V} \leftarrow$  SPEC( $\lambda \mathcal{V}'$ .  $\mathcal{V} \cup \text{vars spec0} \subseteq \mathcal{V}'$ );
      PAC-checker spec ( $\mathcal{V}$ , A) st pac
    }
  })

```

**lemma** *restricted-ideal-to-mono*:

```

restricted-ideal-toI  $\mathcal{V}$  I  $\subseteq$  restricted-ideal-toI  $\mathcal{V}'$  J  $\implies$ 
 $\mathcal{U} \subseteq \mathcal{V} \implies$ 
  restricted-ideal-toI  $\mathcal{U}$  I  $\subseteq$  restricted-ideal-toI  $\mathcal{U}$  J
by (auto simp: restricted-ideal-to-def)

```

**lemma** *full-checker-spec*:

```

assumes (A, A')  $\in$  polys-rel
shows
  (full-checker spec A pac  $\leq$   $\Downarrow \{((st, G), (st', G')). (st, st') \in \text{status-rel} \wedge$ 

```

```

    (st ≠ FAILED → (G, G') ∈ polys-rel-full)}
    (PAC-checker-specification spec (A'))
proof -
  have H: (set-mset b ⊆ pac-ideal (set-mset (ran-m A)) ⇒
    x ∈ pac-ideal (set-mset b) ⇒ x ∈ pac-ideal (set-mset A')) for b x
  using assms apply (auto simp: polys-rel-def)
  by (metis (no-types, lifting) ideal.span-subset-spanI ideal.span-superset le-sup-iff pac-ideal-alt-def subsetD)
  have 1: (x ∈ {(st, V', A')}.
    (¬ is-failed st → pac-ideal (set-mset (ran-m x2)) =
      pac-ideal (set-mset (ran-m A')) ∧
      ⋃ (vars ' set-mset (ran-m ABC)) ⊆ V' ∧
      ⋃ (vars ' set-mset (ran-m A')) ⊆ V') ∧
    (st = FOUND → specA ∈# ran-m A')) ⇒
    x = (st, x') ⇒ x' = (V, Aa) ⇒ ((V', Aa), V', ran-m Aa) ∈ polys-rel-full) for Aa specA x2 st x
  V' V x' ABC
  by (auto simp: polys-rel-def polys-rel-full-def)
show ?thesis
supply[[goals-limit=1]]
unfolding full-checker-def normalize-poly-spec-def
  PAC-checker-specification-def remap-polys-change-all-def
apply (refine-vcg PAC-checker-PAC-checker-specification2[THEN order-trans, of - -]
  lhs-step-If)
subgoal by (auto simp: is-failed-def RETURN-RES-refine-iff)
apply (rule 1; assumption)
subgoal
  using fmap-ext assms by (auto simp: polys-rel-def ran-m-def)
subgoal
  by auto
subgoal
  by auto
subgoal for specA x1 x2 x x1a x2a x1b
  apply (rule ref-two-step[OF conc-fun-R-mono])
  apply auto[]
  using assms
  apply (auto simp add: PAC-checker-specification-spec-def conc-fun-RES polys-rel-def polys-rel-full-def
    dest!: rtranclp-PAC-Format-subset-ideal dest: is-failed-is-success-completeD)
  apply (drule restricted-ideal-to-mono[of - - - ⋃ (vars ' set-mset (ran-m A)) ∪ vars spec])
  apply auto[]
  apply auto[]
  apply (metis (no-types, lifting) group-eq-aux ideal.span-add ideal.span-base in-mono pac-ideal-alt-def
    sup.cobounded2)
  apply (smt le-sup-iff restricted-ideal-to-mono subsetD subset-trans sup-ge1 sup-ge2)
  apply (metis (no-types, lifting) cancel-comm-monoid-add-class.diff-cancel diff-add-eq
    diff-in-polynom-bool-pac-idealI group-eq-aux ideal.span-add-eq2)
  apply (drule restricted-ideal-to-mono[of - - - ⋃ (vars ' set-mset (ran-m A)) ∪ vars spec])
  apply auto[]
  apply auto[]
  apply (smt le-sup-iff restricted-ideal-to-mono subsetD subset-trans sup-ge1 sup-ge2)
  apply (drule restricted-ideal-to-mono[of - - - ⋃ (vars ' set-mset (ran-m A)) ∪ vars spec])
  apply auto[]
  apply auto[]
  apply (metis (no-types, lifting) group-eq-aux ideal.span-add ideal.span-base in-mono pac-ideal-alt-def
    sup.cobounded2)
  apply (smt le-sup-iff restricted-ideal-to-mono subsetD subset-trans sup-ge1 sup-ge2)

```

```

    apply (metis (no-types, lifting) group-eq-aux ideal.span-add ideal.span-base in-mono pac-ideal-alt-def
sup.cobounded2)
  done
done
qed

```

```

lemma full-checker-spec':
  shows
    ((uncurry2 full-checker, uncurry2 (λspec A -. PAC-checker-specification spec A)) ∈
      (Id ×r polys-rel) ×r Id →f ⟨{((st, G), (st', G')). (st, st') ∈ status-rel ∧
      (st ≠ FAILED → (G, G') ∈ polys-rel-full)}⟩ nres-rel)
  using full-checker-spec
  by (auto intro!: frefI nres-relI)

```

```

end
theory PAC-Polynoms
  imports PAC-Specification Finite-Map-Multiset
begin

```

## 7 Polynomials of strings

Isabelle's definition of polynoms only work with variables of type *nat*. Therefore, we introduce a version that uses strings.

### 7.1 Polynomials and Variables

```

lemma poly-embed-EX:
  ⟨∃ φ. bij (φ :: string ⇒ nat)⟩
  by (rule countableE-infinite[of ⟨UNIV :: string set⟩])
  (auto intro!: infinite-UNIV-listI)

```

Using a multiset instead of a list has some advantage from an abstract point of view. First, we can have monomials that appear several times and the coefficient can also be zero. Basically, we can represent un-normalised polynomials, which is very useful to talk about intermediate states in our program.

```

type-synonym term-poly = ⟨string multiset⟩
type-synonym mset-polynom =
  ⟨(term-poly * int) multiset⟩

```

```

definition normalized-poly :: ⟨mset-polynom ⇒ bool⟩ where
  ⟨normalized-poly p ⟷
    distinct-mset (fst '# p) ∧
    0 ∉# snd '# p)

```

```

lemma normalized-poly-simps[simp]:
  ⟨normalized-poly {#}⟩
  ⟨normalized-poly (add-mset t p) ⟷ snd t ≠ 0 ∧
    fst t ∉# fst '# p ∧ normalized-poly p⟩
  by (auto simp: normalized-poly-def)

```

```

lemma normalized-poly-mono:
  ⟨normalized-poly B ⟹ A ⊆# B ⟹ normalized-poly A⟩
  unfolding normalized-poly-def

```



by (auto intro: distinct-mset-mono image-mset-subseteq-mono)

**definition** *mult-poly-by-monom* ::  $\langle \text{term-poly} * \text{int} \Rightarrow \text{mset-polynom} \Rightarrow \text{mset-polynom} \rangle$  **where**  
 $\langle \text{mult-poly-by-monom} = (\lambda y s q. \text{image-mset } (\lambda x s. (\text{fst } x s + \text{fst } y s, \text{snd } y s * \text{snd } x s)) q) \rangle$

**definition** *mult-poly-raw* ::  $\langle \text{mset-polynom} \Rightarrow \text{mset-polynom} \Rightarrow \text{mset-polynom} \rangle$  **where**  
 $\langle \text{mult-poly-raw } p \ q =$   
 $(\text{sum-mset } ((\lambda y. \text{mult-poly-by-monom } y \ q) \text{ '\# } p)) \rangle$

**definition** *remove-powers* ::  $\langle \text{mset-polynom} \Rightarrow \text{mset-polynom} \rangle$  **where**  
 $\langle \text{remove-powers } x s = \text{image-mset } (\text{apfst remdups-mset}) \ x s \rangle$

**definition** *all-vars-mset* ::  $\langle \text{mset-polynom} \Rightarrow \text{string multiset} \rangle$  **where**  
 $\langle \text{all-vars-mset } p = \bigcup \# (\text{fst } \text{'\# } p) \rangle$

**abbreviation** *all-vars* ::  $\langle \text{mset-polynom} \Rightarrow \text{string set} \rangle$  **where**  
 $\langle \text{all-vars } p \equiv \text{set-mset } (\text{all-vars-mset } p) \rangle$

**definition** *add-to-coefficient* ::  $\langle - \Rightarrow \text{mset-polynom} \Rightarrow \text{mset-polynom} \rangle$  **where**  
 $\langle \text{add-to-coefficient} = (\lambda (a, n) \ b. \{ \#(a', -) \in \# \ b. \ a' \neq a \# \} +$   
 $(\text{if } n + \text{sum-mset } (\text{snd } \text{'\# } \{ \#(a', -) \in \# \ b. \ a' = a \# \}) = 0 \text{ then } \{ \# \}$   
 $\text{else } \{ \#(a, n + \text{sum-mset } (\text{snd } \text{'\# } \{ \#(a', -) \in \# \ b. \ a' = a \# \})) \# \}) \rangle$

**definition** *normalize-poly* ::  $\langle \text{mset-polynom} \Rightarrow \text{mset-polynom} \rangle$  **where**  
 $\langle \text{normalize-poly } p = \text{fold-mset } \text{add-to-coefficient } \{ \# \} \ p \rangle$

**lemma** *add-to-coefficient-simps*:

$\langle n + \text{sum-mset } (\text{snd } \text{'\# } \{ \#(a', -) \in \# \ b. \ a' = a \# \}) \neq 0 \implies$   
 $\text{add-to-coefficient } (a, n) \ b = \{ \#(a', -) \in \# \ b. \ a' \neq a \# \} +$   
 $\{ \#(a, n + \text{sum-mset } (\text{snd } \text{'\# } \{ \#(a', -) \in \# \ b. \ a' = a \# \})) \# \} \rangle$   
 $\langle n + \text{sum-mset } (\text{snd } \text{'\# } \{ \#(a', -) \in \# \ b. \ a' = a \# \}) = 0 \implies$   
 $\text{add-to-coefficient } (a, n) \ b = \{ \#(a', -) \in \# \ b. \ a' \neq a \# \} \rangle$  **and**  
*add-to-coefficient-simps-If*:  
 $\langle \text{add-to-coefficient } (a, n) \ b = \{ \#(a', -) \in \# \ b. \ a' \neq a \# \} +$   
 $(\text{if } n + \text{sum-mset } (\text{snd } \text{'\# } \{ \#(a', -) \in \# \ b. \ a' = a \# \}) = 0 \text{ then } \{ \# \}$   
 $\text{else } \{ \#(a, n + \text{sum-mset } (\text{snd } \text{'\# } \{ \#(a', -) \in \# \ b. \ a' = a \# \})) \# \}) \rangle$

by (auto simp: add-to-coefficient-def)

**interpretation** *comp-fun-commute*  $\langle \text{add-to-coefficient} \rangle$

**proof** –

**have** [simp]:

$\langle a \neq aa \implies$   
 $((\text{case } x \text{ of } (a', -) \Rightarrow a' \neq aa) \wedge (\text{case } x \text{ of } (a', -) \Rightarrow a' = a)) \longleftrightarrow$   
 $(\text{case } x \text{ of } (a', -) \Rightarrow a' = a) \rangle$  **for**  $a' \ aa \ a \ x$

**by** auto

**show**  $\langle \text{comp-fun-commute } \text{add-to-coefficient} \rangle$

**unfolding** *add-to-coefficient-def*

**by** standard

(auto intro!: ext simp: filter-filter-mset ac-simps add-eq-0-iff

intro: filter-mset-cong)

qed

```

lemma normalized-poly-normalize-poly[simp]:
  ⟨normalized-poly (normalize-poly p)⟩
  unfolding normalize-poly-def
  apply (induction p)
  subgoal by auto
  subgoal for x p
    by (cases x)
      (auto simp: add-to-coefficient-simps-If
        intro: normalized-poly-mono)
  done

```

## 7.2 Addition

```

inductive add-poly-p :: ⟨mset-polynom × mset-polynom × mset-polynom ⇒ mset-polynom × mset-polynom
  × mset-polynom ⇒ bool⟩ where
  add-new-coeff-r:
    ⟨add-poly-p (p, add-mset x q, r) (p, q, add-mset x r)⟩ |
  add-new-coeff-l:
    ⟨add-poly-p (add-mset x p, q, r) (p, q, add-mset x r)⟩ |
  add-same-coeff-l:
    ⟨add-poly-p (add-mset (x, n) p, q, add-mset (x, m) r) (p, q, add-mset (x, n + m) r)⟩ |
  add-same-coeff-r:
    ⟨add-poly-p (p, add-mset (x, n) q, add-mset (x, m) r) (p, q, add-mset (x, n + m) r)⟩ |
  rem-0-coeff:
    ⟨add-poly-p (p, q, add-mset (x, 0) r) (p, q, r)⟩

```

**inductive-cases** *add-poly-pE*: ⟨*add-poly-p S T*⟩

**lemmas** *add-poly-p-induct* =  
*add-poly-p.induct*[split-format(complete)]

```

lemma add-poly-p-empty-l:
  ⟨add-poly-p** (p, q, r) ({#}, q, p + r)⟩
  apply (induction p arbitrary: r)
  subgoal by auto
  subgoal
    by (metis (no-types, lifting) add-new-coeff-l r-into-rtranclp
      rtranclp-trans union-mset-add-mset-left union-mset-add-mset-right)
  done

```

```

lemma add-poly-p-empty-r:
  ⟨add-poly-p** (p, q, r) (p, {#}, q + r)⟩
  apply (induction q arbitrary: r)
  subgoal by auto
  subgoal
    by (metis (no-types, lifting) add-new-coeff-r r-into-rtranclp
      rtranclp-trans union-mset-add-mset-left union-mset-add-mset-right)
  done

```

```

lemma add-poly-p-sym:
  ⟨add-poly-p (p, q, r) (p', q', r') ⟷ add-poly-p (q, p, r) (q', p', r')⟩
  apply (rule iffI)
  subgoal
    by (cases rule: add-poly-p.cases, assumption)
      (auto intro: add-poly-p.intros)
  subgoal

```

```

  by (cases rule: add-poly-p.cases, assumption)
    (auto intro: add-poly-p.intros)
done

```

**lemma** *wf-if-measure-in-wf*:

```

⟨wf R ⟹ (⋀ a b. (a, b) ∈ S ⟹ (ν a, ν b) ∈ R) ⟹ wf S⟩
by (metis in-inv-image wfE-min wfI-min wf-inv-image)

```

**lemma** *lexn-n*:

```

⟨n > 0 ⟹ (x # xs, y # ys) ∈ lexn r n ⟷
(length xs = n-1 ∧ length ys = n-1) ∧ ((x, y) ∈ r ∨ (x = y ∧ (xs, ys) ∈ lexn r (n-1)))⟩
apply (cases n)
apply simp
by (auto simp: map-prod-def image-iff lex-prod-def)

```

**lemma** *wf-add-poly-p*:

```

⟨wf {(x, y). add-poly-p y x}⟩
by (rule wf-if-measure-in-wf[where R = ⟨lexn less-than 3⟩ and
  ν = ⟨λ(a,b,c). [size a, size b, size c]⟩])
(auto simp: add-poly-p.simps wf-lexn
  simp: lexn-n simp del: lexn.simps(2))

```

**lemma** *mult-poly-by-monom-simps[simp]*:

```

⟨mult-poly-by-monom t {#} = {#}⟩
⟨mult-poly-by-monom t (ps + qs) = mult-poly-by-monom t ps + mult-poly-by-monom t qs⟩
⟨mult-poly-by-monom a (add-mset p ps) = add-mset (fst a + fst p, snd a * snd p) (mult-poly-by-monom
a ps)⟩
proof -
  interpret comp-fun-commute ⟨(λxs. add-mset (xs + t))⟩ for t
  by standard auto
  show
    ⟨mult-poly-by-monom t (ps + qs) = mult-poly-by-monom t ps + mult-poly-by-monom t qs⟩ for t
  by (induction ps)
    (auto simp: mult-poly-by-monom-def)
  show
    ⟨mult-poly-by-monom a (add-mset p ps) = add-mset (fst a + fst p, snd a * snd p) (mult-poly-by-monom
a ps)⟩
  by (induction t {#} = {#}) for t
  by (auto simp: mult-poly-by-monom-def)
qed

```

**inductive** *mult-poly-p* :: ⟨mset-polynom ⇒ mset-polynom × mset-polynom ⇒ mset-polynom × mset-polynom  
⇒ bool⟩

**for** *q* :: mset-polynom **where**

*mult-step*:

```

⟨mult-poly-p q (add-mset (xs, n) p, r) (p, (λ(ys, m). (remdups-mset (xs + ys), n * m))) ' # q + r)⟩

```

**lemmas** *mult-poly-p-induct* = *mult-poly-p.induct[split-format(complete)]*

### 7.3 Normalisation

**inductive** *normalize-poly-p* :: ⟨mset-polynom ⇒ mset-polynom ⇒ bool⟩ **where**

*rem-0-coeff*[*simp*, *intro*]:

```

⟨normalize-poly-p p q ⟹ normalize-poly-p (add-mset (xs, 0) p) q |
merge-dup-coeff[simp, intro]:

```

```

  ⟨normalize-poly-p p q ⇒ normalize-poly-p (add-mset (xs, m) (add-mset (xs, n) p)) (add-mset (xs,
m + n) q)⟩ |
same[simp, intro]:
  ⟨normalize-poly-p p p⟩ |
keep-coeff[simp, intro]:
  ⟨normalize-poly-p p q ⇒ normalize-poly-p (add-mset x p) (add-mset x q)⟩

```

## 7.4 Correctness

This locale maps string polynoms to real polynoms.

```

locale poly-embed =
  fixes  $\varphi :: \langle \text{string} \Rightarrow \text{nat} \rangle$ 
  assumes  $\varphi\text{-inj} :: \langle \text{inj } \varphi \rangle$ 
begin

```

```

definition poly-of-vars :: term-poly ⇒ ('a :: {comm-semiring-1}) mpoly where
  ⟨poly-of-vars xs = fold-mset (λa b. Var (φ a) * b) (1 :: 'a mpoly) xs⟩

```

```

lemma poly-of-vars-simps[simp]:

```

**shows**

```

  ⟨poly-of-vars (add-mset x xs) = Var (φ x) * (poly-of-vars xs :: ('a :: {comm-semiring-1}) mpoly)⟩ (is
?A) and

```

```

  ⟨poly-of-vars (xs + ys) = poly-of-vars xs * (poly-of-vars ys :: ('a :: {comm-semiring-1}) mpoly)⟩ (is
?B)

```

**proof** –

```

  interpret comp-fun-commute ⟨(λa b. (b :: 'a :: {comm-semiring-1}) mpoly) * Var (φ a)⟩

```

**by** standard

```

  (auto simp: algebra-simps ac-simps

```

```

    Var-def times-monomial-monomial intro!: ext)

```

**show** ?A

```

  by (auto simp: poly-of-vars-def comp-fun-commute-axioms fold-mset-fusion
    ac-simps)

```

**show** ?B

```

  apply (auto simp: poly-of-vars-def ac-simps)

```

```

  by (simp add: local.comp-fun-commute-axioms local.fold-mset-fusion
    semiring-normalization-rules(18))

```

**qed**

```

definition mononom-of-vars where

```

```

  ⟨mononom-of-vars ≡ (λ(xs, n). (+) (Const n * poly-of-vars xs))⟩

```

```

interpretation comp-fun-commute ⟨mononom-of-vars⟩

```

**by** standard

```

  (auto simp: algebra-simps ac-simps mononom-of-vars-def

```

```

    Var-def times-monomial-monomial intro!: ext)

```

```

lemma [simp]:

```

```

  ⟨poly-of-vars {#} = 1⟩

```

```

  by (auto simp: poly-of-vars-def)

```

```

lemma mononom-of-vars-add[simp]:

```

```

  ⟨NO-MATCH 0 b ⇒ mononom-of-vars xs b = Const (snd xs) * poly-of-vars (fst xs) + b⟩

```

```

  by (cases xs)

```

(auto simp: ac-simps mononom-of-vars-def)

**definition** *polynom-of-mset* ::  $\langle \text{mset-polynom} \Rightarrow \cdot \rangle$  **where**  
 $\langle \text{polynom-of-mset } p = \text{sum-mset } (\text{mononom-of-vars } \text{'\#'} p) 0 \rangle$

**lemma** *polynom-of-mset-append*[simp]:  
 $\langle \text{polynom-of-mset } (xs + ys) = \text{polynom-of-mset } xs + \text{polynom-of-mset } ys \rangle$   
**by** (auto simp: ac-simps Const-def polynom-of-mset-def)

**lemma** *polynom-of-mset-Cons*[simp]:  
 $\langle \text{polynom-of-mset } (\text{add-mset } x \text{ } ys) = \text{Const } (\text{snd } x) * \text{poly-of-vars } (\text{fst } x) + \text{polynom-of-mset } ys \rangle$   
**by** (cases x)  
(auto simp: ac-simps polynom-of-mset-def mononom-of-vars-def)

**lemma** *polynom-of-mset-empty*[simp]:  
 $\langle \text{polynom-of-mset } \{\#\} = 0 \rangle$   
**by** (auto simp: polynom-of-mset-def)

**lemma** *polynom-of-mset-mult-poly-by-monom*[simp]:  
 $\langle \text{polynom-of-mset } (\text{mult-poly-by-monom } x \text{ } ys) =$   
 $(\text{Const } (\text{snd } x) * \text{poly-of-vars } (\text{fst } x) * \text{polynom-of-mset } ys) \rangle$   
**by** (induction ys)  
(auto simp: Const-mult algebra-simps)

**lemma** *polynom-of-mset-mult-poly-raw*[simp]:  
 $\langle \text{polynom-of-mset } (\text{mult-poly-raw } xs \text{ } ys) = \text{polynom-of-mset } xs * \text{polynom-of-mset } ys \rangle$   
**unfolding** *mult-poly-raw-def*  
**by** (induction xs arbitrary: ys)  
(auto simp: Const-mult algebra-simps)

**lemma** *polynom-of-mset-uminus*:  
 $\langle \text{polynom-of-mset } \{\#\text{case } x \text{ of } (a, b) \Rightarrow (a, - b). x \in \#\text{ } za\#\} =$   
 $- \text{polynom-of-mset } za \rangle$   
**by** (induction za)  
auto

**lemma** *X2-X-polynom-bool-mult-in*:  
 $\langle \text{Var } (x1) * (\text{Var } (x1) * p) - \text{Var } (x1) * p \in \text{More-Modules.ideal polynom-bool}$   
**using** *ideal-mult-right-in*[OF *X2-X-in-pac-ideal*[of *x1*  $\langle \{\} \rangle$ , *unfolded pac-ideal-def*], of *p*]  
**by** (auto simp: right-diff-distrib ac-simps power2-eq-square)

**lemma** *polynom-of-list-remove-powers-polynom-bool*:  
 $\langle (\text{polynom-of-mset } xs) - \text{polynom-of-mset } (\text{remove-powers } xs) \in \text{ideal polynom-bool} \rangle$

**proof** (induction xs)  
**case** *empty*  
**then show**  $\langle ?\text{case} \rangle$  **by** (auto simp: remove-powers-def ideal.span-zero)

**next**

**case** (add x xs)  
**have** *H1*:  $\langle x1 \in \#\text{ } x2 \Rightarrow$   
 $\text{Var } (\varphi \text{ } x1) * \text{poly-of-vars } x2 - p \in \text{More-Modules.ideal polynom-bool} \longleftrightarrow$   
 $\text{poly-of-vars } x2 - p \in \text{More-Modules.ideal polynom-bool}$   
 $\rangle$  **for** *x1 x2 p*  
**apply** (subst (2) ideal.span-add-eq[symmetric,

```

    of (Var (φ x1) * poly-of-vars x2 - poly-of-vars x2)])
  apply (drule multi-member-split)
  apply (auto simp: X2-X-polynom-bool-mult-in)
done

have diff: (poly-of-vars (x) - poly-of-vars (remdups-mset (x)) ∈ ideal polynom-bool) for x
  apply (induction x)
  apply (auto simp: remove-powers-def ideal.span-zero H1)
  apply (metis ideal.span-scale right-diff-distrib)
  done
show ?case
  using add
  apply (cases x)
  subgoal for ys y
    using ideal-mult-right-in2[OF diff, of (poly-of-vars ys) ys]
    apply (auto simp: remove-powers-def right-diff-distrib
      ideal.span-diff ideal.span-add field-simps)
    by (metis add-diff-add diff ideal.scale-right-diff-distrib ideal.span-add ideal.span-scale)
  done
qed

lemma add-poly-p-polynom-of-mset:
  (add-poly-p (p, q, r) (p', q', r') ⇒
    polynom-of-mset r + (polynom-of-mset p + polynom-of-mset q) =
    polynom-of-mset r' + (polynom-of-mset p' + polynom-of-mset q'))
  apply (induction rule: add-poly-p-induct)
  subgoal
    by auto
  subgoal
    by auto
  subgoal
    by (auto simp: algebra-simps Const-add)
  subgoal
    by (auto simp: algebra-simps Const-add)
  subgoal
    by (auto simp: algebra-simps Const-add)
  done

lemma rtrancp-add-poly-p-polynom-of-mset:
  (add-poly-p** (p, q, r) (p', q', r') ⇒
    polynom-of-mset r + (polynom-of-mset p + polynom-of-mset q) =
    polynom-of-mset r' + (polynom-of-mset p' + polynom-of-mset q'))
  by (induction rule: rtrancp-induct[of add-poly-p (λ(-, -, -) (λ(-, -, -), split-format(complete), of for r)]
    (auto dest: add-poly-p-polynom-of-mset))

lemma rtrancp-add-poly-p-polynom-of-mset-full:
  (add-poly-p** (p, q, {#}) ({#}, {#}, r') ⇒
    polynom-of-mset r' = (polynom-of-mset p + polynom-of-mset q))
  by (drule rtrancp-add-poly-p-polynom-of-mset)
  (auto simp: ac-simps add-eq-0-iff)

lemma poly-of-vars-remdups-mset:
  (poly-of-vars (remdups-mset (xs)) - (poly-of-vars xs)
    ∈ More-Modules.ideal polynom-bool)

```

```

apply (induction xs)
apply (auto dest!: simp: ideal.span-zero dest!: )
apply (drule multi-member-split)
apply auto
apply (drule multi-member-split)
apply (smt X2-X-polynom-bool-mult-in diff-add-cancel diff-diff-eq2 ideal.span-diff)
apply (smt X2-X-polynom-bool-mult-in diff-add-eq group-eq-aux ideal.span-add-eq)
by (metis ideal.span-scale right-diff-distrib')

```

**lemma** *polynom-of-mset-mult-map*:

```

⟨polynom-of-mset
  {#case x of (ys, n) ⇒ (remdups-mset (ys + xs), n * m). x ∈# q#} -
  Const m * (poly-of-vars xs * polynom-of-mset q)
  ∈ More-Modules.ideal polynom-bool⟩
(is ⟨?P q ∈ -⟩)

```

**proof** (induction q)

**case** *empty*

**then show** ?case **by** (auto simp: algebra-simps ideal.span-zero)

**next**

**case** (add x q)

**then have** uP: ⟨- ?P q ∈ More-Modules.ideal polynom-bool⟩

**using** ideal.span-neg **by** blast

**show** ?case

**apply** (subst ideal.span-add-eq2[symmetric, OF uP])

**apply** (cases x)

**apply** (auto simp: field-simps Const-mult)

**by** (metis ideal.span-scale poly-of-vars-remdups-mset  
poly-of-vars-simps(2) right-diff-distrib')

**qed**

**lemma** *mult-poly-p-mult-ideal*:

```

⟨mult-poly-p q (p, r) (p', r') ⇒

```

```

  (polynom-of-mset p' * polynom-of-mset q + polynom-of-mset r') - (polynom-of-mset p * polynom-of-mset
  q + polynom-of-mset r)
  ∈ ideal polynom-bool

```

**proof** (induction rule: mult-poly-p-induct)

**case** (mult-step xs n p r)

**show** ?case

**by** (auto simp: algebra-simps polynom-of-mset-mult-map)

**qed**

**lemma** *rtrancp-mult-poly-p-mult-ideal*:

```

⟨(mult-poly-p q)** (p, r) (p', r') ⇒

```

```

  (polynom-of-mset p' * polynom-of-mset q + polynom-of-mset r') - (polynom-of-mset p * polynom-of-mset
  q + polynom-of-mset r)
  ∈ ideal polynom-bool

```

**apply** (induction p' r' rule: rtrancp-induct[of ⟨mult-poly-p q⟩ ⟨(p, r)⟩ ⟨(p', q')⟩ **for** p' q', split-format(complete)])

**subgoal**

**by** (auto simp: ideal.span-zero)

**subgoal for** a b aa ba

**apply** (drule mult-poly-p-mult-ideal)

**apply** (drule ideal.span-add)

**apply** assumption

**apply** (auto simp: group-add-class.diff-add-eq-diff-diff-swap  
add.assoc add.inverse-distrib-swap ac-simps)

```

    simp flip: ab-group-add-class.ab-diff-conv-add-uminus)
  by (metis (no-types, hide-lams) ab-group-add-class.ab-diff-conv-add-uminus
    ab-semigroup-add-class.add commute add.assoc add.inverse-distrib-swap)
done

lemma rtrancp-mult-poly-p-mult-ideal-final:
  ⟨(mult-poly-p q)** (p, {#}) ({#}, r) ⟹
    (polynom-of-mset r) = (polynom-of-mset p * polynom-of-mset q)
    ∈ ideal polynom-bool
  by (drule rtrancp-mult-poly-p-mult-ideal) auto

lemma normalize-poly-p-poly-of-mset:
  ⟨normalize-poly-p p q ⟹ polynom-of-mset p = polynom-of-mset q⟩
  apply (induction rule: normalize-poly-p.induct)
  apply (auto simp: Const-add algebra-simps)
done

lemma rtrancp-normalize-poly-p-poly-of-mset:
  ⟨normalize-poly-p** p q ⟹ polynom-of-mset p = polynom-of-mset q⟩
  by (induction rule: rtrancp-induct)
  (auto simp: normalize-poly-p-poly-of-mset)

end

It would be nice to have the property in the other direction too, but this requires a deep dive
into the definitions of polynomials.

locale poly-embed-bij = poly-embed +
  fixes V N
  assumes  $\varphi$ -bij: ⟨bij-betw  $\varphi$  V N⟩
begin

definition  $\varphi'$  :: ⟨nat ⟹ string⟩ where
  ⟨ $\varphi'$  = the-inv-into V  $\varphi$ ⟩

lemma  $\varphi'$ - $\varphi$ [simp]:
  ⟨ $x \in V \implies \varphi' (\varphi x) = x$ ⟩
  using  $\varphi$ -bij unfolding  $\varphi'$ -def
  by (meson bij-betw-imp-inj-on the-inv-into-f-f)

lemma  $\varphi$ - $\varphi'$ [simp]:
  ⟨ $x \in N \implies \varphi (\varphi' x) = x$ ⟩
  using  $\varphi$ -bij unfolding  $\varphi'$ -def
  by (meson f-the-inv-into-f-bij-betw)

end

end

theory PAC-Polynoms-Term
  imports PAC-Polynoms
    Refine-Imperative-HOL.IICF
begin

```



## 8 Terms

We define some helper functions.

### 8.1 Ordering

**lemma** *fref-to-Down-curry-left*:

**fixes**  $f :: \langle 'a \Rightarrow 'b \Rightarrow 'c \text{ nres} \rangle$  **and**

$A :: \langle ('a \times 'b) \times 'd \rangle \text{ set}$

**shows**

$\langle (\text{uncurry } f, g) \in [P]_f A \rightarrow \langle B \rangle \text{nres-rel} \Rightarrow$

$(\bigwedge a \ b \ x'. P \ x' \Rightarrow ((a, b), x') \in A \Rightarrow f \ a \ b \leq \Downarrow B \ (g \ x')) \rangle$

**unfolding** *fref-def uncurry-def nres-rel-def*

**by** *auto*

**lemma** *fref-to-Down-curry-right*:

**fixes**  $g :: \langle 'a \Rightarrow 'b \Rightarrow 'c \text{ nres} \rangle$  **and**  $f :: \langle 'd \Rightarrow - \text{ nres} \rangle$  **and**

$A :: \langle 'd \times ('a \times 'b) \rangle \text{ set}$

**shows**

$\langle (f, \text{uncurry } g) \in [P]_f A \rightarrow \langle B \rangle \text{nres-rel} \Rightarrow$

$(\bigwedge a \ b \ x'. P \ (a, b) \Rightarrow (x', (a, b)) \in A \Rightarrow f \ x' \leq \Downarrow B \ (g \ a \ b)) \rangle$

**unfolding** *fref-def uncurry-def nres-rel-def*

**by** *auto*

**type-synonym** *term-poly-list* =  $\langle \text{string list} \rangle$

**type-synonym** *llist-polynom* =  $\langle (\text{term-poly-list} \times \text{int}) \text{ list} \rangle$

We instantiate the characters with typeclass *linorder* to be able to talk about sorted and so on.

**definition** *less-eq-char* ::  $\langle \text{char} \Rightarrow \text{char} \Rightarrow \text{bool} \rangle$  **where**

$\langle \text{less-eq-char } c \ d = (((\text{of-char } c) :: \text{nat}) \leq \text{of-char } d) \rangle$

**definition** *less-char* ::  $\langle \text{char} \Rightarrow \text{char} \Rightarrow \text{bool} \rangle$  **where**

$\langle \text{less-char } c \ d = (((\text{of-char } c) :: \text{nat}) < \text{of-char } d) \rangle$

**global-interpretation** *char*: *linorder less-eq-char less-char*

**using** *linorder-char*

**unfolding** *linorder-class-def class.linorder-def*

*less-eq-char-def*[*symmetric*] *less-char-def*[*symmetric*]

*class.order-def order-class-def*

*class.preorder-def preorder-class-def*

*ord-class-def*

**apply** *auto*

**done**

**abbreviation** *less-than-char* ::  $\langle (\text{char} \times \text{char}) \text{ set} \rangle$  **where**

$\langle \text{less-than-char} \equiv \text{p2rel less-char} \rangle$

**lemma** *less-than-char-def*:

$\langle (x, y) \in \text{less-than-char} \longleftrightarrow \text{less-char } x \ y \rangle$

**by** (*auto simp: p2rel-def*)

**lemma** *trans-less-than-char*[*simp*]:

$\langle \text{trans less-than-char} \rangle$  **and**

*irrefl-less-than-char*:

$\langle \text{irrefl less-than-char} \rangle$  and  
 $\text{antisym-less-than-char}:$   
 $\langle \text{antisym less-than-char} \rangle$   
 by (auto simp: less-than-char-def trans-def irrefl-def antisym-def)

## 8.2 Polynomials

**definition**  $\text{var-order-rel} :: \langle (\text{string} \times \text{string}) \text{ set} \rangle$  **where**  
 $\langle \text{var-order-rel} \equiv \text{lexord less-than-char} \rangle$

**abbreviation**  $\text{var-order} :: \langle \text{string} \Rightarrow \text{string} \Rightarrow \text{bool} \rangle$  **where**  
 $\langle \text{var-order} \equiv \text{rel2p var-order-rel} \rangle$

**abbreviation**  $\text{term-order-rel} :: \langle (\text{term-poly-list} \times \text{term-poly-list}) \text{ set} \rangle$  **where**  
 $\langle \text{term-order-rel} \equiv \text{lexord var-order-rel} \rangle$

**abbreviation**  $\text{term-order} :: \langle \text{term-poly-list} \Rightarrow \text{term-poly-list} \Rightarrow \text{bool} \rangle$  **where**  
 $\langle \text{term-order} \equiv \text{rel2p term-order-rel} \rangle$

**definition**  $\text{term-poly-list-rel} :: \langle (\text{term-poly-list} \times \text{term-poly}) \text{ set} \rangle$  **where**  
 $\langle \text{term-poly-list-rel} = \{(xs, ys).$   
 $\quad ys = \text{mset } xs \wedge$   
 $\quad \text{distinct } xs \wedge$   
 $\quad \text{sorted-wrt } (\text{rel2p var-order-rel}) \text{ } xs\} \rangle$

**definition**  $\text{unsorted-term-poly-list-rel} :: \langle (\text{term-poly-list} \times \text{term-poly}) \text{ set} \rangle$  **where**  
 $\langle \text{unsorted-term-poly-list-rel} = \{(xs, ys).$   
 $\quad ys = \text{mset } xs \wedge \text{distinct } xs\} \rangle$

**definition**  $\text{poly-list-rel} :: \langle \text{('a} \Rightarrow (\text{'a} \times \text{int}) \text{ list} \times \text{mset-polynom}) \text{ set} \rangle$  **where**  
 $\langle \text{poly-list-rel } R = \{(xs, ys).$   
 $\quad (xs, ys) \in \langle R \times_r \text{int-rel} \rangle \text{list-rel } O \text{ list-mset-rel} \wedge$   
 $\quad 0 \notin \# \text{snd '}\# \text{ } ys\} \rangle$

**definition**  $\text{sorted-poly-list-rel-wrt} :: \langle (\text{'a} \Rightarrow \text{'a} \Rightarrow \text{bool})$   
 $\Rightarrow (\text{'a} \times \text{string multiset}) \text{ set} \Rightarrow ((\text{'a} \times \text{int}) \text{ list} \times \text{mset-polynom}) \text{ set} \rangle$  **where**  
 $\langle \text{sorted-poly-list-rel-wrt } S \text{ } R = \{(xs, ys).$   
 $\quad (xs, ys) \in \langle R \times_r \text{int-rel} \rangle \text{list-rel } O \text{ list-mset-rel} \wedge$   
 $\quad \text{sorted-wrt } S \text{ } (\text{map fst } xs) \wedge$   
 $\quad \text{distinct } (\text{map fst } xs) \wedge$   
 $\quad 0 \notin \# \text{snd '}\# \text{ } ys\} \rangle$

**abbreviation**  $\text{sorted-poly-list-rel}$  **where**  
 $\langle \text{sorted-poly-list-rel } R \equiv \text{sorted-poly-list-rel-wrt } R \text{ term-poly-list-rel} \rangle$

**abbreviation**  $\text{sorted-poly-rel}$  **where**  
 $\langle \text{sorted-poly-rel} \equiv \text{sorted-poly-list-rel term-order} \rangle$

**definition**  $\text{sorted-repeat-poly-list-rel-wrt} :: \langle (\text{'a} \Rightarrow \text{'a} \Rightarrow \text{bool})$   
 $\Rightarrow (\text{'a} \times \text{string multiset}) \text{ set} \Rightarrow ((\text{'a} \times \text{int}) \text{ list} \times \text{mset-polynom}) \text{ set} \rangle$  **where**  
 $\langle \text{sorted-repeat-poly-list-rel-wrt } S \text{ } R = \{(xs, ys).$   
 $\quad (xs, ys) \in \langle R \times_r \text{int-rel} \rangle \text{list-rel } O \text{ list-mset-rel} \wedge$   
 $\quad \text{sorted-wrt } S \text{ } (\text{map fst } xs) \wedge$   
 $\quad 0 \notin \# \text{snd '}\# \text{ } ys\} \rangle$

**abbreviation** *sorted-repeat-poly-list-rel* **where**

$\langle \text{sorted-repeat-poly-list-rel } R \equiv \text{sorted-repeat-poly-list-rel-wrt } R \text{ term-poly-list-rel} \rangle$

**abbreviation** *sorted-repeat-poly-rel* **where**

$\langle \text{sorted-repeat-poly-rel} \equiv \text{sorted-repeat-poly-list-rel } (\text{rel2p } (\text{Id} \cup \text{lexord var-order-rel})) \rangle$

**abbreviation** *unsorted-poly-rel* **where**

$\langle \text{unsorted-poly-rel} \equiv \text{poly-list-rel term-poly-list-rel} \rangle$

**lemma** *sorted-poly-list-rel-empty-l[simp]*:

$\langle ([], s') \in \text{sorted-poly-list-rel-wrt } S \ T \longleftrightarrow s' = \{\#\} \rangle$

**by** (*cases s'*)

(*auto simp: sorted-poly-list-rel-wrt-def list-mset-rel-def br-def*)

**definition** *fully-unsorted-poly-list-rel* ::  $\langle - \Rightarrow (('a \times \text{int}) \text{ list} \times \text{mset-polynom}) \text{ set} \rangle$  **where**

$\langle \text{fully-unsorted-poly-list-rel } R = \{(xs, ys) \cdot$

$(xs, ys) \in \langle R \times_r \text{int-rel} \rangle \text{list-rel } O \text{ list-mset-rel} \}$

**abbreviation** *fully-unsorted-poly-rel* **where**

$\langle \text{fully-unsorted-poly-rel} \equiv \text{fully-unsorted-poly-list-rel unsorted-term-poly-list-rel} \rangle$

**lemma** *fully-unsorted-poly-list-rel-empty-iff[simp]*:

$\langle (p, \{\#\}) \in \text{fully-unsorted-poly-list-rel } R \longleftrightarrow p = [] \rangle$

$\langle ([], p') \in \text{fully-unsorted-poly-list-rel } R \longleftrightarrow p' = \{\#\} \rangle$

**by** (*auto simp: fully-unsorted-poly-list-rel-def list-mset-rel-def br-def*)

**definition** *poly-list-rel-with0* ::  $\langle - \Rightarrow (('a \times \text{int}) \text{ list} \times \text{mset-polynom}) \text{ set} \rangle$  **where**

$\langle \text{poly-list-rel-with0 } R = \{(xs, ys) \cdot$

$(xs, ys) \in \langle R \times_r \text{int-rel} \rangle \text{list-rel } O \text{ list-mset-rel} \}$

**abbreviation** *unsorted-poly-rel-with0* **where**

$\langle \text{unsorted-poly-rel-with0} \equiv \text{fully-unsorted-poly-list-rel term-poly-list-rel} \rangle$

**lemma** *poly-list-rel-with0-empty-iff[simp]*:

$\langle (p, \{\#\}) \in \text{poly-list-rel-with0 } R \longleftrightarrow p = [] \rangle$

$\langle ([], p') \in \text{poly-list-rel-with0 } R \longleftrightarrow p' = \{\#\} \rangle$

**by** (*auto simp: poly-list-rel-with0-def list-mset-rel-def br-def*)

**definition** *sorted-repeat-poly-list-rel-with0-wrt* ::  $\langle ('a \Rightarrow 'a \Rightarrow \text{bool})$

$\Rightarrow ('a \times \text{string multiset}) \text{ set} \Rightarrow (('a \times \text{int}) \text{ list} \times \text{mset-polynom}) \text{ set} \rangle$  **where**

$\langle \text{sorted-repeat-poly-list-rel-with0-wrt } S \ R = \{(xs, ys) \cdot$

$(xs, ys) \in \langle R \times_r \text{int-rel} \rangle \text{list-rel } O \text{ list-mset-rel} \wedge$

$\text{sorted-wrt } S \ (\text{map fst } xs) \}$

**abbreviation** *sorted-repeat-poly-list-rel-with0* **where**

$\langle \text{sorted-repeat-poly-list-rel-with0 } R \equiv \text{sorted-repeat-poly-list-rel-with0-wrt } R \text{ term-poly-list-rel} \rangle$

**abbreviation** *sorted-repeat-poly-rel-with0* **where**

$\langle \text{sorted-repeat-poly-rel-with0} \equiv \text{sorted-repeat-poly-list-rel-with0 } (\text{rel2p } (\text{Id} \cup \text{lexord var-order-rel})) \rangle$

**lemma** *term-poly-list-relD*:

```

  ⟨(xs, ys) ∈ term-poly-list-rel ⇒ distinct xs⟩
  ⟨(xs, ys) ∈ term-poly-list-rel ⇒ ys = mset xs⟩
  ⟨(xs, ys) ∈ term-poly-list-rel ⇒ sorted-wrt (rel2p var-order-rel) xs⟩
  ⟨(xs, ys) ∈ term-poly-list-rel ⇒ sorted-wrt (rel2p (Id ∪ var-order-rel)) xs⟩
  apply (auto simp: term-poly-list-rel-def; fail)+
  by (metis (mono-tags, lifting) CollectD UnI2 rel2p-def sorted-wrt-mono-rel split-conv
      term-poly-list-rel-def)

end
theory PAC-Polynoms-Operations
  imports PAC-Polynoms-Term PAC-Checker-Specification
begin

```

## 9 Polynomialss as Lists

### 9.1 Addition

In this section, we refine the polynomials to list. These lists will be used in our checker to represent the polynomials and execute operations.

There is one *key* difference between the list representation and the usual representation: in the former, coefficients can be zero and monomials can appear several times. This makes it easier to reason on intermediate representation where this has not yet been sanitized.

```

fun add-poly-l' :: ⟨llist-polynom × llist-polynom ⇒ llist-polynom⟩ where
  ⟨add-poly-l' (p, []) = p⟩ |
  ⟨add-poly-l' ([], q) = q⟩ |
  ⟨add-poly-l' ((xs, n) # p, (ys, m) # q) =
    (if xs = ys then if n + m = 0 then add-poly-l' (p, q) else
      let pq = add-poly-l' (p, q) in
      ((xs, n + m) # pq)
    else if (xs, ys) ∈ term-order-rel
      then
        let pq = add-poly-l' (p, (ys, m) # q) in
        ((xs, n) # pq)
    else
      let pq = add-poly-l' ((xs, n) # p, q) in
      ((ys, m) # pq)
  )⟩

```

```

definition add-poly-l :: ⟨llist-polynom × llist-polynom ⇒ llist-polynom nres⟩ where
  ⟨add-poly-l = RECT
    (λadd-poly-l (p, q).
      case (p, q) of
        (p, []) ⇒ RETURN p
      | ([], q) ⇒ RETURN q
      | ((xs, n) # p, (ys, m) # q) ⇒
        (if xs = ys then if n + m = 0 then add-poly-l (p, q) else
          do {
            pq ← add-poly-l (p, q);
            RETURN ((xs, n + m) # pq)
          }
        else if (xs, ys) ∈ term-order-rel
          then do {
            pq ← add-poly-l (p, (ys, m) # q);
            RETURN ((xs, n) # pq)
          }
    )
  ⟩

```

```

    }
  else do {
    pq ← add-poly-l ((xs, n) # p, q);
    RETURN ((ys, m) # pq)
  })))

```

**definition** *nonzero-coeffs* **where**

$\langle \text{nonzero-coeffs } a \longleftrightarrow 0 \notin \# \text{ snd } a \rangle$

**lemma** *nonzero-coeffs-simps*[simp]:

$\langle \text{nonzero-coeffs } \{ \# \} \rangle$

$\langle \text{nonzero-coeffs } (\text{add-mset } (xs, n) a) \longleftrightarrow \text{nonzero-coeffs } a \wedge n \neq 0 \rangle$

**by** (auto simp: nonzero-coeffs-def)

**lemma** *nonzero-coeffsD*:

$\langle \text{nonzero-coeffs } a \implies (x, n) \in \# a \implies n \neq 0 \rangle$

**by** (auto simp: nonzero-coeffs-def)

**lemma** *sorted-poly-list-rel-ConsD*:

$\langle ((ys, n) \# p, a) \in \text{sorted-poly-list-rel } S \implies (p, \text{remove1-mset } (\text{mset } ys, n) a) \in \text{sorted-poly-list-rel } S$

$\wedge$

$(\text{mset } ys, n) \in \# a \wedge (\forall x \in \text{set } p. S \text{ } ys \text{ } (\text{fst } x)) \wedge \text{sorted-wrt } (\text{rel2p var-order-rel}) \text{ } ys \wedge$

$\text{distinct } ys \wedge ys \notin \text{set } (\text{map fst } p) \wedge n \neq 0 \wedge \text{nonzero-coeffs } a \rangle$

**unfolding** sorted-poly-list-rel-wrt-def prod.case mem-Collect-eq list-rel-def

**apply** (clarsimp)

**apply** (subst (asm) list.rel-sel)

**apply** (intro conjI)

**apply** (rename-tac y, rule-tac b =  $\langle \text{tl } y \rangle$  in relcompI)

**apply** (auto simp: sorted-poly-list-rel-wrt-def list-mset-rel-def br-def

list.tl-def term-poly-list-rel-def nonzero-coeffs-def split: list.splits)

**done**

**lemma** *sorted-poly-list-rel-Cons-iff*:

$\langle ((ys, n) \# p, a) \in \text{sorted-poly-list-rel } S \longleftrightarrow (p, \text{remove1-mset } (\text{mset } ys, n) a) \in \text{sorted-poly-list-rel } S$

$\wedge$

$(\text{mset } ys, n) \in \# a \wedge (\forall x \in \text{set } p. S \text{ } ys \text{ } (\text{fst } x)) \wedge \text{sorted-wrt } (\text{rel2p var-order-rel}) \text{ } ys \wedge$

$\text{distinct } ys \wedge ys \notin \text{set } (\text{map fst } p) \wedge n \neq 0 \wedge \text{nonzero-coeffs } a \rangle$

**apply** (rule iffI)

**subgoal**

**by** (auto dest!: sorted-poly-list-rel-ConsD)

**subgoal**

**unfolding** sorted-poly-list-rel-wrt-def prod.case mem-Collect-eq

list-rel-def

**apply** (clarsimp)

**apply** (intro conjI)

**apply** (rename-tac y; rule-tac b =  $\langle (\text{mset } ys, n) \# y \rangle$  in relcompI)

**by** (auto simp: sorted-poly-list-rel-wrt-def list-mset-rel-def br-def

term-poly-list-rel-def add-mset-eq-add-mset eq-commute[of -  $\langle \text{mset } - \rangle$ ]

nonzero-coeffs-def

dest!: multi-member-split)

**done**

**lemma** *sorted-repeat-poly-list-rel-ConsD*:

$\langle (ys, n) \# p, a \rangle \in \text{sorted-repeat-poly-list-rel } S \implies (p, \text{remove1-mset } (\text{mset } ys, n) a) \in \text{sorted-repeat-poly-list-rel } S \wedge$

$(\text{mset } ys, n) \in \# a \wedge (\forall x \in \text{set } p. S \text{ } ys \text{ } (\text{fst } x)) \wedge \text{sorted-wrt } (\text{rel2p var-order-rel}) \text{ } ys \wedge$   
 $\text{distinct } ys \wedge n \neq 0 \wedge \text{nonzero-coeffs } a$

**unfolding** *sorted-repeat-poly-list-rel-wrt-def prod.case mem-Collect-eq list-rel-def*

**apply** (*clarsimp*)

**apply** (*subst (asm) list.rel-sel*)

**apply** (*intro conjI*)

**apply** (*rename-tac y, rule-tac b = \langle tl y \rangle in relcompI*)

**apply** (*auto simp: sorted-poly-list-rel-wrt-def list-mset-rel-def br-def list.tl-def term-poly-list-rel-def nonzero-coeffs-def split: list.splits*)

**done**

**lemma** *sorted-repeat-poly-list-rel-Cons-iff*:

$\langle (ys, n) \# p, a \rangle \in \text{sorted-repeat-poly-list-rel } S \longleftrightarrow (p, \text{remove1-mset } (\text{mset } ys, n) a) \in \text{sorted-repeat-poly-list-rel } S \wedge$

$(\text{mset } ys, n) \in \# a \wedge (\forall x \in \text{set } p. S \text{ } ys \text{ } (\text{fst } x)) \wedge \text{sorted-wrt } (\text{rel2p var-order-rel}) \text{ } ys \wedge$   
 $\text{distinct } ys \wedge n \neq 0 \wedge \text{nonzero-coeffs } a$

**apply** (*rule iffI*)

**subgoal**

**by** (*auto dest!: sorted-repeat-poly-list-rel-ConsD*)

**subgoal**

**unfolding** *sorted-repeat-poly-list-rel-wrt-def prod.case mem-Collect-eq list-rel-def*

**apply** (*clarsimp*)

**apply** (*intro conjI*)

**apply** (*rename-tac y, rule-tac b = \langle (\text{mset } ys, n) \# y \rangle in relcompI*)

**by** (*auto simp: sorted-repeat-poly-list-rel-wrt-def list-mset-rel-def br-def term-poly-list-rel-def add-mset-eq-add-mset eq-commute[of - \langle \text{mset } - \rangle] nonzero-coeffs-def dest!: multi-member-split*)

**done**

**lemma** *add-poly-p-add-mset-sum-0*:

$(n + m = 0 \implies \text{add-poly-p}^{**} (A, Aa, \{\#\}) (\{\#\}, \{\#\}, r) \implies$

$\text{add-poly-p}^{**} (\text{add-mset } (\text{mset } ys, n) A, \text{add-mset } (\text{mset } ys, m) Aa, \{\#\}) (\{\#\}, \{\#\}, r)$

**apply** (*rule converse-rtranclp-into-rtranclp*)

**apply** (*rule add-poly-p.add-new-coeff-r*)

**apply** (*rule converse-rtranclp-into-rtranclp*)

**apply** (*rule add-poly-p.add-same-coeff-l*)

**apply** (*rule converse-rtranclp-into-rtranclp*)

**apply** (*auto intro: add-poly-p.rem-0-coeff*)

**done**

**lemma** *monoms-add-poly-l'D*:

$\langle (aa, ba) \in \text{set } (\text{add-poly-l}' x) \implies aa \in \text{fst } ' \text{set } (\text{fst } x) \vee aa \in \text{fst } ' \text{set } (\text{snd } x)$

**by** (*induction x rule: add-poly-l'.induct*)

(*auto split: if-splits*)

**lemma** *add-poly-p-add-to-result*:

$\langle \text{add-poly-p}^{**} (A, B, r) (A', B', r') \Rightarrow$   
 $\text{add-poly-p}^{**}$   
 $(A, B, p + r) (A', B', p + r') \rangle$   
**apply** (induction rule: rtrancp-induct[of add-poly-p  $\langle (-, -, -) \rangle \langle (-, -, -) \rangle$ , split-format(complete), of for  
 $r$ ])  
**subgoal by auto**  
**by** (elim add-poly-pE)  
(metis (no-types, lifting) Pair-inject add-poly-p.intros rtrancp.simps union-mset-add-mset-right)+

**lemma** add-poly-p-add-mset-comb:

$\langle \text{add-poly-p}^{**} (A, Aa, \{\#\}) (\{\#\}, \{\#\}, r) \Rightarrow$   
 $\text{add-poly-p}^{**}$   
 $(\text{add-mset } (xs, n) A, Aa, \{\#\})$   
 $(\{\#\}, \{\#\}, \text{add-mset } (xs, n) r) \rangle$   
**apply** (rule converse-rtrancp-into-rtrancp)  
**apply** (rule add-poly-p.add-new-coeff-l)  
**using** add-poly-p-add-to-result[of A Aa  $\langle \{\#\} \rangle \langle \{\#\} \rangle \langle \{\#\} \rangle r \langle \{\#(xs, n)\#\} \rangle]$   
**by auto**

**lemma** add-poly-p-add-mset-comb2:

$\langle \text{add-poly-p}^{**} (A, Aa, \{\#\}) (\{\#\}, \{\#\}, r) \Rightarrow$   
 $\text{add-poly-p}^{**}$   
 $(\text{add-mset } (ys, n) A, \text{add-mset } (ys, m) Aa, \{\#\})$   
 $(\{\#\}, \{\#\}, \text{add-mset } (ys, n + m) r) \rangle$   
**apply** (rule converse-rtrancp-into-rtrancp)  
**apply** (rule add-poly-p.add-new-coeff-r)  
**apply** (rule converse-rtrancp-into-rtrancp)  
**apply** (rule add-poly-p.add-same-coeff-l)  
**using** add-poly-p-add-to-result[of A Aa  $\langle \{\#\} \rangle \langle \{\#\} \rangle \langle \{\#\} \rangle r \langle \{\#(ys, n+m)\#\} \rangle]$   
**by auto**

**lemma** add-poly-p-add-mset-comb3:

$\langle \text{add-poly-p}^{**} (A, Aa, \{\#\}) (\{\#\}, \{\#\}, r) \Rightarrow$   
 $\text{add-poly-p}^{**}$   
 $(A, \text{add-mset } (ys, m) Aa, \{\#\})$   
 $(\{\#\}, \{\#\}, \text{add-mset } (ys, m) r) \rangle$   
**apply** (rule converse-rtrancp-into-rtrancp)  
**apply** (rule add-poly-p.add-new-coeff-r)  
**using** add-poly-p-add-to-result[of A Aa  $\langle \{\#\} \rangle \langle \{\#\} \rangle \langle \{\#\} \rangle r \langle \{\#(ys, m)\#\} \rangle]$   
**by auto**

**lemma** total-on-lexord:

$\langle \text{Relation.total-on UNIV } R \Rightarrow \text{Relation.total-on UNIV } (\text{lexord } R) \rangle$   
**apply** (auto simp: Relation.total-on-def)  
**by** (meson lexord-linear)

**lemma** antisym-lexord:

$\langle \text{antisym } R \Rightarrow \text{irrefl } R \Rightarrow \text{antisym } (\text{lexord } R) \rangle$   
**by** (auto simp: antisym-def lexord-def irrefl-def  
elim!: list-match-lel-lel)

**lemma** less-than-char-linear:

$\langle (a, b) \in \text{less-than-char} \vee$   
 $a = b \vee (b, a) \in \text{less-than-char} \rangle$

by (auto simp: less-than-char-def)

**lemma** total-on-lexord-less-than-char-linear:

$\langle xs \neq ys \implies (xs, ys) \notin \text{lexord } (\text{lexord less-than-char}) \longleftrightarrow$   
 $(ys, xs) \in \text{lexord } (\text{lexord less-than-char}) \rangle$   
**using** lexord-linear[of  $\langle \text{lexord less-than-char} \rangle xs\ ys$ ]  
**using** lexord-linear[of  $\langle \text{less-than-char} \rangle$  less-than-char-linear]  
**using** lexord-irrefl[OF irrefl-less-than-char]  
 $\text{antisym-lexord}[OF \text{antisym-lexord}[OF \text{antisym-less-than-char irrefl-less-than-char}]]$   
**apply** (auto simp: antisym-def Relation.total-on-def)  
**done**

**lemma** sorted-poly-list-rel-nonzeroD:

$\langle (p, r) \in \text{sorted-poly-list-rel term-order} \implies$   
 $\text{nonzero-coeffs } (r) \rangle$   
 $\langle (p, r) \in \text{sorted-poly-list-rel } (\text{rel2p } (\text{lexord } (\text{lexord less-than-char}))) \implies$   
 $\text{nonzero-coeffs } (r) \rangle$   
**by** (auto simp: sorted-poly-list-rel-wrt-def nonzero-coeffs-def)

**lemma** add-poly-l'-add-poly-p:

**assumes**  $\langle (pq, pq') \in \text{sorted-poly-rel} \times_r \text{sorted-poly-rel} \rangle$   
**shows**  $\langle \exists r. (\text{add-poly-l}' pq, r) \in \text{sorted-poly-rel} \wedge$   
 $\text{add-poly-p}^{**} (\text{fst } pq', \text{snd } pq', \{\#\}) (\{\#\}, \{\#\}, r) \rangle$   
**supply** [[goals-limit=1]]  
**using** assms  
**apply** (induction  $\langle pq \rangle$  arbitrary:  $pq'$  rule: add-poly-l'.induct)  
**subgoal for**  $p\ pq'$   
**using** add-poly-p-empty-l[of  $\langle \text{fst } pq' \rangle \langle \{\#\} \rangle \langle \{\#\} \rangle$ ]  
**by** (cases  $pq'$ ) (auto intro!: exI[of -  $\langle \text{fst } pq' \rangle$ ])  
**subgoal for**  $x\ p\ pq'$   
**using** add-poly-p-empty-r[of  $\langle \{\#\} \rangle \langle \text{snd } pq' \rangle \langle \{\#\} \rangle$ ]  
**by** (cases  $pq'$ ) (auto intro!: exI[of -  $\langle \text{snd } pq' \rangle$ ])  
**subgoal premises**  $p\ \text{for } xs\ n\ p\ ys\ m\ q\ pq'$   
**apply** (cases  $pq'$ ) — Isabelle does a completely stupid case distinction here  
**apply** (cases  $\langle xs = ys \rangle$ )  
**subgoal**  
**apply** (cases  $\langle n + m = 0 \rangle$ )  
**subgoal**  
**using**  $p(1)[\text{of } \langle (\text{remove1-mset } (\text{mset } xs, n) (\text{fst } pq'), \text{remove1-mset } (\text{mset } ys, m) (\text{snd } pq')) \rangle]$   
 $p(5-)$   
**apply** (auto dest!: sorted-poly-list-rel-ConsD multi-member-split)  
**using** add-poly-p-add-mset-sum-0 **by** blast  
**subgoal**  
**using**  $p(2)[\text{of } \langle (\text{remove1-mset } (\text{mset } xs, n) (\text{fst } pq'), \text{remove1-mset } (\text{mset } ys, m) (\text{snd } pq')) \rangle]$   
 $p(5-)$   
**apply** (auto dest!: sorted-poly-list-rel-ConsD multi-member-split)  
**apply** (rule-tac  $x = \langle \text{add-mset } (\text{mset } ys, n + m) r \rangle$  **in**  $exI$ )  
**apply** (fastforce dest!: monoms-add-poly-l'D simp: sorted-poly-list-rel-Cons-iff rel2p-def  
sorted-poly-list-rel-nonzeroD var-order-rel-def  
intro: add-poly-p-add-mset-comb2)  
**done**  
**done**  
**subgoal**



```

apply (cases  $\langle (xs, ys) \in \text{term-order-rel} \rangle$ )
subgoal
  using  $p(3)[\text{of } \langle (\text{remove1-mset } (\text{mset } xs, n) (\text{fst } pq'), (\text{snd } pq') \rangle) ] p(5-)$ 
  apply (auto dest!: multi-member-split simp: sorted-poly-list-rel-Cons-iff rel2p-def)
  apply (rule-tac  $x = \langle \text{add-mset } (\text{mset } xs, n) r \rangle$  in exI)
  apply (auto dest!: monoms-add-poly-l'D)
  apply (auto intro: lexord-trans add-poly-p-add-mset-comb simp: lexord-transI var-order-rel-def)
  apply (rule lexord-trans)
  apply assumption
  apply (auto intro: lexord-trans add-poly-p-add-mset-comb simp: lexord-transI
    sorted-poly-list-rel-nonzeroD var-order-rel-def)
  using total-on-lexord-less-than-char-linear by fastforce

```

```

subgoal
  using  $p(4)[\text{of } \langle (\text{fst } pq', \text{remove1-mset } (\text{mset } ys, m) (\text{snd } pq')) \rangle ] p(5-)$ 
  apply (auto dest!: multi-member-split simp: sorted-poly-list-rel-Cons-iff rel2p-def
    var-order-rel-def)
  apply (rule-tac  $x = \langle \text{add-mset } (\text{mset } ys, m) r \rangle$  in exI)
  apply (auto dest!: monoms-add-poly-l'D
    simp: total-on-lexord-less-than-char-linear)
  apply (auto intro: lexord-trans add-poly-p-add-mset-comb simp: lexord-transI
    total-on-lexord-less-than-char-linear var-order-rel-def)
  apply (rule lexord-trans)
  apply assumption
  apply (auto intro: lexord-trans add-poly-p-add-mset-comb3 simp: lexord-transI
    sorted-poly-list-rel-nonzeroD var-order-rel-def)
  using total-on-lexord-less-than-char-linear by fastforce
done
done
done

```

```

lemma add-poly-l-add-poly:
   $\langle \text{add-poly-l } x = \text{RETURN } (\text{add-poly-l'} x) \rangle$ 
  unfolding add-poly-l-def
  by (induction x rule: add-poly-l'.induct)
  (solves  $\langle \text{subst RECT-unfold, refine-mono, simp split: list.split} \rangle +$ )

```

```

lemma add-poly-l-spec:
   $\langle (\text{add-poly-l, uncurry } (\lambda p q. \text{SPEC}(\lambda r. \text{add-poly-p}^{**} (p, q, \{\#\}) (\{\#\}, \{\#\}, r)))) \in$ 
     $\text{sorted-poly-rel} \times_r \text{sorted-poly-rel} \rightarrow_f \langle \text{sorted-poly-rel} \rangle \text{nres-rel} \rangle$ 
  unfolding add-poly-l-add-poly
  apply (intro nres-relI frefI)
  apply (drule add-poly-l'-add-poly-p)
  apply (auto simp: conc-fun-RES)
done

```

```

definition sort-poly-spec ::  $\langle \text{l-list-polynom} \Rightarrow \text{l-list-polynom nres} \rangle$  where
   $\langle \text{sort-poly-spec } p =$ 
     $\text{SPEC}(\lambda p'. \text{mset } p = \text{mset } p' \wedge \text{sorted-wrt } (\text{rel2p } (\text{Id} \cup \text{term-order-rel})) (\text{map fst } p')) \rangle$ 

```

```

lemma sort-poly-spec-id:
  assumes  $\langle (p, p') \in \text{unsorted-poly-rel} \rangle$ 
  shows  $\langle \text{sort-poly-spec } p \leq \Downarrow (\text{sorted-repeat-poly-rel}) (\text{RETURN } p') \rangle$ 
proof –

```

```

obtain  $y$  where
   $py: \langle (p, y) \in \langle \text{term-poly-list-rel} \times_r \text{int-rel} \rangle \text{list-rel} \rangle$  and
   $p'-y: \langle p' = \text{mset } y \rangle$  and
   $\text{zero}: \langle 0 \notin \# \text{snd } ' \# p' \rangle$ 
  using assms
  unfolding sort-poly-spec-def poly-list-rel-def sorted-poly-list-rel-wrt-def
  by (auto simp: list-mset-rel-def br-def)
then have [simp]:  $\langle \text{length } y = \text{length } p \rangle$ 
  by (auto simp: list-rel-def list-all2-conv-all-nth)
have  $H: \langle (x, p') \in \langle \text{term-poly-list-rel} \times_r \text{int-rel} \rangle \text{list-rel } O \text{ list-mset-rel} \rangle$ 
  if  $px: \langle \text{mset } p = \text{mset } x \rangle$  and  $\langle \text{sorted-wrt } (\text{rel2p } (\text{Id} \cup \text{lexord var-order-rel})) (\text{map fst } x) \rangle$ 
  for  $x :: \langle \text{list-polynom} \rangle$ 
proof –
  obtain  $f$  where
     $f: \langle \text{bij-betw } f \{ .. < \text{length } x \} \{ .. < \text{length } p \} \rangle$  and
    [simp]:  $\langle \bigwedge i. i < \text{length } x \implies x ! i = p ! (f i) \rangle$ 
    using  $px$  apply – apply (subst (asm)(2) eq-commute) unfolding mset-eq-perm
    by (auto dest!: permutation-Ex-bij)
  let  $?y = \langle \text{map } (\lambda i. y ! f i) [0 .. < \text{length } x] \rangle$ 
  have  $\langle i < \text{length } y \implies (p ! f i, y ! f i) \in \text{term-poly-list-rel} \times_r \text{int-rel} \rangle$  for  $i$ 
    using list-all2-nthD[of - p y]
       $\langle f i, OF py[\text{unfolded list-rel-def mem-Collect-eq prod.case}]$ 
       $\text{mset-eq-length}[OF px] f$ 
    by (auto simp: list-rel-def list-all2-conv-all-nth bij-betw-def)
  then have  $\langle (x, ?y) \in \langle \text{term-poly-list-rel} \times_r \text{int-rel} \rangle \text{list-rel} \rangle$  and
     $xy: \langle \text{length } x = \text{length } y \rangle$ 
    using  $py$  list-all2-nthD[of  $\langle \text{rel2p } (\text{term-poly-list-rel} \times_r \text{int-rel}) \rangle p y$ ]
       $\langle f i \text{ for } i, \text{simplified} \rangle \text{mset-eq-length}[OF px]$ 
    by (auto simp: list-rel-def list-all2-conv-all-nth)
  moreover {
    have  $f: \langle \text{mset-set } \{0..<\text{length } x\} = f \# \text{mset-set } \{0..<\text{length } x\} \rangle$ 
      using  $f$  mset-eq-length[OF px]
      by (auto simp: bij-betw-def lessThan-atLeast0 image-mset-mset-set)
    have  $\langle \text{mset } y = \{ \# y ! f x. x \in \# \text{mset-set } \{0..<\text{length } x\} \# \} \rangle$ 
      by (subst drop-0[symmetric], subst mset-drop-upto, subst xy[symmetric], subst f)
      auto
    then have  $\langle (?y, p') \in \text{list-mset-rel} \rangle$ 
      by (auto simp: list-mset-rel-def br-def p'-y)
  }
  ultimately show ?thesis
    by (auto intro!: relcompI[of - ?y])
qed
show ?thesis
  using zero
  unfolding sort-poly-spec-def poly-list-rel-def sorted-repeat-poly-list-rel-wrt-def
  by refine-rcg (auto intro: H)
qed

```

## 9.2 Multiplication

```

fun mult-monoms ::  $\langle \text{term-poly-list} \Rightarrow \text{term-poly-list} \Rightarrow \text{term-poly-list} \rangle$  where
   $\langle \text{mult-monoms } p [] = p \rangle$  |
   $\langle \text{mult-monoms } [] p = p \rangle$  |
   $\langle \text{mult-monoms } (x \# p) (y \# q) =$ 
     $(\text{if } x = y \text{ then } x \# \text{mult-monoms } p q$ 

```

else if  $(x, y) \in \text{var-order-rel}$  then  $x \# \text{mult-monoms } p (y \# q)$   
 else  $y \# \text{mult-monoms } (x \# p) q$

**lemma** *term-poly-list-rel-empty-iff*[simp]:  
 $\langle \langle [], q' \rangle \in \text{term-poly-list-rel} \iff q' = \{\#\} \rangle$   
**by** (auto simp: *term-poly-list-rel-def*)

**lemma** *term-poly-list-rel-Cons-iff*:  
 $\langle (y \# p, p') \in \text{term-poly-list-rel} \iff$   
 $(p, \text{remove1-mset } y \ p') \in \text{term-poly-list-rel} \wedge$   
 $y \in \# p' \wedge y \notin \text{set } p \wedge y \notin \# \text{remove1-mset } y \ p' \wedge$   
 $(\forall x \in \# \text{mset } p. (y, x) \in \text{var-order-rel}) \rangle$   
**apply** (auto simp: *term-poly-list-rel-def* *rel2p-def* *dest!*: *multi-member-split*)  
**by** (*metis* *list.set-intros*(1) *list-of-mset-exi* *mset.simps*(2) *mset-eq-setD*)

**lemma** *var-order-rel-antisym*[simp]:  
 $\langle (y, y) \notin \text{var-order-rel} \rangle$   
**by** (simp add: *less-than-char-def* *lexord-irreflexive* *var-order-rel-def*)

**lemma** *term-poly-list-rel-remdups-mset*:  
 $\langle (p, p') \in \text{term-poly-list-rel} \implies$   
 $(p, \text{remdups-mset } p') \in \text{term-poly-list-rel} \rangle$   
**by** (auto simp: *term-poly-list-rel-def* *distinct-mset-remdups-mset-id* *simp flip*: *distinct-mset-mset-distinct*)

**lemma** *var-notin-notin-mult-monomsD*:  
 $\langle y \in \text{set } (\text{mult-monoms } p \ q) \implies y \in \text{set } p \vee y \in \text{set } q \rangle$   
**by** (*induction* *p q arbitrary*:  $p' \ q'$  *rule*: *mult-monoms.induct*) (*auto split*: *if-splits*)

**lemma** *term-poly-list-rel-set-mset*:  
 $\langle (p, q) \in \text{term-poly-list-rel} \implies \text{set } p = \text{set-mset } q \rangle$   
**by** (auto simp: *term-poly-list-rel-def*)

**lemma** *mult-monoms-spec*:  
 $\langle (\text{mult-monoms}, (\lambda a \ b. \text{remdups-mset } (a + b))) \in \text{term-poly-list-rel} \rightarrow \text{term-poly-list-rel} \rightarrow \text{term-poly-list-rel} \rangle$   
**apply** (*intro fun-relI*)  
**apply** (*rename-tac* *p p' q q'*)  
**subgoal for**  $p \ p' \ q \ q'$   
**apply** (*induction* *p q arbitrary*:  $p' \ q'$  *rule*: *mult-monoms.induct*)  
**subgoal by** (auto simp: *term-poly-list-rel-Cons-iff* *rel2p-def* *term-poly-list-rel-remdups-mset*)  
**subgoal for**  $x \ p \ p' \ q'$   
**by** (auto simp: *term-poly-list-rel-Cons-iff* *rel2p-def* *term-poly-list-rel-remdups-mset*  
*dest!*: *multi-member-split*[*of* -  $q'$ ])  
**subgoal premises**  $p$  **for**  $x \ p \ y \ q \ p' \ q'$   
**apply** (*cases*  $x = y$ )  
**subgoal**  
**using**  $p(1)[\text{of } \langle \text{remove1-mset } y \ p' \rangle \langle \text{remove1-mset } y \ q' \rangle] \ p(4-)$   
**apply** (auto simp: *term-poly-list-rel-Cons-iff* *rel2p-def*  
*dest!*: *var-notin-notin-mult-monomsD*  
*dest!*: *multi-member-split*)  
**by** (*metis* *set-mset-remdups-mset union-iff union-single-eq-member*)  
**apply** (*cases*  $\langle (x, y) \in \text{var-order-rel} \rangle$ )  
**subgoal**  
**using**  $p(2)[\text{of } \langle \text{remove1-mset } x \ p' \rangle \langle q' \rangle] \ p(4-)$   
**apply** (auto simp: *term-poly-list-rel-Cons-iff*  
*term-poly-list-rel-set-mset* *rel2p-def* *var-order-rel-def*)

```

    dest!: multi-member-split[of - p] multi-member-split[of - q]
    var-notin-notin-mult-monomsD
    split: if-splits)
  apply (meson lexord-cons-cons list.inject total-on-lexord-less-than-char-linear)
  apply (meson lexord-cons-cons list.inject total-on-lexord-less-than-char-linear)
  apply (meson lexord-cons-cons list.inject total-on-lexord-less-than-char-linear)
  using lexord-trans trans-less-than-char var-order-rel-antisym
  unfolding var-order-rel-def apply blast+
done
subgoal
  using p(3)[of ⟨p⟩ (remove1-mset y q) p(4-)]
  apply (auto simp: term-poly-list-rel-Cons-iff rel2p-def
    term-poly-list-rel-set-mset rel2p-def var-order-rel-antisym
    dest!: multi-member-split[of - p] multi-member-split[of - q]
    var-notin-notin-mult-monomsD
    split: if-splits)
  using lexord-trans trans-less-than-char var-order-rel-antisym
  unfolding var-order-rel-def apply blast
  apply (meson lexord-cons-cons list.inject total-on-lexord-less-than-char-linear)
  by (meson less-than-char-linear lexord-linear lexord-trans trans-less-than-char)
done
done
done

definition mult-monomials :: (term-poly-list × int ⇒ term-poly-list × int ⇒ term-poly-list × int) where
  ⟨mult-monomials = (λ(x, a) (y, b). (mult-monoms x y, a * b))⟩

definition mult-poly-raw :: (llist-polynom ⇒ llist-polynom ⇒ llist-polynom) where
  ⟨mult-poly-raw p q = foldl (λb x. map (mult-monomials x) q @ b) [] p⟩

fun map-append where
  ⟨map-append f b [] = b⟩ |
  ⟨map-append f b (x # xs) = f x # map-append f b xs⟩

lemma map-append-alt-def:
  ⟨map-append f b xs = map f xs @ b⟩
by (induction f b xs rule: map-append.induct)
  auto

lemma foldl-append-empty:
  ⟨NO-MATCH [] xs ⇒ foldl (λb x. f x @ b) xs p = foldl (λb x. f x @ b) [] p @ xs⟩
apply (induction p arbitrary: xs)
apply simp
by (metis (mono-tags, lifting) NO-MATCH-def append.assoc append-self-conv foldl-Cons)

lemma poly-list-rel-empty-iff[simp]:
  ⟨([], r) ∈ poly-list-rel R ⟷ r = {#}⟩
by (auto simp: poly-list-rel-def list-mset-rel-def br-def)

lemma mult-poly-raw-simp[simp]:
  ⟨mult-poly-raw [] q = []⟩
  ⟨mult-poly-raw (x # p) q = mult-poly-raw p q @ map (mult-monomials x) q⟩
subgoal by (auto simp: mult-poly-raw-def)

```

subgoal by (induction p) (auto simp: mult-poly-row-def foldl-append-empty)  
done

**lemma** sorted-poly-list-relD:

$\langle (q, q') \in \text{sorted-poly-list-rel } R \implies q' = (\lambda(a, b). (\text{mset } a, b)) \text{ `# mset } q \rangle$   
**apply** (induction q arbitrary: q')  
**apply** (auto simp: sorted-poly-list-rel-wrt-def list-mset-rel-def br-def  
list-rel-split-right-iff)  
**apply** (subst (asm)(2) term-poly-list-rel-def)  
**apply** (simp add: relcomp.relcompI)  
done

**lemma** list-all2-in-set-ExD:

$\langle \text{list-all2 } R \text{ } p \text{ } q \implies x \in \text{set } p \implies \exists y \in \text{set } q. R \text{ } x \text{ } y \rangle$   
**by** (induction p q rule: list-all2-induct)  
auto

**inductive-cases** mult-poly-p-elim:  $\langle \text{mult-poly-p } q \text{ } (A, r) \text{ } (B, r') \rangle$

**lemma** mult-poly-p-add-mset-same:

$\langle (\text{mult-poly-p } q')^{**} (A, r) (B, r') \implies (\text{mult-poly-p } q')^{**} (\text{add-mset } x \text{ } A, r) (\text{add-mset } x \text{ } B, r') \rangle$   
**apply** (induction rule: rtranclp-induct[*of*  $\langle \text{mult-poly-p } q' \rangle \langle (p, r) \rangle \langle (p', q') \rangle$  **for**  $p' \text{ } q'$ , *split-format(complete)*])  
**apply** (auto elim!: mult-poly-p-elim intro: mult-poly-p.intros)  
**by** (smt add-mset-commute mult-step rtranclp.rtrancl-into-rtrancl)

**lemma** mult-poly-row-mult-poly-p:

**assumes**  $\langle (p, p') \in \text{sorted-poly-rel} \rangle$  **and**  $\langle (q, q') \in \text{sorted-poly-rel} \rangle$   
**shows**  $\langle \exists r. (\text{mult-poly-row } p \text{ } q, r) \in \text{unsorted-poly-rel} \wedge (\text{mult-poly-p } q')^{**} (p', \{\#\}) (\{\#\}, r) \rangle$

**proof** –

**have**  $H: \langle (q, q') \in \text{sorted-poly-list-rel term-order} \implies n < \text{length } q \implies$   
distinct aa  $\implies$  sorted-wrt var-order aa  $\implies$   
 $(\text{mult-monoms aa } (\text{fst } (q \text{ ! } n)),$   
 $\text{mset } (\text{mult-monoms aa } (\text{fst } (q \text{ ! } n))))$   
 $\in \text{term-poly-list-rel} \rangle$  **for** aa n  
**using** mult-monoms-spec[unfolded fun-rel-def, simplified] **apply** –  
**apply** (drule bspec[*of* - -  $\langle (aa, (\text{mset } aa)) \rangle$ ])  
**apply** (auto simp: term-poly-list-rel-def)[]  
**unfolding** prod.case sorted-poly-list-rel-wrt-def  
**apply** clarsimp  
**subgoal for** y  
**apply** (drule bspec[*of* - -  $\langle (\text{fst } (q \text{ ! } n), \text{mset } (\text{fst } (q \text{ ! } n))) \rangle$ ])  
**apply** (cases  $\langle q \text{ ! } n \rangle$ ; cases  $\langle y \text{ ! } n \rangle$ )  
**using** param-nth[*of* n y n q  $\langle \text{term-poly-list-rel } \times_r \text{ int-rel} \rangle$ ]  
**by** (auto simp: list-rel-imp-same-length term-poly-list-rel-def)  
done

**have**  $H': \langle (q, q') \in \text{sorted-poly-list-rel term-order} \implies$

distinct aa  $\implies$  sorted-wrt var-order aa  $\implies$   
 $(ab, ba) \in \text{set } q \implies$   
 $\text{remdups-mset } (\text{mset } aa + \text{mset } ab) = \text{mset } (\text{mult-monoms aa } ab) \rangle$  **for** aa n ab ba  
**using** mult-monoms-spec[unfolded fun-rel-def, simplified] **apply** –  
**apply** (drule bspec[*of* - -  $\langle (aa, (\text{mset } aa)) \rangle$ ])  
**apply** (auto simp: term-poly-list-rel-def)[]  
**unfolding** prod.case sorted-poly-list-rel-wrt-def  
**apply** clarsimp

```

subgoal for y
  apply (drule bspec[of - - (ab, mset ab)])
  apply (auto simp: list-rel-imp-same-length term-poly-list-rel-def list-rel-def
    dest: list-all2-in-set-ExD)
done
done

have H: (q, q') ∈ sorted-poly-list-rel term-order ⇒
  a = (aa, b) ⇒
  (pq, r) ∈ unsorted-poly-rel ⇒
  p' = add-mset (mset aa, b) A ⇒
  ∀ x ∈ set p. term-order aa (fst x) ⇒
  sorted-wrt var-order aa ⇒
  distinct aa ⇒ b ≠ 0 ⇒
  (∧ aaa. (aaa, 0) ∉ # q') ⇒
  (pq @
    map (mult-monomials (aa, b)) q,
    {#case x of (ys, n) ⇒ (remdups-mset (mset aa + ys), n * b)
     . x ∈ # q'#} +
    r)
  ∈ unsorted-poly-rel for a p p' pq aa b r
apply (auto simp: poly-list-rel-def)
apply (rule-tac b = (y @ map (λ(a,b). (mset a, b)) (map (mult-monomials (aa, b)) q)) in relcompI)
apply (auto simp: list-rel-def list-all2-append list-all2-lengthD H
  list-mset-rel-def br-def mult-monomials-def case-prod-beta intro!: list-all2-all-nthI
  simp: sorted-poly-list-relD)
apply (subst sorted-poly-list-relD[of q q' term-order])
apply (auto simp: case-prod-beta H' intro!: image-mset-cong)
done

show ?thesis
  using assms
  apply (induction p arbitrary: p')
  subgoal
    by auto
  subgoal premises p for a p p'
    using p(1)[of (remove1-mset (mset (fst a), snd a) p')] p(2-)
    apply (cases a)
    apply (auto simp: sorted-poly-list-rel-Cons-iff
      dest!: multi-member-split)
    apply (rule-tac x = (λ(ys, n). (remdups-mset (mset (fst a) + ys), n * snd a)) '# q' + r in exI)
    apply (auto 5 3 intro: mult-poly-p.intros simp: intro!: H
      dest: sorted-poly-list-rel-nonzeroD nonzero-coeffsD)
    apply (rule rtranclp-trans)
    apply (rule mult-poly-p-add-mset-same)
    apply assumption
    apply (rule converse-rtranclp-into-rtranclp)
    apply (auto intro!: mult-poly-p.intros simp: ac-simps)
    done
  done
qed

fun merge-coeffs :: (llist-polynom ⇒ llist-polynom) where
  merge-coeffs [] = [] |
  merge-coeffs [(xs, n)] = [(xs, n)] |

```

$\langle \text{merge-coeffs } ((xs, n) \# (ys, m) \# p) =$   
 $\quad (\text{if } xs = ys$   
 $\quad \text{then if } n + m \neq 0 \text{ then merge-coeffs } ((xs, n + m) \# p) \text{ else merge-coeffs } p$   
 $\quad \text{else } (xs, n) \# \text{merge-coeffs } ((ys, m) \# p)) \rangle$

**abbreviation**  $(\text{in } -)\text{mononoms} :: \langle \text{l-list-polynom} \Rightarrow \text{term-poly-list set} \rangle$  **where**  
 $\langle \text{mononoms } p \equiv \text{fst 'set } p \rangle$

**lemma** *fst-normalize-polynom-subset*:  
 $\langle \text{mononoms } (\text{merge-coeffs } p) \subseteq \text{mononoms } p \rangle$   
**by** (*induction p rule: merge-coeffs.induct*) *auto*

**lemma** *fst-normalize-polynom-subsetD*:  
 $\langle (a, b) \in \text{set } (\text{merge-coeffs } p) \implies a \in \text{mononoms } p \rangle$   
**apply** (*induction p rule: merge-coeffs.induct*)  
**subgoal**  
**by** *auto*  
**subgoal**  
**by** *auto*  
**subgoal**  
**by** (*auto split: if-splits*)  
**done**

**lemma** *distinct-merge-coeffs*:  
**assumes**  $\langle \text{sorted-wrt } R \text{ (map fst } xs) \rangle$  **and**  $\langle \text{transp } R \rangle \langle \text{antisym } R \rangle$   
**shows**  $\langle \text{distinct (map fst (merge-coeffs } xs)) \rangle$   
**using** *assms*  
**by** (*induction xs rule: merge-coeffs.induct*)  
 $(\text{auto } 5 \ 4 \text{ dest: antisymD dest!: fst-normalize-polynom-subsetD})$

**lemma** *in-set-merge-coeffsD*:  
 $\langle (a, b) \in \text{set } (\text{merge-coeffs } p) \implies \exists b. (a, b) \in \text{set } p \rangle$   
**by** (*auto dest!: fst-normalize-polynom-subsetD*)

**lemma** *rtrancp-normalize-poly-add-mset*:  
 $\langle \text{normalize-poly-p}^{**} A \ r \implies \text{normalize-poly-p}^{**} (\text{add-mset } x \ A) \ (\text{add-mset } x \ r) \rangle$   
**by** (*induction rule: rtrancp-induct*)  
 $(\text{auto dest: normalize-poly-p.keep-coeff[of - - } x])$

**lemma** *nonzero-coeffs-diff*:  
 $\langle \text{nonzero-coeffs } A \implies \text{nonzero-coeffs } (A - B) \rangle$   
**by** (*auto simp: nonzero-coeffs-def dest: in-diffD*)

**lemma** *merge-coeffs-is-normalize-poly-p*:  
 $\langle (xs, ys) \in \text{sorted-repeat-poly-rel} \implies \exists r. (\text{merge-coeffs } xs, r) \in \text{sorted-poly-rel} \wedge \text{normalize-poly-p}^{**} ys$   
 $r \rangle$   
**apply** (*induction xs arbitrary: ys rule: merge-coeffs.induct*)  
**subgoal by** (*auto simp: sorted-repeat-poly-list-rel-wrt-def sorted-poly-list-rel-wrt-def*)  
**subgoal**  
**by** (*auto simp: sorted-repeat-poly-list-rel-wrt-def sorted-poly-list-rel-wrt-def*)  
**subgoal premises p for xs n ys m p ysa**  
**apply** (*cases xs = ys, cases m+n  $\neq$  0*)

```

subgoal
  using p(1)[of ⟨add-mset (mset ys, m+n) ysa - {#(mset ys, m), (mset ys, n)#}⟩] p(4-)
  apply (auto simp: sorted-poly-list-rel-Cons-iff ac-simps add-mset-commute
    remove1-mset-add-mset-If nonzero-coeffs-diff sorted-repeat-poly-list-rel-Cons-iff)
  apply (rule-tac x = ⟨r⟩ in exI)
  using normalize-poly-p.merge-dup-coeff[of ⟨ysa - {#(mset ys, m), (mset ys, n)#}⟩ ysa - {#(mset
ys, m), (mset ys, n)#}⟩ ⟨mset ys⟩ m n]
  apply (auto dest!: multi-member-split simp del: normalize-poly-p.merge-dup-coeff)
  by (metis add-mset-commute converse-rtranclp-into-rtranclp)
subgoal
  using p(2)[of ⟨ysa - {#(mset ys, m), (mset ys, n)#}⟩] p(4-)
  apply (auto simp: sorted-poly-list-rel-Cons-iff ac-simps add-mset-commute
    remove1-mset-add-mset-If nonzero-coeffs-diff sorted-repeat-poly-list-rel-Cons-iff)
  apply (rule-tac x = ⟨r⟩ in exI)
  using normalize-poly-p.rem-0-coeff[of ⟨add-mset (mset ys, m+n) ysa - {#(mset ys, m), (mset
ys, n)#}⟩ ⟨add-mset (mset ys, m+n) ysa - {#(mset ys, m), (mset ys, n)#}⟩ ⟨mset ys⟩]
  using normalize-poly-p.merge-dup-coeff[of ⟨ysa - {#(mset ys, m), (mset ys, n)#}⟩ ysa - {#(mset
ys, m), (mset ys, n)#}⟩ ⟨mset ys⟩ m n]
  apply (auto intro: normalize-poly-p.intros add-mset-commute add-mset-commute converse-rtranclp-into-rtranclp
    dest!: multi-member-split
    simp del: normalize-poly-p.rem-0-coeff
    simp: add-eq-0-iff2)
  by (metis (full-types) add.right-inverse converse-rtranclp-into-rtranclp merge-dup-coeff normalize-poly-p.rem-0-coeff
same)
subgoal
  using p(3)[of ⟨add-mset (mset ys, m) ysa - {#(mset xs, n), (mset ys, m)#}⟩] p(4-)
  apply (auto simp: sorted-poly-list-rel-Cons-iff ac-simps add-mset-commute
    remove1-mset-add-mset-If sorted-repeat-poly-list-rel-Cons-iff)
  apply (rule-tac x = ⟨add-mset (mset xs, n) r⟩ in exI)
  apply (auto dest!: in-set-merge-coeffsD)
  apply (auto intro: normalize-poly-p.intros rtranclp-normalize-poly-add-mset
    simp: rel2p-def var-order-rel-def
    dest!: multi-member-split
    dest: sorted-poly-list-rel-nonzeroD)
  using total-on-lexord-less-than-char-linear apply fastforce
  using total-on-lexord-less-than-char-linear apply fastforce
done
done
done

```

### 9.3 Normalisation

**definition** *normalize-poly* **where**

```

⟨normalize-poly p = do {
  p ← sort-poly-spec p;
  RETURN (merge-coeffs p)
}⟩

```

**definition** *sort-coeff* :: ⟨string list ⇒ string list nres⟩ **where**

```

⟨sort-coeff ys = SPEC(λxs. mset xs = mset ys ∧ sorted-wrt (rel2p (Id ∪ var-order-rel)) xs)⟩

```

**lemma** *distinct-var-order-Id-var-order*:

```

⟨distinct a ⇒ sorted-wrt (rel2p (Id ∪ var-order-rel)) a ⇒
  sorted-wrt var-order a⟩
by (induction a) (auto simp: rel2p-def)

```

**definition** *sort-all-coeffs* :: ⟨llist-polynom ⇒ llist-polynom nres⟩ **where**



$\langle \text{sort-all-coeffs } xs = \text{monadic-nfoldli } xs \ (\lambda\cdot. \text{RETURN True}) \ (\lambda(a, n) \ b. \text{do } \{a \leftarrow \text{sort-coeff } a; \text{RETURN } ((a, n) \# b)\}) \ \square \rangle$

**lemma** *sort-all-coeffs-gen*:

**assumes**  $\langle (\forall xs \in \text{mononoms } xs'. \text{sorted-wrt } (\text{rel2p } (\text{var-order-rel})) \ xs) \rangle$  **and**  
 $\langle \forall x \in \text{mononoms } (xs \ @ \ xs'). \text{distinct } x \rangle$   
**shows**  $\langle \text{monadic-nfoldli } xs \ (\lambda\cdot. \text{RETURN True}) \ (\lambda(a, n) \ b. \text{do } \{a \leftarrow \text{sort-coeff } a; \text{RETURN } ((a, n) \# b)\}) \ xs' \leq$   
 $\Downarrow \text{Id } (\text{SPEC}(\lambda ys. \text{map } (\lambda(a,b). (\text{mset } a, b)) \ (\text{rev } xs \ @ \ xs') = \text{map } (\lambda(a,b). (\text{mset } a, b)) \ (ys) \wedge$   
 $(\forall xs \in \text{mononoms } ys. \text{sorted-wrt } (\text{rel2p } (\text{var-order-rel})) \ xs))) \rangle$   
**using** *assms*  
**unfolding** *sort-all-coeffs-def sort-coeff-def*  
**apply** (*induction xs arbitrary: xs'*)  
**subgoal**  
**using** *assms*  
**by** *auto*  
**subgoal premises** *p for a xs*  
**using** *p(2-)*  
**apply** (*cases a, simp only: monadic-nfoldli-simp bind-to-let-conv Let-def if-True Refine-Basic.nres-monad3*  
*intro-spec-refine-iff prod.case*)  
**apply** (*auto 5 3 simp: intro-spec-refine-iff image-Un*  
*dest: same-mset-distinct-iff*  
*intro!: p(1)[THEN order-trans] distinct-var-order-Id-var-order*)  
**apply** (*metis UnCI fst-eqD rel2p-def sorted-wrt-mono-rel*)  
**done**  
**done**

**definition** *shuffle-coefficients where*

$\langle \text{shuffle-coefficients } xs = (\text{SPEC}(\lambda ys. \text{map } (\lambda(a,b). (\text{mset } a, b)) \ (\text{rev } xs) = \text{map } (\lambda(a,b). (\text{mset } a, b))$   
 $ys \wedge$   
 $(\forall xs \in \text{mononoms } ys. \text{sorted-wrt } (\text{rel2p } (\text{var-order-rel})) \ xs))) \rangle$

**lemma** *sort-all-coeffs*:

$\langle \forall x \in \text{mononoms } xs. \text{distinct } x \implies$   
 $\text{sort-all-coeffs } xs \leq \Downarrow \text{Id } (\text{shuffle-coefficients } xs) \rangle$   
**unfolding** *sort-all-coeffs-def shuffle-coefficients-def*  
**by** (*rule sort-all-coeffs-gen[THEN order-trans]*)  
*auto*

**lemma** *unsorted-term-poly-list-rel-mset*:

$\langle (ys, aa) \in \text{unsorted-term-poly-list-rel} \implies \text{mset } ys = aa \rangle$   
**by** (*auto simp: unsorted-term-poly-list-rel-def*)

**lemma** *RETURN-map-alt-def*:

$\langle \text{RETURN } o \ (\text{map } f) =$   
 $\text{RECT}_T \ (\lambda g \ xs.$   
 $\text{case } xs \text{ of}$   
 $\square \Rightarrow \text{RETURN } \square$   
 $\mid x \# xs \Rightarrow \text{do } \{xs \leftarrow g \ xs; \text{RETURN } (f \ x \ \# \ xs)\} \rangle$   
**unfolding** *comp-def*  
**apply** (*subst eq-commute*)  
**apply** (*intro ext*)  
**apply** (*induct-tac x*)  
**subgoal**  
**apply** (*subst RECT-unfold*)

```

  apply refine-mono
  apply auto
  done
subgoal
  apply (subst RECT-unfold)
  apply refine-mono
  apply auto
  done
done

```

**lemma** *fully-unsorted-poly-rel-Cons-iff*:

```

  ⟨⟨(ys, n) # p, a⟩ ∈ fully-unsorted-poly-rel ⟷
    (p, remove1-mset (mset ys, n) a) ∈ fully-unsorted-poly-rel ∧
    (mset ys, n) ∈# a ∧ distinct ys⟩
  apply (auto simp: poly-list-rel-def list-rel-split-right-iff list-mset-rel-def br-def
    unsorted-term-poly-list-rel-def
    nonzero-coeffs-def fully-unsorted-poly-list-rel-def dest!: multi-member-split)
  apply blast
  apply (rule-tac b = ⟨(mset ys, n) # y⟩ in relcompI)
  apply auto
  done

```

**lemma** *map-mset-unsorted-term-poly-list-rel*:

```

  ⟨⟨∧ a. a ∈ mononoms s ⟹ distinct a ⟹ ∀ x ∈ mononoms s. distinct x ⟹
    (∀ xs ∈ mononoms s. sorted-wrt (rel2p (Id ∪ var-order-rel)) xs) ⟹
    (s, map (λ(a, y). (mset a, y)) s)
    ∈ (term-poly-list-rel ×r int-rel) list-rel⟩
  by (induction s) (auto simp: term-poly-list-rel-def
    distinct-var-order-Id-var-order)

```

**lemma** *list-rel-unsorted-term-poly-list-relD*:

```

  ⟨(p, y) ∈ (unsorted-term-poly-list-rel ×r int-rel) list-rel ⟹
    mset y = (λ(a, y). (mset a, y)) ' # mset p ∧ (∀ x ∈ mononoms p. distinct x)⟩
  by (induction p arbitrary: y)
    (auto simp: list-rel-split-right-iff
      unsorted-term-poly-list-rel-def)

```

**lemma** *shuffle-terms-distinct-iff*:

```

  assumes ⟨map (λ(a, y). (mset a, y)) p = map (λ(a, y). (mset a, y)) s⟩
  shows ⟨(∀ x ∈ set p. distinct (fst x)) ⟷ (∀ x ∈ set s. distinct (fst x))⟩

```

**proof** –

```

  have ⟨∀ x ∈ set s. distinct (fst x)⟩
  if m: ⟨map (λ(a, y). (mset a, y)) p = map (λ(a, y). (mset a, y)) s⟩ and
    dist: ⟨∀ x ∈ set p. distinct (fst x)⟩
  for s p
  proof standard+
  fix x
  assume x: ⟨x ∈ set s⟩
  obtain v n where [simp]: ⟨x = (v, n)⟩ by (cases x)
  then have ⟨(mset v, n) ∈ set (map (λ(a, y). (mset a, y)) p)⟩
    using x unfolding m by auto
  then obtain v' where
    ⟨(v', n) ∈ set p⟩ and
    ⟨mset v' = mset v⟩

```

```

    by (auto simp: image-iff)
  then show ⟨distinct (fst x)⟩
    using dist by (metis ⟨x = (v, n)⟩ distinct-mset-mset-distinct fst-conv)
qed
from this[of p s] this[of s p]
show ⟨?thesis⟩
  unfolding assms
  by blast
qed

```

```

lemma
  ⟨(p, y) ∈ ⟨unsorted-term-poly-list-rel ×r int-rel⟩list-rel ⟹
    (a, b) ∈ set p ⟹ distinct a⟩
  using list-rel-unsorted-term-poly-list-relD by fastforce

```

```

lemma sort-all-coeffs-unsorted-poly-rel-with0:
  assumes ⟨(p, p') ∈ fully-unsorted-poly-rel⟩
  shows ⟨sort-all-coeffs p ≤ ↓ (unsorted-poly-rel-with0) (RETURN p')⟩
proof -
  have ⟨(map (λ(a, y). (mset a, y)) (rev p)) =
    map (λ(a, y). (mset a, y)) s ⟷
    (map (λ(a, y). (mset a, y)) p) =
    map (λ(a, y). (mset a, y)) (rev s)⟩ for s
  apply (auto simp flip: rev-map)
  by (metis rev-rev-ident)
show ?thesis
apply (rule sort-all-coeffs[THEN order-trans])
using assms
apply (auto simp: shuffle-coefficients-def poly-list-rel-def
  RETURN-def fully-unsorted-poly-list-rel-def list-mset-rel-def
  br-def dest: list-rel-unsorted-term-poly-list-relD
  intro!: RES-refine)
apply (rule-tac b= ⟨map (λ(a, y). (mset a, y)) (rev p)⟩ in relcompI)
apply (auto dest: list-rel-unsorted-term-poly-list-relD
  simp:)
apply (auto simp: mset-map rev-map
  dest!: list-rel-unsorted-term-poly-list-relD
  intro!: map-mset-unsorted-term-poly-list-rel)
apply (force dest: shuffle-terms-distinct-iff[THEN iffD1])
apply (force dest: shuffle-terms-distinct-iff[THEN iffD1])
apply (metis Un-iff fst-conv rel2p-def sorted-wrt-mono-rel)
by (metis mset-map mset-rev)
qed

```

```

lemma sort-poly-spec-id':
  assumes ⟨(p, p') ∈ unsorted-poly-rel-with0⟩
  shows ⟨sort-poly-spec p ≤ ↓ (sorted-repeat-poly-rel-with0) (RETURN p')⟩
proof -
  obtain y where
    py: ⟨(p, y) ∈ ⟨term-poly-list-rel ×r int-rel⟩list-rel⟩ and
    p'-y: ⟨p' = mset y⟩
  using assms
  unfolding fully-unsorted-poly-list-rel-def poly-list-rel-def sorted-poly-list-rel-wrt-def
  by (auto simp: list-mset-rel-def br-def)
then have [simp]: ⟨length y = length p⟩

```

```

  by (auto simp: list-rel-def list-all2-conv-all-nth)
have H:  $\langle (x, p') \in \langle \text{term-poly-list-rel} \times_r \text{int-rel} \rangle \text{list-rel } O \text{ list-mset-rel} \rangle$ 
   $\in \langle \text{term-poly-list-rel} \times_r \text{int-rel} \rangle \text{list-rel } O \text{ list-mset-rel}$ 
  if px:  $\langle \text{mset } p = \text{mset } x \rangle$  and  $\langle \text{sorted-wrt } (\text{rel2p } (Id \cup \text{lexord var-order-rel})) (\text{map fst } x) \rangle$ 
  for x ::  $\langle \text{list-polynom} \rangle$ 
proof -
  obtain f where
    f:  $\langle \text{bij-betw } f \{0..<\text{length } x\} \{0..<\text{length } p\} \rangle$  and
    [simp]:  $\langle \bigwedge i. i < \text{length } x \implies x ! i = p ! (f i) \rangle$ 
    using px apply - apply (subst (asm)(2) eq-commute) unfolding mset-eq-perm
    by (auto dest!: permutation-Ex-bij)
  let ?y =  $\langle \text{map } (\lambda i. y ! f i) [0..<\text{length } x] \rangle$ 
  have  $\langle i < \text{length } y \implies (p ! f i, y ! f i) \in \text{term-poly-list-rel} \times_r \text{int-rel} \rangle$  for i
    using list-all2-nthD[of - p y]
     $\langle f i, OF py[\text{unfolded list-rel-def mem-Collect-eq prod.case}]$ 
     $\text{mset-eq-length}[OF px] f$ 
    by (auto simp: list-rel-def list-all2-conv-all-nth bij-betw-def)
  then have  $\langle (x, ?y) \in \langle \text{term-poly-list-rel} \times_r \text{int-rel} \rangle \text{list-rel} \rangle$  and
    xy:  $\langle \text{length } x = \text{length } y \rangle$ 
    using py list-all2-nthD[of  $\langle \text{rel2p } (\text{term-poly-list-rel} \times_r \text{int-rel}) \rangle p y$ ]
     $\langle f i \rangle$  for i, [simplified] mset-eq-length[OF px]
    by (auto simp: list-rel-def list-all2-conv-all-nth)
  moreover {
    have f:  $\langle \text{mset-set } \{0..<\text{length } x\} = f \# \text{mset-set } \{0..<\text{length } x\} \rangle$ 
      using f mset-eq-length[OF px]
      by (auto simp: bij-betw-def lessThan-atLeast0 image-mset-mset-set)
    have  $\langle \text{mset } y = \{ \#y ! f x. x \in \# \text{mset-set } \{0..<\text{length } x\} \# \} \rangle$ 
      by (subst drop-0[symmetric], subst mset-drop-upto, subst xy[symmetric], subst f)
      auto
    then have  $\langle (?y, p') \in \text{list-mset-rel} \rangle$ 
      by (auto simp: list-mset-rel-def br-def p'-y)
  }
  ultimately show ?thesis
    by (auto intro!: relcompI[of - ?y])
qed
show ?thesis
  unfolding sort-poly-spec-def poly-list-rel-def sorted-repeat-poly-list-rel-with0-wrt-def
  by refine-rcg (auto intro: H)
qed

```

```

fun merge-coeffs0 ::  $\langle \text{list-polynom} \Rightarrow \text{list-polynom} \rangle$  where
   $\langle \text{merge-coeffs0 } [] = [] \rangle$  |
   $\langle \text{merge-coeffs0 } [(x, n)] = (\text{if } n = 0 \text{ then } [] \text{ else } [(x, n)]) \rangle$  |
   $\langle \text{merge-coeffs0 } ((x, n) \# (y, m) \# p) =$ 
     $(\text{if } x = y$ 
      then  $\text{if } n + m \neq 0 \text{ then merge-coeffs0 } ((x, n + m) \# p) \text{ else merge-coeffs0 } p$ 
      else  $\text{if } n = 0 \text{ then merge-coeffs0 } ((y, m) \# p)$ 
      else  $(x, n) \# \text{merge-coeffs0 } ((y, m) \# p) \rangle$ 

```

```

lemma sorted-repeat-poly-list-rel-with0-wrt-ConsD:
   $\langle ((y, n) \# p, a) \in \text{sorted-repeat-poly-list-rel-with0-wrt } S \text{ term-poly-list-rel} \implies$ 
     $(p, \text{remove1-mset } (\text{mset } y, n) a) \in \text{sorted-repeat-poly-list-rel-with0-wrt } S \text{ term-poly-list-rel} \wedge$ 
     $(\text{mset } y, n) \in \# a \wedge (\forall x \in \text{set } p. S y (fst x)) \wedge \text{sorted-wrt } (\text{rel2p var-order-rel}) y \wedge$ 

```

*distinct ys*  
**unfolding** *sorted-repeat-poly-list-rel-with0-wrt-def prod.case mem-Collect-eq*  
*list-rel-def*  
**apply** (*clarsimp*)  
**apply** (*subst (asm) list.rel-sel*)  
**apply** (*intro conjI*)  
**apply** (*rule-tac b = <tl y> in relcompI*)  
**apply** (*auto simp: sorted-poly-list-rel-wrt-def list-mset-rel-def br-def*)  
**apply** (*case-tac <lead-coeff y>; case-tac y*)  
**apply** (*auto simp: term-poly-list-rel-def*)  
**apply** (*case-tac <lead-coeff y>; case-tac y*)  
**apply** (*auto simp: term-poly-list-rel-def*)  
**apply** (*case-tac <lead-coeff y>; case-tac y*)  
**apply** (*auto simp: term-poly-list-rel-def*)  
**apply** (*case-tac <lead-coeff y>; case-tac y*)  
**apply** (*auto simp: term-poly-list-rel-def*)  
**done**

**lemma** *sorted-repeat-poly-list-rel-with0-wrtl-Cons-iff:*

$\langle (ys, n) \# p, a \rangle \in \text{sorted-repeat-poly-list-rel-with0-wrt } S \text{ term-poly-list-rel} \longleftrightarrow$   
 $(p, \text{remove1-mset } (\text{mset } ys, n) a) \in \text{sorted-repeat-poly-list-rel-with0-wrt } S \text{ term-poly-list-rel} \wedge$   
 $(\text{mset } ys, n) \in \# a \wedge (\forall x \in \text{set } p. S \text{ } ys \text{ } (\text{fst } x)) \wedge \text{sorted-wrt } (\text{rel2p var-order-rel}) \text{ } ys \wedge$   
*distinct ys*

**apply** (*rule iffI*)

**subgoal**

**by** (*auto dest!: sorted-repeat-poly-list-rel-with0-wrt-ConsD*)

**subgoal**

**unfolding** *sorted-poly-list-rel-wrt-def prod.case mem-Collect-eq*

*list-rel-def sorted-repeat-poly-list-rel-with0-wrt-def*

**apply** (*clarsimp*)

**apply** (*rule-tac b = <(mset ys, n) # y> in relcompI*)

**by** (*auto simp: sorted-poly-list-rel-wrt-def list-mset-rel-def br-def*  
*term-poly-list-rel-def add-mset-eq-add-mset eq-commute[of - (mset -)]*  
*nonzero-coeffs-def*  
*dest!: multi-member-split*)

**done**

**lemma** *fst-normalize0-polynom-subsetD:*

$\langle (a, b) \in \text{set } (\text{merge-coeffs0 } p) \implies a \in \text{monoms } p$

**apply** (*induction p rule: merge-coeffs0.induct*)

**subgoal**

**by** *auto*

**subgoal**

**by** (*auto split: if-splits*)

**subgoal**

**by** (*auto split: if-splits*)

**done**

**lemma** *in-set-merge-coeffs0D:*

$\langle (a, b) \in \text{set } (\text{merge-coeffs0 } p) \implies \exists b. (a, b) \in \text{set } p$

**by** (*auto dest!: fst-normalize0-polynom-subsetD*)

**lemma** *merge-coeffs0-is-normalize-poly-p:*

$\langle (xs, ys) \in \text{sorted-repeat-poly-rel-with0} \implies \exists r. (\text{merge-coeffs0 } xs, r) \in \text{sorted-poly-rel} \wedge \text{normalize-poly-p}^{**}$

*ys r*)

```

apply (induction xs arbitrary: ys rule: merge-coeffs0.induct)
subgoal by (auto simp: sorted-repeat-poly-list-rel-wrt-def sorted-poly-list-rel-wrt-def
  sorted-repeat-poly-list-rel-with0-wrt-def list-mset-rel-def br-def)
subgoal for xs n ys
  by (force simp: sorted-repeat-poly-list-rel-wrt-def sorted-poly-list-rel-wrt-def
    sorted-repeat-poly-list-rel-with0-wrt-def list-mset-rel-def br-def
    list-rel-split-right-iff)
subgoal premises p for xs n ys m p ysa
  apply (cases <xs = ys>, cases <m+n ≠ 0>)
  subgoal
    using p(1)[of <add-mset (mset ys, m+n) ysa - {#(mset ys, m), (mset ys, n)#}>] p(5-)
    apply (auto simp: sorted-repeat-poly-list-rel-with0-wrtl-Cons-iff ac-simps add-mset-commute
      remove1-mset-add-mset-If nonzero-coeffs-diff sorted-repeat-poly-list-rel-Cons-iff)
    apply (auto intro: normalize-poly-p.intros add-mset-commute add-mset-commute
      converse-rtranclp-into-rtranclp dest!: multi-member-split
      simp del: normalize-poly-p.merge-dup-coeff)
    apply (rule-tac x = <r> in exI)
    using normalize-poly-p.merge-dup-coeff[of <ysa - {#(mset ys, m), (mset ys, n)#}> <ysa - {#(mset
ys, m), (mset ys, n)#}> <mset ys> m n]
    apply (auto intro: normalize-poly-p.intros add-mset-commute add-mset-commute
      converse-rtranclp-into-rtranclp dest!: multi-member-split
      simp del: normalize-poly-p.merge-dup-coeff)
    by (metis add-mset-commute converse-rtranclp-into-rtranclp)
  subgoal
    using p(2)[of <ysa - {#(mset ys, m), (mset ys, n)#}>] p(5-)
    apply (auto simp: sorted-repeat-poly-list-rel-with0-wrtl-Cons-iff ac-simps add-mset-commute
      remove1-mset-add-mset-If nonzero-coeffs-diff sorted-repeat-poly-list-rel-Cons-iff)
    apply (rule-tac x = <r> in exI)
    using normalize-poly-p.rem-0-coeff[of <add-mset (mset ys, m+n) ysa - {#(mset ys, m), (mset
ys, n)#}> <add-mset (mset ys, m+n) ysa - {#(mset ys, m), (mset ys, n)#}> <mset ys>]
    using normalize-poly-p.merge-dup-coeff[of <ysa - {#(mset ys, m), (mset ys, n)#}> <ysa - {#(mset
ys, m), (mset ys, n)#}> <mset ys> m n]
    apply (auto intro: normalize-poly-p.intros add-mset-commute add-mset-commute converse-rtranclp-into-rtranclp
      dest!: multi-member-split
      simp del: normalize-poly-p.rem-0-coeff)
    by (metis add-mset-commute converse-rtranclp-into-rtranclp normalize-poly-p.simps)
  apply (cases <n = 0>)
  subgoal
    using p(3)[of <add-mset (mset ys, m) ysa - {#(mset xs, n), (mset ys, m)#}>] p(4-)
    apply (auto simp: sorted-repeat-poly-list-rel-with0-wrtl-Cons-iff ac-simps add-mset-commute
      remove1-mset-add-mset-If sorted-repeat-poly-list-rel-Cons-iff)
    apply (rule-tac x = <r> in exI)
    apply (auto dest!: in-set-merge-coeffsD)
    apply (auto intro: normalize-poly-p.intros rtranclp-normalize-poly-add-mset
      simp: rel2p-def var-order-rel-def sorted-poly-list-rel-Cons-iff
      dest!: multi-member-split
      dest: sorted-poly-list-rel-nonzeroD)
    by (metis converse-rtranclp-into-rtranclp normalize-poly-p.simps)
  subgoal
    using p(4)[of <add-mset (mset ys, m) ysa - {#(mset xs, n), (mset ys, m)#}>] p(5-)
    apply (auto simp: sorted-repeat-poly-list-rel-with0-wrtl-Cons-iff ac-simps add-mset-commute
      remove1-mset-add-mset-If sorted-repeat-poly-list-rel-Cons-iff)
    apply (rule-tac x = <add-mset (mset xs, n) r> in exI)
    apply (auto dest!: in-set-merge-coeffs0D)

```

```

apply (auto intro: normalize-poly-p.intros rtrancp-normalize-poly-add-mset
  simp: rel2p-def var-order-rel-def sorted-poly-list-rel-Cons-iff
  dest!: multi-member-split
  dest: sorted-poly-list-rel-nonzeroD)
using in-set-merge-coeffs0D total-on-lexord-less-than-char-linear apply fastforce
using in-set-merge-coeffs0D total-on-lexord-less-than-char-linear apply fastforce
done
done
done

```

**definition** *full-normalize-poly* **where**

```

⟨full-normalize-poly p = do {
  p ← sort-all-coeffs p;
  p ← sort-poly-spec p;
  RETURN (merge-coeffs0 p)
}⟩

```

**fun** *sorted-remdups* **where**

```

⟨sorted-remdups (x # y # zs) =
  (if x = y then sorted-remdups (y # zs) else x # sorted-remdups (y # zs))⟩ |
⟨sorted-remdups zs = zs⟩

```

**lemma** *set-sorted-remdups*[simp]:

```

⟨set (sorted-remdups xs) = set xs⟩
by (induction xs rule: sorted-remdups.induct)
  auto

```

**lemma** *distinct-sorted-remdups*:

```

⟨sorted-wrt R xs ⟹ transp R ⟹ Restricted-Predicates.total-on R UNIV ⟹
  antisyp R ⟹ distinct (sorted-remdups xs)⟩
by (induction xs rule: sorted-remdups.induct)
  (auto dest: antisypD)

```

**lemma** *full-normalize-poly-normalize-poly-p*:

```

assumes ⟨(p, p') ∈ fully-unsorted-poly-rel⟩
shows ⟨full-normalize-poly p ≤ ↓ (sorted-poly-rel) (SPEC (λr. normalize-poly-p** p' r))⟩
(is ⟨?A ≤ ↓ ?R ?B⟩)

```

**proof** –

```

have 1: ⟨?B = do {
  p' ← RETURN p';
  p' ← RETURN p';
  SPEC (λr. normalize-poly-p** p' r)
}⟩
by auto
have [refine0]: ⟨sort-all-coeffs p ≤ SPEC(λp. (p, p') ∈ unsorted-poly-rel-with0)⟩
by (rule sort-all-coeffs-unsorted-poly-rel-with0[OF assms, THEN order-trans])
  (auto simp: conc-fun-RES RETURN-def)
have [refine0]: ⟨sort-poly-spec p ≤ SPEC (λc. (c, p') ∈ sorted-repeat-poly-rel-with0)⟩
if ⟨(p, p') ∈ unsorted-poly-rel-with0⟩
for p p'
by (rule sort-poly-spec-id'[THEN order-trans, OF that])
  (auto simp: conc-fun-RES RETURN-def)
show ?thesis
apply (subst 1)
unfolding full-normalize-poly-def

```

by (refine-rcg)  
 (auto intro!: RES-refine  
 dest!: merge-coeffs0-is-normalize-poly-p  
 simp: RETURN-def)  
 qed

**definition** *mult-poly-full* ::  $\langle \cdot \rangle$  **where**

$\langle \text{mult-poly-full } p \ q = \text{do } \{$   
 let  $pq = \text{mult-poly-raw } p \ q;$   
 normalize-poly  $pq$   
 $\} \rangle$

**lemma** *normalize-poly-normalize-poly-p*:

**assumes**  $\langle (p, p') \in \text{unsorted-poly-rel} \rangle$

**shows**  $\langle \text{normalize-poly } p \leq \Downarrow (\text{sorted-poly-rel}) (\text{SPEC } (\lambda r. \text{normalize-poly-p}^{**} p' r)) \rangle$

**proof** –

**have** 1:  $\langle \text{SPEC } (\lambda r. \text{normalize-poly-p}^{**} p' r) = \text{do } \{$   
 $p' \leftarrow \text{RETURN } p';$   
 $\text{SPEC } (\lambda r. \text{normalize-poly-p}^{**} p' r)$   
 $\} \rangle$

**by** *auto*

**show** *?thesis*

**unfolding** *normalize-poly-def*

**apply** (*subst* 1)

**apply** (*refine-rcg sort-poly-spec-id*[*OF* *assms*]  
*merge-coeffs-is-normalize-poly-p*)

**subgoal**

**by** (*drule merge-coeffs-is-normalize-poly-p*)

(*auto intro!: RES-refine simp: RETURN-def*)

**done**

qed

## 9.4 Multiplication and normalisation

**definition** *mult-poly-p'* ::  $\langle \cdot \rangle$  **where**

$\langle \text{mult-poly-p}' p' q' = \text{do } \{$   
 $pq \leftarrow \text{SPEC } (\lambda r. (\text{mult-poly-p } q')^{**} (p', \{\#\}) (\{\#\}, r));$   
 $\text{SPEC } (\lambda r. \text{normalize-poly-p}^{**} pq r)$   
 $\} \rangle$

**lemma** *unsorted-poly-rel-fully-unsorted-poly-rel*:

$\langle \text{unsorted-poly-rel} \subseteq \text{fully-unsorted-poly-rel} \rangle$

**proof** –

**have**  $\langle \text{term-poly-list-rel} \times_r \text{int-rel} \subseteq \text{unsorted-term-poly-list-rel} \times_r \text{int-rel} \rangle$

**by** (*auto simp: unsorted-term-poly-list-rel-def term-poly-list-rel-def*)

**from** *list-rel-mono*[*OF* *this*]

**show** *?thesis*

**unfolding** *poly-list-rel-def fully-unsorted-poly-list-rel-def*

**by** (*auto simp:*)

qed

**lemma** *mult-poly-full-mult-poly-p'*:

**assumes**  $\langle (p, p') \in \text{sorted-poly-rel} \rangle \langle (q, q') \in \text{sorted-poly-rel} \rangle$

**shows**  $\langle \text{mult-poly-full } p \ q \leq \Downarrow (\text{sorted-poly-rel}) (\text{mult-poly-p}' p' q') \rangle$

**unfolding** *mult-poly-full-def mult-poly-p'-def*

**apply** (*refine-rcg full-normalize-poly-normalize-poly-p*)



```

  normalize-poly-normalize-poly-p)
apply (subst RETURN-RES-refine-iff)
apply (subst Bex-def)
apply (subst mem-Collect-eq)
apply (subst conj-commute)
apply (rule mult-poly-raw-mult-poly-p[OF assms(1,2)])
subgoal
  by blast
done

```

**definition** *add-poly-spec* ::  $\langle \cdot \rangle$  **where**  
 $\langle \text{add-poly-spec } p \ q = \text{SPEC } (\lambda r. \ p + q - r \in \text{ideal polynom-bool}) \rangle$

**definition** *add-poly-p'* ::  $\langle \cdot \rangle$  **where**  
 $\langle \text{add-poly-p'} \ p \ q = \text{SPEC}(\lambda r. \ \text{add-poly-p}^{**} \ (p, q, \{\#\}) \ (\{\#\}, \{\#\}, r)) \rangle$

**lemma** *add-poly-l-add-poly-p'*:  
**assumes**  $\langle (p, p') \in \text{sorted-poly-rel} \rangle \langle (q, q') \in \text{sorted-poly-rel} \rangle$   
**shows**  $\langle \text{add-poly-l } (p, q) \leq \Downarrow (\text{sorted-poly-rel}) (\text{add-poly-p'} \ p' \ q') \rangle$   
**unfolding** *add-poly-p'-def*  
**apply** (refine-rcg *add-poly-l-spec[THEN fref-to-Down-curry-right, THEN order-trans, of - p' q']*)  
**subgoal by** *auto*  
**subgoal using** *assms* **by** *auto*  
**subgoal**  
**by** *auto*  
**done**

## 9.5 Correctness

**context** *poly-embed*  
**begin**

**definition** *mset-poly-rel* **where**  
 $\langle \text{mset-poly-rel} = \{(a, b). \ b = \text{polynom-of-mset } a\} \rangle$

**definition** *var-rel* **where**  
 $\langle \text{var-rel} = \text{br } \varphi \ (\lambda \cdot. \ \text{True}) \rangle$

**lemma** *normalize-poly-p-normalize-poly-spec*:  
 $\langle (p, p') \in \text{mset-poly-rel} \implies \text{SPEC } (\lambda r. \ \text{normalize-poly-p}^{**} \ p \ r) \leq \Downarrow \text{mset-poly-rel} \ (\text{normalize-poly-spec } p') \rangle$   
**by** (*auto simp: mset-poly-rel-def rtranclp-normalize-poly-p-poly-of-mset ideal.span-zero normalize-poly-spec-def intro!: RES-refine*)

**lemma** *mult-poly-p'-mult-poly-spec*:  
 $\langle (p, p') \in \text{mset-poly-rel} \implies (q, q') \in \text{mset-poly-rel} \implies \text{mult-poly-p'} \ p \ q \leq \Downarrow \text{mset-poly-rel} \ (\text{mult-poly-spec } p' \ q') \rangle$   
**unfolding** *mult-poly-p'-def mult-poly-spec-def*  
**apply** *refine-rcg*  
**apply** (*auto simp: mset-poly-rel-def dest!: rtranclp-mult-poly-p-mult-ideal-final*)  
**apply** (*intro RES-refine*)  
**apply** *auto*  
**by** (*smt cancel-comm-monoid-add-class.diff-cancel diff-diff-add group-eq-aux ideal.span-diff rtranclp-normalize-poly-p-poly-of-mset*)

```

lemma add-poly-p'-add-poly-spec:
   $\langle (p, p') \in \text{mset-poly-rel} \implies (q, q') \in \text{mset-poly-rel} \implies$ 
   $\text{add-poly-p}' p q \leq \Downarrow \text{mset-poly-rel} (\text{add-poly-spec } p' q') \rangle$ 
  unfolding add-poly-p'-def add-poly-spec-def
  apply (auto simp: mset-poly-rel-def dest!: rtranclp-add-poly-p-polynom-of-mset-full)
  apply (intro RES-refine)
  apply (auto simp: rtranclp-add-poly-p-polynom-of-mset-full ideal.span-zero)
  done

```

**end**

```

definition weak-equality-l ::  $\langle \text{llist-polynom} \Rightarrow \text{llist-polynom} \Rightarrow \text{bool nres} \rangle$  where
   $\langle \text{weak-equality-l } p q = \text{RETURN } (p = q) \rangle$ 

```

```

definition weak-equality ::  $\langle \text{int mpoly} \Rightarrow \text{int mpoly} \Rightarrow \text{bool nres} \rangle$  where
   $\langle \text{weak-equality } p q = \text{SPEC } (\lambda r. r \longrightarrow p = q) \rangle$ 

```

```

definition weak-equality-spec ::  $\langle \text{mset-polynom} \Rightarrow \text{mset-polynom} \Rightarrow \text{bool nres} \rangle$  where
   $\langle \text{weak-equality-spec } p q = \text{SPEC } (\lambda r. r \longrightarrow p = q) \rangle$ 

```

```

lemma term-poly-list-rel-same-rightD:
   $\langle (a, aa) \in \text{term-poly-list-rel} \implies (a, ab) \in \text{term-poly-list-rel} \implies aa = ab \rangle$ 
  by (auto simp: term-poly-list-rel-def)

```

```

lemma list-rel-term-poly-list-rel-same-rightD:
   $\langle (xa, y) \in \langle \text{term-poly-list-rel} \times_r \text{int-rel} \rangle \text{list-rel} \implies$ 
   $(xa, ya) \in \langle \text{term-poly-list-rel} \times_r \text{int-rel} \rangle \text{list-rel} \implies$ 
   $y = ya \rangle$ 
  by (induction xa arbitrary: y ya)
  (auto simp: list-rel-split-right-iff
  dest: term-poly-list-rel-same-rightD)

```

```

lemma weak-equality-l-weak-equality-spec:
   $\langle (\text{uncurry weak-equality-l}, \text{uncurry weak-equality-spec}) \in$ 
   $\text{sorted-poly-rel} \times_r \text{sorted-poly-rel} \rightarrow_f \langle \text{bool-rel} \rangle \text{nres-rel} \rangle$ 
  by (intro frefl nres-relI)
  (auto simp: weak-equality-l-def weak-equality-spec-def
  sorted-poly-list-rel-wrt-def list-mset-rel-def br-def
  dest: list-rel-term-poly-list-rel-same-rightD)

```

**end**

**theory** *PAC-Checker*

```

imports PAC-Polynoms-Operations
  PAC-Checker-Specification
  PAC-Map-Rel
  Show.Show
  Show.Show-Instances

```

**begin**

## 10 Executable Checker

In this layer we finally refine the checker to executable code.

## 10.1 Definitions

Compared to the previous layer, we add an error message when an error is discovered. We do not attempt to prove anything on the error message (neither that there really is an error, nor that the error message is correct).

**Extended error message datatype**  $'a$  code-status =  
 $is-cfailed: CFAILED$  (the-error:  $'a$ ) |  
 $CSUCCESS$  |  
 $is-cfound: CFOUND$

In the following function, we merge errors. We will never merge an error message with an another error message; hence we do not attempt to concatenate error messages.

**fun** merge-cstatus **where**  
 $\langle merge-cstatus (CFAILED a) - = CFAILED a \rangle$  |  
 $\langle merge-cstatus - (CFAILED a) = CFAILED a \rangle$  |  
 $\langle merge-cstatus CFOUND - = CFOUND \rangle$  |  
 $\langle merge-cstatus - CFOUND = CFOUND \rangle$  |  
 $\langle merge-cstatus - - = CSUCCESS \rangle$

**definition** code-status-status-rel ::  $\langle ('a$  code-status  $\times$  status) set  $\rangle$  **where**  
 $\langle code-status-status-rel =$   
 $\{(CFOUND, FOUND), (CSUCCESS, SUCCESS)\} \cup$   
 $\{(CFAILED a, FAILED) \mid a. True\}$

**lemma** in-code-status-status-rel-iff[simp]:  
 $\langle (CFOUND, b) \in code-status-status-rel \longleftrightarrow b = FOUND \rangle$   
 $\langle (a, FOUND) \in code-status-status-rel \longleftrightarrow a = CFOUND \rangle$   
 $\langle (CSUCCESS, b) \in code-status-status-rel \longleftrightarrow b = SUCCESS \rangle$   
 $\langle (a, SUCCESS) \in code-status-status-rel \longleftrightarrow a = CSUCCESS \rangle$   
 $\langle (a, FAILED) \in code-status-status-rel \longleftrightarrow is-cfailed a \rangle$   
 $\langle (CFAILED C, b) \in code-status-status-rel \longleftrightarrow b = FAILED \rangle$   
**by** (cases a; cases b; auto simp: code-status-status-rel-def; fail)+

**Refinement relation** **fun** pac-step-rel-raw ::  $\langle ('olbl \times 'lbl)$  set  $\Rightarrow ('a \times 'b)$  set  $\Rightarrow ('c \times 'd)$  set  $\Rightarrow$   
 $('a, 'c, 'olbl)$  pac-step  $\Rightarrow ('b, 'd, 'lbl)$  pac-step  $\Rightarrow bool$  **where**  
 $\langle pac-step-rel-raw R1 R2 R3 (Add p1 p2 i r) (Add p1' p2' i' r') \longleftrightarrow$   
 $(p1, p1') \in R1 \wedge (p2, p2') \in R2 \wedge (i, i') \in R1 \wedge$   
 $(r, r') \in R2 \rangle$  |  
 $\langle pac-step-rel-raw R1 R2 R3 (Mult p1 p2 i r) (Mult p1' p2' i' r') \longleftrightarrow$   
 $(p1, p1') \in R1 \wedge (p2, p2') \in R2 \wedge (i, i') \in R1 \wedge$   
 $(r, r') \in R2 \rangle$  |  
 $\langle pac-step-rel-raw R1 R2 R3 (Del p1) (Del p1') \longleftrightarrow$   
 $(p1, p1') \in R1 \rangle$  |  
 $\langle pac-step-rel-raw R1 R2 R3 (Extension i x p1) (Extension j x' p1') \longleftrightarrow$   
 $(i, j) \in R1 \wedge (x, x') \in R3 \wedge (p1, p1') \in R2 \rangle$  |  
 $\langle pac-step-rel-raw R1 R2 R3 - - \longleftrightarrow False \rangle$

**fun** pac-step-rel-assn ::  $\langle ('olbl \Rightarrow 'lbl \Rightarrow assn) \Rightarrow ('a \Rightarrow 'b \Rightarrow assn) \Rightarrow ('c \Rightarrow 'd \Rightarrow assn) \Rightarrow ('a, 'c, 'olbl)$   
 $pac-step \Rightarrow ('b, 'd, 'lbl)$  pac-step  $\Rightarrow assn$  **where**  
 $\langle pac-step-rel-assn R1 R2 R3 (Add p1 p2 i r) (Add p1' p2' i' r') =$   
 $R1 p1 p1' * R1 p2 p2' * R1 i i' *$   
 $R2 r r' \rangle$  |  
 $\langle pac-step-rel-assn R1 R2 R3 (Mult p1 p2 i r) (Mult p1' p2' i' r') =$

```

    R1 p1 p1' * R2 p2 p2' * R1 i i' *
    R2 r r' |
  ⟨pac-step-rel-assn R1 R2 R3 (Del p1) (Del p1') =
    R1 p1 p1' |
  ⟨pac-step-rel-assn R1 R2 R3 (Extension i x p1) (Extension i' x' p1') =
    R1 i i' * R3 x x' * R2 p1 p1' |
  ⟨pac-step-rel-assn R1 R2 - - = false⟩

```

**lemma** *pac-step-rel-assn-alt-def*:

```

  ⟨pac-step-rel-assn R1 R2 R3 x y = (
    case (x, y) of
      (Add p1 p2 i r, Add p1' p2' i' r') ⇒
        R1 p1 p1' * R1 p2 p2' * R1 i i' * R2 r r'
    | (Mult p1 p2 i r, Mult p1' p2' i' r') ⇒
        R1 p1 p1' * R2 p2 p2' * R1 i i' * R2 r r'
    | (Del p1, Del p1') ⇒ R1 p1 p1'
    | (Extension i x p1, Extension i' x' p1') ⇒ R1 i i' * R3 x x' * R2 p1 p1'
    | - ⇒ false
  )⟩
  by (auto split: pac-step.splits)

```

**Addition checking** **definition** *error-msg* **where**

```

  ⟨error-msg i msg = CFAILED ("s CHECKING failed at line " @ show i @ " with error " @ msg)⟩

```

**definition** *error-msg-notin-dom-err* **where**

```

  ⟨error-msg-notin-dom-err = "notin domain"⟩

```

**definition** *error-msg-notin-dom* :: (nat ⇒ string) **where**

```

  ⟨error-msg-notin-dom i = show i @ error-msg-notin-dom-err⟩

```

**definition** *error-msg-reused-dom* **where**

```

  ⟨error-msg-reused-dom i = show i @ "already in domain"⟩

```

**definition** *error-msg-not-equal-dom* **where**

```

  ⟨error-msg-not-equal-dom p q pq r = show p @ " + " @ show q @ " = " @ show pq @ " not equal"
  @ show r⟩

```

**definition** *check-not-equal-dom-err* :: (llist-polynom ⇒ llist-polynom ⇒ llist-polynom ⇒ llist-polynom ⇒ string nres) **where**

```

  ⟨check-not-equal-dom-err p q pq r = SPEC (λ-. True)⟩

```

**definition** *vars-llist* :: (llist-polynom ⇒ string set) **where**

```

  ⟨vars-llist xs = ⋃ (set 'fst 'set xs)⟩

```

**definition** *check-addition-l* :: (· ⇒ · ⇒ string set ⇒ nat ⇒ nat ⇒ nat ⇒ llist-polynom ⇒ string code-status nres) **where**

```

  ⟨check-addition-l spec A V p q i r = do {
    let b = p ∈# dom-m A ∧ q ∈# dom-m A ∧ i ∉# dom-m A ∧ vars-llist r ⊆ V;
    if ¬b
    then RETURN (error-msg i ((if p ∉# dom-m A then error-msg-notin-dom p else []) @ (if q ∉#
    dom-m A then error-msg-notin-dom p else [])) @

```

```

    (if  $i \in \# \text{ dom-}m \ A$  then error-msg-reused-dom  $p$  else  $[]$ ))
  else do {
    ASSERT ( $p \in \# \text{ dom-}m \ A$ );
    let  $p = \text{the } (\text{fmlookup } A \ p)$ ;
    ASSERT ( $q \in \# \text{ dom-}m \ A$ );
    let  $q = \text{the } (\text{fmlookup } A \ q)$ ;
     $pq \leftarrow \text{add-poly-l } (p, q)$ ;
     $b \leftarrow \text{weak-equality-l } pq \ r$ ;
     $b' \leftarrow \text{weak-equality-l } r \ \text{spec}$ ;
    if  $b$  then (if  $b'$  then RETURN CFOUND else RETURN CSUCCESS)
    else do {
       $c \leftarrow \text{check-not-equal-dom-err } p \ q \ pq \ r$ ;
      RETURN (error-msg  $i \ c$ )
    }
  }
}
```

**Multiplication checking** definition  $\text{check-mult-l-dom-err} :: \langle \text{bool} \Rightarrow \text{nat} \Rightarrow \text{bool} \Rightarrow \text{nat} \Rightarrow \text{string nres} \rangle$  where

$\langle \text{check-mult-l-dom-err } p \ \text{notin } p \ i \ \text{already } i = \text{SPEC } (\lambda -. \text{True}) \rangle$

**definition**  $\text{check-mult-l-mult-err} :: \langle \text{l-list-polynom} \Rightarrow \text{l-list-polynom} \Rightarrow \text{l-list-polynom} \Rightarrow \text{l-list-polynom} \Rightarrow \text{string nres} \rangle$  where

$\langle \text{check-mult-l-mult-err } p \ q \ pq \ r = \text{SPEC } (\lambda -. \text{True}) \rangle$

**definition**  $\text{check-mult-l} :: \langle - \Rightarrow - \Rightarrow - \Rightarrow \text{nat} \Rightarrow \text{l-list-polynom} \Rightarrow \text{nat} \Rightarrow \text{l-list-polynom} \Rightarrow \text{string code-status nres} \rangle$  where

```

 $\langle \text{check-mult-l spec } A \ \mathcal{V} \ p \ q \ i \ r = \text{do } \{$ 
  let  $b = p \in \# \text{ dom-}m \ A \wedge i \notin \# \text{ dom-}m \ A \wedge \text{vars-l-list } q \subseteq \mathcal{V} \wedge \text{vars-l-list } r \subseteq \mathcal{V}$ ;
  if  $\neg b$ 
  then do {
     $c \leftarrow \text{check-mult-l-dom-err } (p \notin \# \text{ dom-}m \ A) \ p \ (i \in \# \text{ dom-}m \ A) \ i$ ;
    RETURN (error-msg  $i \ c$ )
  }
  else do {
    ASSERT ( $p \in \# \text{ dom-}m \ A$ );
    let  $p = \text{the } (\text{fmlookup } A \ p)$ ;
     $pq \leftarrow \text{mult-poly-full } p \ q$ ;
     $b \leftarrow \text{weak-equality-l } pq \ r$ ;
     $b' \leftarrow \text{weak-equality-l } r \ \text{spec}$ ;
    if  $b$  then (if  $b'$  then RETURN CFOUND else RETURN CSUCCESS) else do {
       $c \leftarrow \text{check-mult-l-mult-err } p \ q \ pq \ r$ ;
      RETURN (error-msg  $i \ c$ )
    }
  }
}
```

**Deletion checking** definition  $\text{check-del-l} :: \langle - \Rightarrow - \Rightarrow \text{nat} \Rightarrow \text{string code-status nres} \rangle$  where

$\langle \text{check-del-l spec } A \ p = \text{RETURN CSUCCESS} \rangle$

**Extension checking** definition  $\text{check-extension-l-dom-err} :: \langle \text{nat} \Rightarrow \text{string nres} \rangle$  where

$\langle \text{check-extension-l-dom-err } p = \text{SPEC } (\lambda -. \text{True}) \rangle$

**definition**  $\text{check-extension-l-no-new-var-err} :: \langle \text{l-list-polynom} \Rightarrow \text{string nres} \rangle$  where

$\langle \text{check-extension-l-no-new-var-err } p = \text{SPEC } (\lambda\cdot. \text{True}) \rangle$

**definition** *check-extension-l-new-var-multiple-err* ::  $\langle \text{string} \Rightarrow \text{l-list-polynom} \Rightarrow \text{string nres} \rangle$  **where**  
 $\langle \text{check-extension-l-new-var-multiple-err } v \ p = \text{SPEC } (\lambda\cdot. \text{True}) \rangle$

**definition** *check-extension-l-side-cond-err*  
 ::  $\langle \text{string} \Rightarrow \text{l-list-polynom} \Rightarrow \text{l-list-polynom} \Rightarrow \text{l-list-polynom} \Rightarrow \text{string nres} \rangle$   
**where**  
 $\langle \text{check-extension-l-side-cond-err } v \ p \ p' \ q = \text{SPEC } (\lambda\cdot. \text{True}) \rangle$

**definition** *check-extension-l*  
 ::  $\langle \cdot \Rightarrow \cdot \Rightarrow \text{string set} \Rightarrow \text{nat} \Rightarrow \text{string} \Rightarrow \text{l-list-polynom} \Rightarrow (\text{string code-status}) \text{ nres} \rangle$   
**where**  
 $\langle \text{check-extension-l spec } A \ \mathcal{V} \ i \ v \ p = \text{do } \{$   
 let  $b = i \notin \# \text{ dom-m } A \wedge v \notin \mathcal{V} \wedge ([v], -1) \in \text{set } p;$   
 if  $\neg b$   
 then do {  
    $c \leftarrow \text{check-extension-l-dom-err } i;$   
    $\text{RETURN } (\text{error-msg } i \ c)$   
 } else do {  
   let  $p' = \text{remove1 } ([v], -1) \ p;$   
   let  $b = \text{vars-llist } p' \subseteq \mathcal{V};$   
   if  $\neg b$   
   then do {  
      $c \leftarrow \text{check-extension-l-new-var-multiple-err } v \ p';$   
      $\text{RETURN } (\text{error-msg } i \ c)$   
   }  
   else do {  
      $p2 \leftarrow \text{mult-poly-full } p' \ p';$   
     let  $p' = \text{map } (\lambda(a,b). (a, -b)) \ p';$   
      $q \leftarrow \text{add-poly-l } (p2, p');$   
      $eq \leftarrow \text{weak-equality-l } q \ [];$   
     if  $eq$  then do {  
        $\text{RETURN } (\text{CSUCCESS})$   
     }  
     else do {  
        $c \leftarrow \text{check-extension-l-side-cond-err } v \ p \ p' \ q;$   
        $\text{RETURN } (\text{error-msg } i \ c)$   
     }  
   }  
 }  
 $\rangle$

**lemma** *check-extension-alt-def*:  
 $\langle \text{check-extension } A \ \mathcal{V} \ i \ v \ p \geq \text{do } \{$   
 $b \leftarrow \text{SPEC}(\lambda b. b \longrightarrow i \notin \# \text{ dom-m } A \wedge v \notin \mathcal{V});$   
 if  $\neg b$   
 then  $\text{RETURN } (\text{False})$   
 else do {  
    $p' \leftarrow \text{RETURN } (p + \text{Var } v);$   
    $b \leftarrow \text{SPEC}(\lambda b. b \longrightarrow \text{vars } p' \subseteq \mathcal{V});$   
   if  $\neg b$   
   then  $\text{RETURN } (\text{False})$   
   else do {  
      $pq \leftarrow \text{mult-poly-spec } p' \ p';$   
 }  
 $\rangle$

```

    let  $p' = -p'$ ;
     $p \leftarrow \text{add-poly-spec } pq \ p'$ ;
     $eq \leftarrow \text{weak-equality } p \ 0$ ;
    if  $eq$  then RETURN(True)
    else RETURN(False)
  }
}
}
}
proof -
  have [intro]:  $\langle ab \notin \mathcal{V} \implies$ 
    vars  $ba \subseteq \mathcal{V} \implies$ 
    MPoly-Type.coeff ( $ba + \text{Var } ab$ ) ( $\text{monomial } (\text{Suc } 0) \ ab$ ) = 1  $\rangle$  for  $ab \ ba$ 
  apply (auto simp flip: coeff-add simp: not-in-vars-coeff0
    Var.abs-eq Var0-def)
  apply (subst not-in-vars-coeff0)
  apply auto
  by (metis MPoly-Type.coeff-def lookup-single-eq monom.abs-eq monom.rep-eq)
  have [simp]:  $\langle \text{MPoly-Type.coeff } p \ (\text{monomial } (\text{Suc } 0) \ ab) = -1 \rangle$ 
  if  $\langle \text{vars } (p + \text{Var } ab) \subseteq \mathcal{V} \rangle$ 
  if  $\langle ab \notin \mathcal{V} \rangle$ 
  for  $ab$ 
  proof -
    define  $q$  where  $\langle q \equiv p + \text{Var } ab \rangle$ 
    then have  $p$ :  $\langle p = q - \text{Var } ab \rangle$ 
    by auto
    show ?thesis
    unfolding  $p$ 
    apply (auto simp flip: coeff-minus simp: not-in-vars-coeff0
      Var.abs-eq Var0-def)
    apply (subst not-in-vars-coeff0)
    using that unfolding  $q$ -def[symmetric] apply auto
    by (metis MPoly-Type.coeff-def lookup-single-eq monom.abs-eq monom.rep-eq)
  qed
  have [simp]:  $\langle \text{vars } (p - \text{Var } ab) = \text{vars } (\text{Var } ab - p) \rangle$  for  $ab$ 
  using vars-uminus[of  $\langle p - \text{Var } ab \rangle$ ]
  by simp
  show ?thesis
  unfolding check-extension-def
  apply (auto 5 5 simp: check-extension-def weak-equality-def
    mult-poly-spec-def field-simps
    add-poly-spec-def power2-eq-square cong: if-cong
    intro!: intro-spec-refine[where  $R=Id$ , simplified]
    split: option.splits dest: ideal.span-add)
  done
qed

```

**lemma** RES-RES-RETURN-RES:  $\langle \text{RES } A \gg (\lambda T. \text{RES } (f \ T)) = \text{RES } (\bigcup (f \ 'A)) \rangle$   
 by (auto simp: pw-eq-iff refine-pw-simps)

**lemma** check-add-alt-def:  
 $\langle \text{check-add } A \ \mathcal{V} \ p \ q \ i \ r \geq$   
 do {  
 $b \leftarrow \text{SPEC}(\lambda b. b \longrightarrow p \in \# \text{ dom-}m \ A \wedge q \in \# \text{ dom-}m \ A \wedge i \notin \# \text{ dom-}m \ A \wedge \text{vars } r \subseteq \mathcal{V});$

```

  if  $\neg b$ 
  then RETURN False
  else do {
    ASSERT ( $p \in \# \text{ dom-}m \ A$ );
    let  $p = \text{the } (\text{fmlookup } A \ p)$ ;
    ASSERT ( $q \in \# \text{ dom-}m \ A$ );
    let  $q = \text{the } (\text{fmlookup } A \ q)$ ;
     $pq \leftarrow \text{add-poly-spec } p \ q$ ;
     $eq \leftarrow \text{weak-equality } pq \ r$ ;
    RETURN  $eq$ 
  }
} (is  $\langle - \geq ?A \rangle$ )
proof -
have check-add-alt-def:  $\langle \text{check-add } A \ \vee \ p \ q \ i \ r = \text{do } \{$ 
   $b \leftarrow \text{SPEC}(\lambda b. b \longrightarrow p \in \# \text{ dom-}m \ A \wedge q \in \# \text{ dom-}m \ A \wedge i \notin \# \text{ dom-}m \ A \wedge \text{vars } r \subseteq \mathcal{V})$ ;
  if  $\neg b$  then  $\text{SPEC}(\lambda b. b \longrightarrow p \in \# \text{ dom-}m \ A \wedge q \in \# \text{ dom-}m \ A \wedge i \notin \# \text{ dom-}m \ A \wedge \text{vars } r \subseteq \mathcal{V} \wedge$ 
     $\text{the } (\text{fmlookup } A \ p) + \text{the } (\text{fmlookup } A \ q) - r \in \text{ideal polynom-bool})$ 
  else
     $\text{SPEC}(\lambda b. b \longrightarrow p \in \# \text{ dom-}m \ A \wedge q \in \# \text{ dom-}m \ A \wedge i \notin \# \text{ dom-}m \ A \wedge \text{vars } r \subseteq \mathcal{V} \wedge$ 
       $\text{the } (\text{fmlookup } A \ p) + \text{the } (\text{fmlookup } A \ q) - r \in \text{ideal polynom-bool})\rangle$ 
(is  $\langle - = ?B \rangle$ )
by (auto simp: check-add-def RES-RES-RETURN-RES)
have  $\langle ?A \leq \Downarrow \text{Id } (\text{check-add } A \ \vee \ p \ q \ i \ r) \rangle$ 
  apply refine-vcg
  apply ((auto simp: check-add-alt-def weak-equality-def
    add-poly-spec-def RES-RES-RETURN-RES summarize-ASSERT-conv
    cong: if-cong
    intro!: ideal.span-zero; fail)+)
  done
then show ?thesis
  unfolding check-add-alt-def[symmetric]
  by simp
qed

lemma check-mult-alt-def:
 $\langle \text{check-mult } A \ \vee \ p \ q \ i \ r \geq$ 
  do {
     $b \leftarrow \text{SPEC}(\lambda b. b \longrightarrow p \in \# \text{ dom-}m \ A \wedge i \notin \# \text{ dom-}m \ A \wedge \text{vars } q \subseteq \mathcal{V} \wedge \text{vars } r \subseteq \mathcal{V})$ ;
    if  $\neg b$ 
    then RETURN False
    else do {
      ASSERT ( $p \in \# \text{ dom-}m \ A$ );
      let  $p = \text{the } (\text{fmlookup } A \ p)$ ;
       $pq \leftarrow \text{mult-poly-spec } p \ q$ ;
       $p \leftarrow \text{weak-equality } pq \ r$ ;
      RETURN  $p$ 
    }
  }
}
unfolding check-mult-def
apply (rule refine-IdD)
by refine-vcg
(auto simp: check-mult-def weak-equality-def
  mult-poly-spec-def RES-RES-RETURN-RES
  intro!: ideal.span-zero
  exI[of -  $\langle \text{the } (\text{fmlookup } A \ p) * q \rangle$ ])

```



```

primrec insort-key-rel :: ('b  $\Rightarrow$  'b  $\Rightarrow$  bool)  $\Rightarrow$  'b  $\Rightarrow$  'b list  $\Rightarrow$  'b list where
insort-key-rel f x [] = [x] |
insort-key-rel f x (y#ys) =
  (if f x y then (x#y#ys) else y#(insort-key-rel f x ys))

```

```

lemma set-insort-key-rel[simp]: (set (insort-key-rel R x xs) = insert x (set xs))
by (induction xs)
  auto

```

```

lemma sorted-wrt-insort-key-rel:
  (total-on R (insert x (set xs))  $\Longrightarrow$  transp R  $\Longrightarrow$  reflp R  $\Longrightarrow$ 
    sorted-wrt R xs  $\Longrightarrow$  sorted-wrt R (insort-key-rel R x xs))
apply (induction xs)
apply (auto dest: transpD)
apply (metis Restricted-Predicates.total-on-def in-mono insertI1 reflpD subset-insertI)
by (simp add: Restricted-Predicates.total-on-def)

```

```

lemma sorted-wrt-insort-key-rel2:
  (total-on R (insert x (set xs))  $\Longrightarrow$  transp R  $\Longrightarrow$  x  $\notin$  set xs  $\Longrightarrow$ 
    sorted-wrt R xs  $\Longrightarrow$  sorted-wrt R (insort-key-rel R x xs))
apply (induction xs)
apply (auto dest: transpD)
apply (metis Restricted-Predicates.total-on-def in-mono insertI1 subset-insertI)
by (simp add: Restricted-Predicates.total-on-def)

```

**Step checking** **definition** PAC-checker-l-step :: (string code-status  $\times$  string set  $\times$  -  $\Rightarrow$  (llist-polynom, string, nat) pac-step  $\Rightarrow$  -) **where**

```

(PAC-checker-l-step = ( $\lambda$ spec (st',  $\mathcal{V}$ , A) st. case st of
  Add - - -  $\Rightarrow$ 
    do {
      r  $\leftarrow$  full-normalize-poly (pac-res st);
      eq  $\leftarrow$  check-addition-l spec A  $\mathcal{V}$  (pac-src1 st) (pac-src2 st) (new-id st) r;
      let - = eq;
      if  $\neg$ is-cfailed eq
      then RETURN (merge-cstatus st' eq,
         $\mathcal{V}$ , fmulpd (new-id st) r A)
      else RETURN (eq,  $\mathcal{V}$ , A)
    }
  | Del -  $\Rightarrow$ 
    do {
      eq  $\leftarrow$  check-del-l spec A (pac-src1 st);
      let - = eq;
      if  $\neg$ is-cfailed eq
      then RETURN (merge-cstatus st' eq,  $\mathcal{V}$ , fmdrop (pac-src1 st) A)
      else RETURN (eq,  $\mathcal{V}$ , A)
    }
  | Mult - - -  $\Rightarrow$ 
    do {
      r  $\leftarrow$  full-normalize-poly (pac-res st);
      q  $\leftarrow$  full-normalize-poly (pac-mult st);
      eq  $\leftarrow$  check-mult-l spec A  $\mathcal{V}$  (pac-src1 st) q (new-id st) r;
      let - = eq;
      if  $\neg$ is-cfailed eq
      then RETURN (merge-cstatus st' eq,

```

```

       $\mathcal{V}$ ,  $\text{fmupd } (\text{new-id } st) \ r \ A$ )
    else RETURN ( $eq$ ,  $\mathcal{V}$ ,  $A$ )
  }
| Extension - - -  $\Rightarrow$ 
  do {
     $r \leftarrow \text{full-normalize-poly } ([\text{new-var } st], -1) \# (\text{pac-res } st)$ ;
    ( $eq$ )  $\leftarrow \text{check-extension-l spec } A \ \mathcal{V} \ (\text{new-id } st) \ (\text{new-var } st) \ r$ ;
    if  $\neg \text{is-cfailed } eq$ 
    then do {
      RETURN ( $st'$ ,
        insert ( $\text{new-var } st$ )  $\mathcal{V}$ ,  $\text{fmupd } (\text{new-id } st) \ r \ A$ )
    }
    else RETURN ( $eq$ ,  $\mathcal{V}$ ,  $A$ )
  }
)

```

**lemma** *pac-step-rel-raw-def*:

```

 $\langle \langle K, V, R \rangle \text{ pac-step-rel-raw} = \text{pac-step-rel-raw } K \ V \ R \rangle$ 
by (auto intro!: ext simp: relAPP-def)

```

**definition** *mononoms-equal-up-to-reorder* **where**

```

 $\langle \text{mononoms-equal-up-to-reorder } xs \ ys \longleftrightarrow$ 
  map ( $\lambda(a, b). (\text{mset } a, b)$ )  $xs = \text{map } (\lambda(a, b). (\text{mset } a, b)) \ ys \rangle$ 

```

**definition** *normalize-poly-l* **where**

```

 $\langle \text{normalize-poly-l } p = \text{SPEC } (\lambda p'.$ 
  normalize-poly- $p^{**} ((\lambda(a, b). (\text{mset } a, b)) \text{'\# mset } p) ((\lambda(a, b). (\text{mset } a, b)) \text{'\# mset } p') \wedge$ 
   $0 \notin \text{snd ' \# mset } p' \wedge$ 
  sorted-wrt ( $\text{rel2p } (\text{term-order-rel} \times_r \text{int-rel})$ )  $p' \wedge$ 
  ( $\forall x \in \text{mononoms } p'. \text{sorted-wrt } (\text{rel2p } \text{var-order-rel}) \ x) \rangle$ 

```

**definition** *remap-polys-l-dom-err* ::  $\langle \text{string nres} \rangle$  **where**

```

 $\langle \text{remap-polys-l-dom-err} = \text{SPEC } (\lambda-. \text{True}) \rangle$ 

```

**definition** *remap-polys-l* ::  $\langle \text{l-list-polynom} \Rightarrow \text{string set} \Rightarrow (\text{nat}, \text{l-list-polynom}) \text{ fmap} \Rightarrow$

$(- \text{code-status} \times \text{string set} \times (\text{nat}, \text{l-list-polynom}) \text{ fmap}) \text{ nres} \rangle$  **where**

```

 $\langle \text{remap-polys-l spec} = (\lambda \mathcal{V} \ A. \text{do}\{$ 
   $\text{dom} \leftarrow \text{SPEC}(\lambda \text{dom}. \text{set-mset } (\text{dom-m } A) \subseteq \text{dom} \wedge \text{finite dom});$ 
   $\text{failed} \leftarrow \text{SPEC}(\lambda-. \text{bool. True});$ 
  if failed
  then do {
     $c \leftarrow \text{remap-polys-l-dom-err};$ 
    RETURN ( $\text{error-msg } (0 :: \text{nat}) \ c, \mathcal{V}, \text{fmempty}$ )
  }
  else do {
    ( $b, \mathcal{V}, A$ )  $\leftarrow \text{FOREACH dom}$ 
    ( $\lambda i \ (b, \mathcal{V}, A').$ 
      if  $i \in \# \text{dom-m } A$ 
      then do {
         $p \leftarrow \text{full-normalize-poly } (\text{the } (\text{fmlookup } A \ i));$ 
         $eq \leftarrow \text{weak-equality-l } p \ \text{spec};$ 
         $\mathcal{V} \leftarrow \text{RETURN}(\mathcal{V} \cup \text{vars-l-list } (\text{the } (\text{fmlookup } A \ i)));$ 
        RETURN( $b \vee eq, \mathcal{V}, \text{fmupd } i \ p \ A'$ )
      }
    )
  }

```

```

    } else RETURN (b,  $\mathcal{V}$ ,  $A'$ )
    (False,  $\mathcal{V}$ , fmempty);
    RETURN (if b then CFOUND else CSUCCESS,  $\mathcal{V}$ , A)
  })

```

**definition** *PAC-checker-l* **where**

```

(PAC-checker-l spec A b st = do {
  (S, -) ← WHILET
    (λ((b, A), n). ¬is-cfailed b ∧ n ≠ [])
    (λ((bA), n). do {
      ASSERT(n ≠ []);
      S ← PAC-checker-l-step spec bA (hd n);
      RETURN (S, tl n)
    })
  ((b, A), st);
  RETURN S
})

```

## 10.2 Correctness

We now enter the locale to reason about polynomials directly.

**context** *poly-embed*  
**begin**

**abbreviation** *pac-step-rel* **where**

$\langle \text{pac-step-rel} \equiv \text{p2rel } (\langle \text{Id}, \text{fully-unsorted-poly-rel } O \text{ mset-poly-rel}, \text{var-rel} \rangle \text{ pac-step-rel-raw}) \rangle$

**abbreviation** *fmap-polys-rel* **where**

$\langle \text{fmap-polys-rel} \equiv \langle \text{nat-rel}, \text{sorted-poly-rel } O \text{ mset-poly-rel} \rangle \text{fmap-rel} \rangle$

**lemma**

$\langle \text{normalize-poly-p } s0 \ s \implies$   
 $(s0, p) \in \text{mset-poly-rel} \implies$   
 $(s, p) \in \text{mset-poly-rel} \rangle$   
**by** (auto simp: mset-poly-rel-def normalize-poly-p-poly-of-mset)

**lemma** *vars-poly-of-vars*:

$\langle \text{vars } (\text{poly-of-vars } a :: \text{int mpoly}) \subseteq (\varphi \text{ ' set-mset } a) \rangle$   
**by** (induction a)  
 (auto simp: vars-mult-Var)

**lemma** *vars-polynom-of-mset*:

$\langle \text{vars } (\text{polynom-of-mset } za) \subseteq \bigcup (\text{image } \varphi \text{ ' (set-mset o fst) ' set-mset } za) \rangle$   
**apply** (induction za)  
**using** vars-poly-of-vars  
**by** (fastforce elim!: in-vars-addE simp: vars-mult-Const split: if-splits)+

**lemma** *fully-unsorted-poly-rel-vars-subset-vars-llist*:

$\langle (A, B) \in \text{fully-unsorted-poly-rel } O \text{ mset-poly-rel} \implies \text{vars } B \subseteq \varphi \text{ ' vars-llist } A \rangle$   
**by** (auto simp: fully-unsorted-poly-list-rel-def mset-poly-rel-def  
 set-rel-def var-rel-def br-def vars-llist-def list-rel-append2 list-rel-append1  
 list-rel-split-right-iff list-mset-rel-def image-iff  
 unsorted-term-poly-list-rel-def list-rel-split-left-iff  
 dest!: set-rev-mp[OF - vars-polynom-of-mset] split-list  
 dest: multi-member-split)

*dest*: *arg-cong*[*of*  $\langle \text{mset} \rightarrow \langle \text{add-mset} \rightarrow \text{set-mset} \rangle \rangle$ ]

**lemma** *fully-unsorted-poly-rel-extend-vars*:

$\langle (A, B) \in \text{fully-unsorted-poly-rel } O \text{ mset-poly-rel} \implies$   
 $(x1c, x1a) \in \langle \text{var-rel} \rangle \text{set-rel} \implies$   
 $\text{RETURN } (x1c \cup \text{vars-llist } A)$   
 $\leq \Downarrow (\langle \text{var-rel} \rangle \text{set-rel})$   
 $(\text{SPEC } ((\subseteq) (x1a \cup \text{vars } (B)))) \rangle$   
**using** *fully-unsorted-poly-rel-vars-subset-vars-llist*[*of*  $A \ B$ ]  
**apply** (*subst RETURN-RES-refine-iff*)  
**apply** *clarsimp*  
**apply** (*rule exI*[*of* -  $\langle x1a \cup \varphi \text{ 'vars-llist } A \rangle$ ])  
**apply** (*auto simp*: *set-rel-def var-rel-def br-def*  
*dest*: *fully-unsorted-poly-rel-vars-subset-vars-llist*)  
**done**

**lemma** *remap-polys-l-remap-polys*:

**assumes**  
 $AB: \langle (A, B) \in \langle \text{nat-rel}, \text{fully-unsorted-poly-rel } O \text{ mset-poly-rel} \rangle \text{fmap-rel} \rangle$  **and**  
 $\text{spec}: \langle (\text{spec}, \text{spec}') \in \text{sorted-poly-rel } O \text{ mset-poly-rel} \rangle$  **and**  
 $V: \langle (\mathcal{V}, \mathcal{V}') \in \langle \text{var-rel} \rangle \text{set-rel} \rangle$   
**shows**  $\langle \text{remap-polys-l spec } \mathcal{V} \ A \leq$   
 $\Downarrow (\text{code-status-status-rel} \times_r \langle \text{var-rel} \rangle \text{set-rel} \times_r \text{fmap-polys-rel}) (\text{remap-polys spec}' \ \mathcal{V}' \ B) \rangle$   
**(is**  $\langle \cdot \leq \Downarrow ?R \rightarrow \rangle$   
**proof** –  
**have**  $1: \langle \text{inj-on id } (\text{dom} :: \text{nat set}) \rangle$  **for** *dom*  
**by** *auto*  
**have**  $H: \langle x \in \# \text{dom-m } A \implies$   
 $(\bigwedge p. (\text{the } (\text{fmlookup } A \ x), p) \in \text{fully-unsorted-poly-rel} \implies$   
 $(p, \text{the } (\text{fmlookup } B \ x)) \in \text{mset-poly-rel} \implies \text{thesis}) \implies$   
 $\text{thesis} \rangle$  **for**  $x$  *thesis*  
**using** *fmap-rel-nat-the-fmlookup*[*OF*  $AB$ , *of*  $x \ x$ ] *fmap-rel-nat-rel-dom-m*[*OF*  $AB$ ] **by** *auto*  
**have** *full-normalize-poly*:  $\langle \text{full-normalize-poly } (\text{the } (\text{fmlookup } A \ x))$   
 $\leq \Downarrow (\text{sorted-poly-rel } O \text{ mset-poly-rel})$   
 $(\text{SPEC}$   
 $(\lambda p. \text{the } (\text{fmlookup } B \ x') - p \in \text{More-Modules.ideal polynom-bool} \wedge$   
 $\text{vars } p \subseteq \text{vars } (\text{the } (\text{fmlookup } B \ x')))) \rangle$   
**if**  $x\text{-dom}: \langle x \in \# \text{dom-m } A \rangle$  **and**  $\langle (x, x') \in \text{Id} \rangle$  **for**  $x \ x'$   
**apply** (*rule*  $H$ [*OF*  $x\text{-dom}$ ])  
**subgoal** **for**  $p$   
**apply** (*rule* *full-normalize-poly-normalize-poly-p*[*THEN* *order-trans*])  
**apply** *assumption*  
**subgoal**  
**using** *that*(2) **apply** –  
**unfolding** *conc-fun-chain*[*symmetric*]  
**by** (*rule* *ref-two-step'*, *rule* *RES-refine*)  
 $(\text{auto simp: } \text{rtranclp-normalize-poly-p-poly-of-mset}$   
 $\text{mset-poly-rel-def ideal.span-zero})$   
**done**  
**done**

**have**  $H': \langle (p, pa) \in \text{sorted-poly-rel } O \text{ mset-poly-rel} \implies$   
 $\text{weak-equality-l } p \ \text{spec} \leq \text{SPEC } (\lambda \text{eqa}. \text{eqa} \longrightarrow pa = \text{spec}') \rangle$  **for**  $p \ pa$   
**using** *spec* **apply** (*auto simp*: *weak-equality-l-def weak-equality-spec-def*  
*list-mset-rel-def br-def*)

$dest: list-rel-term-poly-list-rel-same-rightD \text{ sorted-poly-list-rel}D$   
**by** (*metis* (*mono-tags*) *mem-Collect-eq mset-poly-rel-def prod.simps(2)*  
 $sorted-poly-list-relD$ )

**have** *emp*:  $\langle \mathcal{V}, \mathcal{V}' \rangle \in \langle var-rel \rangle set-rel \implies$   
 $((False, \mathcal{V}, fmempty), False, \mathcal{V}', fmempty) \in bool-rel \times_r \langle var-rel \rangle set-rel \times_r fmap-polys-rel$  **for**  $\mathcal{V} \mathcal{V}'$   
**by** *auto*  
**show** *?thesis*  
**using** *assms*  
**unfolding** *remap-polys-l-def remap-polys-l-dom-err-def*  
 $remap-polys-def prod.case$   
**apply** (*refine-rcg full-normalize-poly fmap-rel-fmupd-fmap-rel*)  
**subgoal**  
**by** *auto*  
**subgoal**  
**by** *auto*  
**subgoal**  
**by** (*auto simp: error-msg-def*)  
**apply** (*rule 1*)  
**subgoal by** *auto*  
**apply** (*rule emp*)  
**subgoal**  
**using** *V* **by** *auto*  
**subgoal by** *auto*  
**subgoal by** *auto*  
**subgoal by** (*rule H'*)  
**apply** (*rule fully-unsorted-poly-rel-extend-vars*)  
**subgoal by** (*auto intro!: fmap-rel-nat-the-fmlookup*)  
**subgoal by** (*auto intro!: fmap-rel-fmupd-fmap-rel*)  
**subgoal by** (*auto intro!: fmap-rel-fmupd-fmap-rel*)  
**subgoal by** *auto*  
**subgoal by** *auto*  
**done**  
**qed**

**lemma** *fref-to-Down-curry*:

$\langle (uncurry f, uncurry g) \in [P]_f A \rightarrow \langle B \rangle nres-rel \implies$   
 $(\bigwedge x x' y y'. P(x', y') \implies ((x, y), (x', y')) \in A \implies f x y \leq \Downarrow B (g x' y'))$   
**unfolding** *fref-def uncurry-def nres-rel-def*  
**by** *auto*

**lemma** *weak-equality-spec-weak-equality*:

$\langle (p, p') \in mset-poly-rel \implies$   
 $(r, r') \in mset-poly-rel \implies$   
 $weak-equality-spec p r \leq weak-equality p' r' \rangle$   
**unfolding** *weak-equality-spec-def weak-equality-def*  
**by** (*auto simp: mset-poly-rel-def*)

**lemma** *weak-equality-l-weak-equality-l'[refine]*:

$\langle weak-equality-l p q \leq \Downarrow bool-rel (weak-equality p' q') \rangle$   
**if**  $\langle (p, p') \in sorted-poly-rel O mset-poly-rel \rangle$   
 $\langle (q, q') \in sorted-poly-rel O mset-poly-rel \rangle$   
**for**  $p p' q q'$

```

using that
by (auto intro!: weak-equality-l-weak-equality-spec[THEN fref-to-Down-curry, THEN order-trans]
      ref-two-step'
      weak-equality-spec-weak-equality
      simp flip: conc-fun-chain)

lemma error-msg-ne-SUCCESS[iff]:
  ⟨error-msg i m ≠ CSUCCESS⟩
  ⟨error-msg i m ≠ CFOUND⟩
  ⟨is-cfailed (error-msg i m)⟩
  ⟨¬is-cfound (error-msg i m)⟩
by (auto simp: error-msg-def)

lemma sorted-poly-rel-vars-llist:
  ⟨(r, r') ∈ sorted-poly-rel O mset-poly-rel ⟹
    vars r' ⊆ φ ' vars-llist r⟩
apply (auto simp: mset-poly-rel-def
  set-rel-def var-rel-def br-def vars-llist-def list-rel-append2 list-rel-append1
  list-rel-split-right-iff list-mset-rel-def image-iff sorted-poly-list-rel-wrt-def
  dest!: set-rev-mp[OF - vars-polynom-of-mset]
  dest!: split-list)
apply (auto dest!: multi-member-split simp: list-rel-append1
  term-poly-list-rel-def eq-commute[of - ⟨mset -⟩]
  list-rel-split-right-iff list-rel-append2 list-rel-split-left-iff
  dest: arg-cong[of ⟨mset -⟩ ⟨add-mset - -⟩ set-mset])
done

lemma check-addition-l-check-add:
  assumes ⟨(A, B) ∈ fmap-polys-rel⟩ and ⟨(r, r') ∈ sorted-poly-rel O mset-poly-rel⟩
  ⟨(p, p') ∈ Id⟩ ⟨(q, q') ∈ Id⟩ ⟨(i, i') ∈ nat-rel⟩
  ⟨(V', V) ∈ ⟨var-rel⟩set-rel⟩
  shows
  ⟨check-addition-l spec A V' p q i r ≤ ⋈ {(st, b). (¬is-cfailed st ⟷ b) ∧
    (is-cfound st ⟹ spec = r)} (check-add B V p' q' i' r')⟩
proof -
have [refine]:
  ⟨add-poly-l (p, q) ≤ ⋈ (sorted-poly-rel O mset-poly-rel) (add-poly-spec p' q')⟩
if ⟨(p, p') ∈ sorted-poly-rel O mset-poly-rel⟩
  ⟨(q, q') ∈ sorted-poly-rel O mset-poly-rel⟩
for p p' q q'
using that
by (auto intro!: add-poly-l-add-poly-p'[THEN order-trans] ref-two-step'
      add-poly-p'-add-poly-spec
      simp flip: conc-fun-chain)

show ?thesis
using assms
unfolding check-addition-l-def
  check-not-equal-dom-err-def apply -
apply (rule order-trans)
defer
apply (rule ref-two-step')
apply (rule check-add-alt-def)
apply refine-rcg

```

```

subgoal
  by (drule sorted-poly-rel-vars-llist)
  (auto simp: set-rel-def var-rel-def br-def)
subgoal
  by auto
subgoal
  by auto
subgoal
  by auto
subgoal
  by auto
subgoal
  by auto
subgoal
  by auto
subgoal
  by (auto simp: weak-equality-l-def bind-RES-RETURN-eq)
done
qed

lemma check-del-l-check-del:
   $\langle (A, B) \in \text{fmap-polys-rel} \implies (x3, x3a) \in \text{Id} \implies \text{check-del-l spec } A \text{ (pac-src1 (Del } x3)) \leq \Downarrow \{(st, b). (\neg \text{is-cfailed } st \longleftrightarrow b) \wedge (b \longrightarrow st = \text{CSUCCESS})\} (\text{check-del } B \text{ (pac-src1 (Del } x3a))) \rangle$ 
  unfolding check-del-l-def check-del-def
  by (refine-vcg lhs-step-If RETURN-SPEC-refine)
  (auto simp: fmap-rel-nat-rel-dom-m bind-RES-RETURN-eq)

lemma check-mult-l-check-mult:
  assumes  $\langle (A, B) \in \text{fmap-polys-rel} \rangle$  and  $\langle (r, r') \in \text{sorted-poly-rel } O \text{ mset-poly-rel} \rangle$  and
     $\langle (q, q') \in \text{sorted-poly-rel } O \text{ mset-poly-rel} \rangle$ 
     $\langle (p, p') \in \text{Id} \rangle \langle (i, i') \in \text{nat-rel} \rangle \langle (\mathcal{V}, \mathcal{V}') \in \langle \text{var-rel} \rangle \text{set-rel} \rangle$ 
  shows
     $\langle \text{check-mult-l spec } A \mathcal{V} p q i r \leq \Downarrow \{(st, b). (\neg \text{is-cfailed } st \longleftrightarrow b) \wedge (\text{is-cfound } st \longrightarrow \text{spec} = r)\} (\text{check-mult } B \mathcal{V}' p' q' i' r') \rangle$ 
  proof -
    have [refine]:
       $\langle \text{mult-poly-full } p q \leq \Downarrow (\text{sorted-poly-rel } O \text{ mset-poly-rel}) (\text{mult-poly-spec } p' q') \rangle$ 
    if  $\langle (p, p') \in \text{sorted-poly-rel } O \text{ mset-poly-rel} \rangle$ 
       $\langle (q, q') \in \text{sorted-poly-rel } O \text{ mset-poly-rel} \rangle$ 
    for  $p p' q q'$ 
    using that
    by (auto intro!: mult-poly-full-mult-poly-p'[THEN order-trans] ref-two-step'
      mult-poly-p'-mult-poly-spec
      simp flip: conc-fun-chain)

  show ?thesis
    using assms
    unfolding check-mult-l-def
      check-mult-l-mult-err-def check-mult-l-dom-err-def apply -
    apply (rule order-trans)
    defer
    apply (rule ref-two-step')
    apply (rule check-mult-alt-def)
    apply refine-rcg
  subgoal

```

```

    by (drule sorted-poly-rel-vars-llist)+
      (fastforce simp: set-rel-def var-rel-def br-def)
  subgoal
    by auto
  subgoal
    by auto
  subgoal
    by auto
  subgoal
    by auto
  subgoal
    by (auto simp: weak-equality-l-def bind-RES-RETURN-eq)
  done
qed

```

**lemma** *normalize-poly-normalize-poly-spec*:

**assumes**  $\langle (r, t) \in \text{unsorted-poly-rel } O \text{ mset-poly-rel} \rangle$

**shows**  $\langle \text{normalize-poly } r \leq \Downarrow(\text{sorted-poly-rel } O \text{ mset-poly-rel}) (\text{normalize-poly-spec } t) \rangle$

**proof** –

**obtain**  $s$  **where**

$rs: \langle (r, s) \in \text{unsorted-poly-rel} \rangle$  **and**

$st: \langle (s, t) \in \text{mset-poly-rel} \rangle$

**using** *assms* **by** *auto*

**show** *?thesis*

**by** (rule *normalize-poly-normalize-poly-p[THEN order-trans, OF rs]*)

(use *st* **in**  $\langle \text{auto dest!}: \text{rtranclp-normalize-poly-p-poly-of-mset}$

$\text{intro!}: \text{ref-two-step}' \text{ RES-refine exI[of - } t]$

$\text{simp}: \text{normalize-poly-spec-def ideal.span-zero mset-poly-rel-def}$

$\text{simp flip}: \text{conc-fun-chain} \rangle$ )

**qed**

**lemma** *remove1-list-rel*:

$\langle (xs, ys) \in \langle R \rangle \text{ list-rel} \implies$

$(a, b) \in R \implies$

$\text{IS-RIGHT-UNIQUE } R \implies$

$\text{IS-LEFT-UNIQUE } R \implies$

$(\text{remove1 } a \text{ } xs, \text{remove1 } b \text{ } ys) \in \langle R \rangle \text{ list-rel} \rangle$

**by** (*induction xs ys rule: list-rel-induct*)

(*auto simp: single-valued-def IS-LEFT-UNIQUE-def*)

**lemma** *remove1-list-rel2*:

$\langle (xs, ys) \in \langle R \rangle \text{ list-rel} \implies$

$(a, b) \in R \implies$

$(\bigwedge c. (a, c) \in R \implies c = b) \implies$

$(\bigwedge c. (c, b) \in R \implies c = a) \implies$

$(\text{remove1 } a \text{ } xs, \text{remove1 } b \text{ } ys) \in \langle R \rangle \text{ list-rel} \rangle$

**apply** (*induction xs ys rule: list-rel-induct*)

**apply** (*simp (no-asm)*)

**by** (*smt list-rel-simp(4) remove1.simps(2)*)

**lemma** *remove1-sorted-poly-rel-mset-poly-rel*:

**assumes**

$\langle (r, r') \in \text{sorted-poly-rel } O \text{ mset-poly-rel} \rangle$  **and**



$\langle [a], 1 \rangle \in \text{set } r \rangle$   
**shows**  
 $\langle \text{remove1 } ([a], 1) \ r, r' - \text{Var } (\varphi \ a) \rangle$   
 $\in \text{sorted-poly-rel } O \ \text{mset-poly-rel} \rangle$   
**proof** –  
**have** [simp]:  $\langle ([a], \{\#a\# \}) \in \text{term-poly-list-rel} \rangle$   
 $\langle \bigwedge aa. ([a], aa) \in \text{term-poly-list-rel} \longleftrightarrow aa = \{\#a\# \} \rangle$   
**by** (auto simp: term-poly-list-rel-def)  
**have** H:  
 $\langle \bigwedge aa. ([a], aa) \in \text{term-poly-list-rel} \implies aa = \{\#a\# \} \rangle$   
 $\langle \bigwedge aa. (aa, \{\#a\# \}) \in \text{term-poly-list-rel} \implies aa = [a] \rangle$   
**by** (auto simp: single-valued-def IS-LEFT-UNIQUE-def term-poly-list-rel-def)  
  
**have** [simp]:  $\langle \text{Const } (1 :: \text{int}) = (1 :: \text{int mpoly}) \rangle$   
**by** (simp add: Const.abs-eq Const<sub>0</sub>-one one-mpoly.abs-eq)  
**have** [simp]:  $\langle \text{sorted-wrt term-order } (\text{map fst } r) \implies$   
 $\text{sorted-wrt term-order } (\text{map fst } (\text{remove1 } ([a], 1) \ r)) \rangle$   
**by** (induction r) auto  
**have** [intro]:  $\langle \text{distinct } (\text{map fst } r) \implies \text{distinct } (\text{map fst } (\text{remove1 } x \ r)) \rangle$  **for** x  
**by** (induction r) (auto dest: in-set-remove1D)  
**have** [simp]:  $\langle (r, ya) \in (\text{term-poly-list-rel} \times_r \text{int-rel}) \text{list-rel} \implies$   
 $\text{polynom-of-mset } (\text{mset } ya) - \text{Var } (\varphi \ a) =$   
 $\text{polynom-of-mset } (\text{remove1-mset } (\{\#a\# \}, 1) (\text{mset } ya)) \rangle$  **for** ya  
**using** assms  
**by** (auto simp: list-rel-append1 list-rel-split-right-iff  
dest!: split-list)  
  
**show** ?thesis  
**using** assms  
**apply** (auto simp: mset-poly-rel-def sorted-poly-list-rel-wrt-def)  
**apply** (rename-tac ya za, rule-tac b =  $\langle \text{remove1-mset } (\{\#a\# \}, 1) \ za \rangle$  **in** relcompI)  
**apply** (auto)  
**apply** (rename-tac ya za, rule-tac b =  $\langle \text{remove1 } (\{\#a\# \}, 1) \ ya \rangle$  **in** relcompI)  
**by** (auto intro!: remove1-list-rel2 intro: H  
simp: list-mset-rel-def br-def in-remove1-mset-neq)  
**qed**

**lemma** remove1-sorted-poly-rel-mset-poly-rel-minus:

**assumes**

$\langle (r, r') \in \text{sorted-poly-rel } O \ \text{mset-poly-rel} \rangle$  **and**

$\langle ([a], -1) \in \text{set } r \rangle$

**shows**

$\langle \text{remove1 } ([a], -1) \ r, r' + \text{Var } (\varphi \ a) \rangle$   
 $\in \text{sorted-poly-rel } O \ \text{mset-poly-rel} \rangle$

**proof** –

**have** [simp]:  $\langle ([a], \{\#a\# \}) \in \text{term-poly-list-rel} \rangle$   
 $\langle \bigwedge aa. ([a], aa) \in \text{term-poly-list-rel} \longleftrightarrow aa = \{\#a\# \} \rangle$   
**by** (auto simp: term-poly-list-rel-def)  
**have** H:  
 $\langle \bigwedge aa. ([a], aa) \in \text{term-poly-list-rel} \implies aa = \{\#a\# \} \rangle$   
 $\langle \bigwedge aa. (aa, \{\#a\# \}) \in \text{term-poly-list-rel} \implies aa = [a] \rangle$   
**by** (auto simp: single-valued-def IS-LEFT-UNIQUE-def term-poly-list-rel-def)

```

have [simp]:  $\langle \text{Const } (1 :: \text{int}) = (1 :: \text{int } \text{mpoly}) \rangle$ 
  by (simp add: Const.abs-eq Const0-one one-mpoly.abs-eq)
have [simp]:  $\langle \text{sorted-wrt term-order } (\text{map fst } r) \implies$ 
   $\text{sorted-wrt term-order } (\text{map fst } (\text{remove1 } ([a], -1) r)) \rangle$ 
  by (induction r) auto
have [intro]:  $\langle \text{distinct } (\text{map fst } r) \implies \text{distinct } (\text{map fst } (\text{remove1 } x r)) \rangle$  for x
  apply (induction r) apply auto
  by (meson img-fst in-set-remove1D)
have [simp]:  $\langle (r, ya) \in \langle \text{term-poly-list-rel } \times_r \text{ int-rel} \rangle \text{list-rel} \implies$ 
   $\text{polynom-of-mset } (\text{mset } ya) + \text{Var } (\varphi a) =$ 
   $\text{polynom-of-mset } (\text{remove1-mset } (\{\#a\}, -1) (\text{mset } ya)) \rangle$  for ya
using assms
  by (auto simp: list-rel-append1 list-rel-split-right-iff
    dest!: split-list)

show ?thesis
  using assms
  apply (auto simp: mset-poly-rel-def sorted-poly-list-rel-wrt-def
    Collect-eq-comp' dest!: )
  apply (rule-tac b =  $\langle \text{remove1-mset } (\{\#a\}, -1) za \rangle$  in relcompI)
  apply (auto)
  apply (rule-tac b =  $\langle \text{remove1 } (\{\#a\}, -1) ya \rangle$  in relcompI)
  apply (auto intro!: remove1-list-rel2 intro: H
    simp: list-mset-rel-def br-def in-remove1-mset-neq)
  done
qed

```

**lemma insert-var-rel-set-rel:**  
 $\langle (\mathcal{V}, \mathcal{V}') \in \langle \text{var-rel} \rangle \text{set-rel} \implies$   
 $(yb, x2) \in \text{var-rel} \implies$   
 $(\text{insert } yb \mathcal{V}, \text{insert } x2 \mathcal{V}') \in \langle \text{var-rel} \rangle \text{set-rel}$   
 by (auto simp: var-rel-def set-rel-def)

**lemma var-rel-set-rel-iff:**  
 $\langle (\mathcal{V}, \mathcal{V}') \in \langle \text{var-rel} \rangle \text{set-rel} \implies$   
 $(yb, x2) \in \text{var-rel} \implies$   
 $yb \in \mathcal{V} \longleftrightarrow x2 \in \mathcal{V}'$   
 using  $\varphi\text{-inj inj-eq}$  by (fastforce simp: var-rel-def set-rel-def br-def)

**lemma check-extension-l-check-extension:**  
 assumes  $\langle (A, B) \in \text{fmap-polys-rel} \rangle$  and  $\langle (r, r') \in \text{sorted-poly-rel } O \text{ mset-poly-rel} \rangle$  and  
 $\langle (i, i') \in \text{nat-rel} \rangle \langle (\mathcal{V}, \mathcal{V}') \in \langle \text{var-rel} \rangle \text{set-rel} \rangle \langle (x, x') \in \text{var-rel} \rangle$   
 shows  
 $\langle \text{check-extension-l spec } A \mathcal{V} i x r \leq$   
 $\Downarrow \{((st), (b)).$   
 $(\neg \text{is-cfailed } st \longleftrightarrow b) \wedge$   
 $(\text{is-cfound } st \longrightarrow \text{spec} = r) \} (\text{check-extension } B \mathcal{V}' i' x' r') \rangle$   
 proof –  
 have  $\langle x' = \varphi x \rangle$   
 using assms(5) by (auto simp: var-rel-def br-def)  
 have [refine]:  
 $\langle \text{mult-poly-full } p q \leq \Downarrow (\text{sorted-poly-rel } O \text{ mset-poly-rel}) (\text{mult-poly-spec } p' q') \rangle$   
 if  $\langle (p, p') \in \text{sorted-poly-rel } O \text{ mset-poly-rel} \rangle$

```

  ⟨(q, q') ∈ sorted-poly-rel O mset-poly-rel⟩
  for p p' q q'
  using that
  by (auto intro!: mult-poly-full-mult-poly-p'[THEN order-trans] ref-two-step'
      mult-poly-p'-mult-poly-spec
      simp flip: conc-fun-chain)
have [refine]:
  ⟨add-poly-l (p, q) ≤ ↓ (sorted-poly-rel O mset-poly-rel) (add-poly-spec p' q')⟩
  if ⟨(p, p') ∈ sorted-poly-rel O mset-poly-rel⟩
  ⟨(q, q') ∈ sorted-poly-rel O mset-poly-rel⟩
  for p p' q q'
  using that
  by (auto intro!: add-poly-l-add-poly-p'[THEN order-trans] ref-two-step'
      add-poly-p'-add-poly-spec
      simp flip: conc-fun-chain)

have [simp]: ⟨(l, l') ∈ (term-poly-list-rel ×r int-rel) list-rel ⟹
  (map (λ(a, b). (a, - b)) l, map (λ(a, b). (a, - b)) l')
  ∈ (term-poly-list-rel ×r int-rel) list-rel⟩ for l l'
  by (induction l l' rule: list-rel-induct)
  (auto simp: list-mset-rel-def br-def)

have [intro!]:
  ⟨(x2c, za) ∈ (term-poly-list-rel ×r int-rel) list-rel O list-mset-rel ⟹
  (map (λ(a, b). (a, - b)) x2c,
   {#case x of (a, b) ⇒ (a, - b). x ∈ # za#})
  ∈ (term-poly-list-rel ×r int-rel) list-rel O list-mset-rel⟩ for x2c za
  apply (auto)
  subgoal for y
    apply (induction x2c y rule: list-rel-induct)
    apply (auto simp: list-mset-rel-def br-def)
    apply (rule-tac b = ⟨(aa, - ba) # map (λ(a, b). (a, - b)) l'⟩ in relcompI)
    by auto
  done
have [simp]: ⟨(λx. fst (case x of (a, b) ⇒ (a, - b))) = fst⟩
  by (auto intro: ext)

have uminus: ⟨(x2c, x2a) ∈ sorted-poly-rel O mset-poly-rel ⟹
  (map (λ(a, b). (a, - b)) x2c,
   - x2a)
  ∈ sorted-poly-rel O mset-poly-rel⟩ for x2c x2a x1c x1a
  apply (clarsimp simp: sorted-poly-list-rel-wrt-def
      mset-poly-rel-def)
  apply (rule-tac b = ⟨(λ(a, b). (a, - b)) ' # za' in relcompI)
  by (auto simp: sorted-poly-list-rel-wrt-def
      mset-poly-rel-def comp-def polynom-of-mset-uminus)
have [simp]: ⟨([], 0) ∈ sorted-poly-rel O mset-poly-rel⟩
  by (auto simp: sorted-poly-list-rel-wrt-def
      mset-poly-rel-def list-mset-rel-def br-def
      intro!: relcompI[of - ⟨{#}⟩])
show ?thesis
  unfolding check-extension-l-def
    check-extension-l-dom-err-def
    check-extension-l-no-new-var-err-def
    check-extension-l-new-var-multiple-err-def

```

```

    check-extension-l-side-cond-err-def
  apply (rule order-trans)
  defer
  apply (rule ref-two-step')
  apply (rule check-extension-alt-def)
  apply (refine-vcg )
  subgoal using assms(1,3,4,5)
    by (auto simp: var-rel-set-rel-iff)
  subgoal using assms(1,3,4,5)
    by (auto simp: var-rel-set-rel-iff)
  subgoal by auto
  subgoal by auto
  apply (subst (x' =  $\varphi$  x), rule remove1-sorted-poly-rel-mset-poly-rel-minus)
  subgoal using assms by auto
  subgoal using assms by auto
  subgoal using sorted-poly-rel-vars-llist[of  $\langle r \rangle$   $\langle r' \rangle$ ]
    assms
    by (force simp: set-rel-def var-rel-def br-def
      dest!: sorted-poly-rel-vars-llist)
  subgoal by auto
  subgoal by auto
  subgoal using assms by auto
  apply (rule uminus)
  subgoal using assms by auto
  subgoal using assms by auto
  subgoal using assms by auto
  subgoal using assms by auto
  subgoal using assms by auto
  done
qed

```

**lemma** *full-normalize-poly-diff-ideal*:

```

  fixes dom
  assumes  $\langle (p, p') \in \text{fully-unsorted-poly-rel } O \text{ mset-poly-rel} \rangle$ 
  shows
     $\langle \text{full-normalize-poly } p \leq \Downarrow (\text{sorted-poly-rel } O \text{ mset-poly-rel})$ 
       $(\text{normalize-poly-spec } p') \rangle$ 
  proof -
    obtain q where
      pq:  $\langle (p, q) \in \text{fully-unsorted-poly-rel} \rangle$  and qp':  $\langle (q, p') \in \text{mset-poly-rel} \rangle$ 
    using assms by auto
    show ?thesis
      unfolding normalize-poly-spec-def
      apply (rule full-normalize-poly-normalize-poly-p[THEN order-trans])
      apply (rule pq)
      unfolding conc-fun-chain[symmetric]
      by (rule ref-two-step', rule RES-refine)
        (use qp' in (auto dest!: rtranclp-normalize-poly-p-poly-of-mset
          simp: mset-poly-rel-def ideal.span-zero))
  qed

```

**lemma** *insort-key-rel-decomp*:

```

 $\langle \exists ys \ zs. \ xs = ys @ zs \wedge \text{insort-key-rel } R \ x \ xs = ys @ x \# zs \rangle$ 

```

```

apply (induction xs)
apply (auto 5 3)
apply (rule-tac x = ⟨a # ys⟩ in exI)
apply auto
done

```

**lemma** *list-rel-append-same-length*:

```

  ⟨length xs = length xs' ⟹ (xs @ ys, xs' @ ys') ∈ ⟨R⟩list-rel ⟷ (xs, xs') ∈ ⟨R⟩list-rel ∧ (ys, ys') ∈
  ⟨R⟩list-rel
by (auto simp: list-rel-def list-all2-append2 dest: list-all2-lengthD)

```

**lemma** *term-poly-list-rel-list-relD*:  $\langle (ys, cs) \in \langle \text{term-poly-list-rel} \times_r \text{int-rel} \rangle \text{list-rel} \implies$

```

  cs = map (λ(a, y). (mset a, y)) ys⟩
by (induction ys arbitrary: cs)
  (auto simp: term-poly-list-rel-def list-rel-def list-all2-append list-all2-Cons1 list-all2-Cons2)

```

**lemma** *term-poly-list-rel-single*:  $\langle ([x32], \{\#x32\# \}) \in \text{term-poly-list-rel} \rangle$

```

by (auto simp: term-poly-list-rel-def)

```

**lemma** *unsorted-poly-rel-list-rel-list-rel-uminus*:

```

  ⟨(map (λ(a, b). (a, - b)) r, yc)
    ∈ ⟨unsorted-term-poly-list-rel ×r int-rel⟩list-rel ⟹
    (r, map (λ(a, b). (a, - b)) yc)
    ∈ ⟨unsorted-term-poly-list-rel ×r int-rel⟩list-rel
by (induction r arbitrary: yc)
  (auto simp: elim!: list-relE3)

```

**lemma** *mset-poly-rel-minus*:  $\langle (\{\#(a, b)\# \}, v') \in \text{mset-poly-rel} \implies$

```

  (mset yc, r') ∈ mset-poly-rel ⟹
  (r, yc)
  ∈ ⟨unsorted-term-poly-list-rel ×r int-rel⟩list-rel ⟹
  (add-mset (a, b) (mset yc),
    v' + r')
  ∈ mset-poly-rel
by (induction r arbitrary: r')
  (auto simp: mset-poly-rel-def polynom-of-mset-uminus)

```

**lemma** *fully-unsorted-poly-rel-diff*:

```

  ⟨([v], v') ∈ fully-unsorted-poly-rel O mset-poly-rel ⟹
  (r, r') ∈ fully-unsorted-poly-rel O mset-poly-rel ⟹
  (v # r,
    v' + r')
  ∈ fully-unsorted-poly-rel O mset-poly-rel
apply auto
apply (rule-tac b = ⟨y + ya⟩ in relcompI)
apply (auto simp: fully-unsorted-poly-list-rel-def list-mset-rel-def br-def)
apply (rule-tac b = ⟨yb @ yc⟩ in relcompI)
apply (auto elim!: list-relE3 simp: unsorted-poly-rel-list-rel-list-rel-uminus mset-poly-rel-minus)
done

```

**lemma** *PAC-checker-l-step-PAC-checker-step*:

```

assumes
  ⟨(Ast, Bst) ∈ code-status-status-rel ×r ⟨var-rel⟩set-rel ×r fmap-polys-rel⟩ and
  ⟨(st, st') ∈ pac-step-rel⟩ and
  spec: ⟨(spec, spec') ∈ sorted-poly-rel O mset-poly-rel⟩

```

**shows**

$\langle \text{PAC-checker-l-step spec Ast st} \leq \Downarrow (\text{code-status-status-rel} \times_r \langle \text{var-rel} \rangle \text{set-rel} \times_r \text{fmap-polys-rel})$   
 $(\text{PAC-checker-step spec}' Bst st') \rangle$

**proof** –

**obtain**  $A \mathcal{V} \text{ cst } B \mathcal{V}' \text{ cst}'$  **where**

$Ast: \langle Ast = (\text{cst}, \mathcal{V}, A) \rangle$  **and**

$Bst: \langle Bst = (\text{cst}', \mathcal{V}', B) \rangle$  **and**

$\mathcal{V}[\text{intro}]: \langle (\mathcal{V}, \mathcal{V}') \in \langle \text{var-rel} \rangle \text{set-rel} \rangle$  **and**

$AB: \langle (A, B) \in \text{fmap-polys-rel} \rangle$

$\langle (\text{cst}, \text{cst}') \in \text{code-status-status-rel} \rangle$

**using**  $\text{assms}(1)$

**by**  $(\text{cases } Ast; \text{cases } Bst; \text{auto})$

**have**  $[\text{refine}]: \langle (r, ra) \in \text{sorted-poly-rel } O \text{ mset-poly-rel} \implies$

$(\text{eqa}, \text{eqaa})$

$\in \{(st, b). (\neg \text{is-cfailed } st \longleftrightarrow b) \wedge (\text{is-cfound } st \longrightarrow \text{spec} = r)\} \implies$

$\text{RETURN eqa}$

$\leq \Downarrow \text{code-status-status-rel}$

$(\text{SPEC}$

$(\lambda st'. (\neg \text{is-failed } st' \wedge$

$\text{is-found } st' \longrightarrow$

$ra - \text{spec}' \in \text{More-Modules.ideal polynom-bool})) \rangle$

**for**  $r \text{ ra eqa eqaa}$

**using**  $\text{spec}$

**by**  $(\text{cases eqa})$

$(\text{auto intro!}; \text{RETURN-RES-refine dest!}; \text{sorted-poly-list-relD}$

$\text{simp}; \text{mset-poly-rel-def ideal.span-zero})$

**have**  $[\text{simp}]: \langle (\text{eqa}, st'a) \in \text{code-status-status-rel} \implies$

$(\text{merge-cstatus cst eqa}, \text{merge-status cst}' st'a)$

$\in \text{code-status-status-rel} \rangle$  **for**  $\text{eqa } st'a$

**using**  $AB$

**by**  $(\text{cases eqa}; \text{cases } st'a)$

$(\text{auto simp}; \text{code-status-status-rel-def})$

**have**  $[\text{simp}]: \langle (\text{merge-cstatus cst CSUCCESS}, \text{cst}') \in \text{code-status-status-rel} \rangle$

**using**  $AB$

**by**  $(\text{cases } st)$

$(\text{auto simp}; \text{code-status-status-rel-def})$

**have**  $[\text{simp}]: \langle (x32, x32a) \in \text{var-rel} \implies$

$([(x32], -1::\text{int}), -\text{Var } x32a) \in \text{fully-unsorted-poly-rel } O \text{ mset-poly-rel} \rangle$  **for**  $x32 \ x32a$

**by**  $(\text{auto simp}; \text{mset-poly-rel-def fully-unsorted-poly-list-rel-def list-mset-rel-def br-def}$

$\text{unsorted-term-poly-list-rel-def var-rel-def Const-1-eq-1}$

$\text{intro!}; \text{relcompI}[\text{of } - \langle \{ \#(\{ \#x32 \# \}, -1 :: \text{int}) \# \} \rangle]$

$\text{relcompI}[\text{of } - \langle [(\{ \#x32 \# \}, -1)] \rangle])$

**have**  $H3: \langle p - \text{Var } a = (-\text{Var } a) + p \rangle$  **for**  $p :: \langle \text{int mpoly} \rangle$  **and**  $a$

**by**  $\text{auto}$

**show**  $?thesis$

**using**  $\text{assms}(2)$

**unfolding**  $\text{PAC-checker-l-step-def PAC-checker-step-def Ast Bst prod.case}$

**apply**  $(\text{cases } st; \text{cases } st'; \text{simp only}; \text{p2rel-def pac-step.case}$

$\text{pac-step-rel-raw-def mem-Collect-eq prod.case pac-step-rel-raw.simps})$

**subgoal**

**apply**  $(\text{refine-rcg normalize-poly-normalize-poly-spec}$

$\text{check-mult-l-check-mult check-addition-l-check-add}$

$\text{full-normalize-poly-diff-ideal})$

**subgoal using**  $AB$  **by**  $\text{auto}$

**subgoal using**  $AB$  **by**  $\text{auto}$

```

subgoal by (auto simp: )
subgoal by (auto simp: )
subgoal by (auto simp: )
subgoal by (auto intro:  $\mathcal{V}$ )
apply assumption+
subgoal
  by (auto simp: code-status-status-rel-def)
subgoal
  by (auto intro!: fmap-rel-fmupd-fmap-rel
    fmap-rel-fmdrop-fmap-rel AB)
subgoal using AB by auto
done
subgoal
  apply (refine-rcg normalize-poly-normalize-poly-spec
    check-mult-l-check-mult check-addition-l-check-add
    full-normalize-poly-diff-ideal[unfolded normalize-poly-spec-def[symmetric]])
  subgoal using AB by auto
  subgoal using AB by auto
  subgoal using AB by (auto simp: )
  subgoal by (auto simp: )
  subgoal by (auto simp: )
  subgoal by (auto simp: )
  apply assumption+
  subgoal
    by (auto simp: code-status-status-rel-def)
  subgoal
    by (auto intro!: fmap-rel-fmupd-fmap-rel
      fmap-rel-fmdrop-fmap-rel AB)
  subgoal using AB by auto
  done
subgoal
  apply (refine-rcg full-normalize-poly-diff-ideal
    check-extension-l-check-extension)
  subgoal using AB by (auto intro!: fully-unsorted-poly-rel-diff[of -  $\langle \text{--- Var -} :: \text{int mpoly} \rangle$ , unfolded
H3[symmetric]] simp: comp-def case-prod-beta)
  subgoal using AB by auto
  subgoal using AB by (auto simp: )
  subgoal by (auto simp: )
  subgoal by auto
  subgoal
    by (auto simp: code-status-status-rel-def)
  subgoal
    by (auto simp: AB
      intro!: fmap-rel-fmupd-fmap-rel insert-var-rel-set-rel)
  subgoal
    by (auto simp: code-status-status-rel-def AB
      code-status.is-cfailed-def)
  done
subgoal
  apply (refine-rcg normalize-poly-normalize-poly-spec
    check-del-l-check-del check-addition-l-check-add
    full-normalize-poly-diff-ideal[unfolded normalize-poly-spec-def[symmetric]])
  subgoal using AB by auto
  subgoal using AB by auto
  subgoal

```

```

    by (auto intro!: fmap-rel-fmupd-fmap-rel
        fmap-rel-fmdrop-fmap-rel code-status-status-rel-def AB)
  subgoal
    by (auto intro!: fmap-rel-fmupd-fmap-rel
        fmap-rel-fmdrop-fmap-rel AB)
  done
done
qed

```

**lemma** *code-status-status-rel-discrim-iff*:

```

  ⟨(x1a, x1c) ∈ code-status-status-rel ⟹ is-cfailed x1a ⟷ is-failed x1c⟩
  ⟨(x1a, x1c) ∈ code-status-status-rel ⟹ is-cfound x1a ⟷ is-found x1c⟩
  by (cases x1a; cases x1c; auto; fail)+

```

**lemma** *PAC-checker-l-PAC-checker*:

```

  assumes
    ⟨(A, B) ∈ ⟨var-rel⟩set-rel ×r fmap-polys-rel⟩ and
    ⟨(st, st') ∈ ⟨pac-step-rel⟩list-rel⟩ and
    ⟨(spec, spec') ∈ sorted-poly-rel O mset-poly-rel⟩ and
    ⟨(b, b') ∈ code-status-status-rel⟩
  shows
    ⟨PAC-checker-l spec A b st ≤↓ (code-status-status-rel ×r ⟨var-rel⟩set-rel ×r fmap-polys-rel) (PAC-checker
spec' B b' st')⟩
  proof -
    have [refine0]: ⟨(((b, A), st), (b', B), st') ∈ ((code-status-status-rel ×r ⟨var-rel⟩set-rel ×r fmap-polys-rel)
×r ⟨pac-step-rel⟩list-rel)⟩
    using assms by (auto simp: code-status-status-rel-def)
    show ?thesis
    using assms
    unfolding PAC-checker-l-def PAC-checker-def
    apply (refine-rcg PAC-checker-l-step-PAC-checker-step
      WHILEIT-refine[where R = ⟨((bool-rel ×r ⟨var-rel⟩set-rel ×r fmap-polys-rel) ×r ⟨pac-step-rel⟩list-rel)⟩])
    subgoal by (auto simp: code-status-status-rel-discrim-iff)
    subgoal by auto
    subgoal by (auto simp: neq-Nil-conv)
    subgoal by (auto simp: neq-Nil-conv intro!: param-nth)
    subgoal by (auto simp: neq-Nil-conv)
    subgoal by auto
  done
qed

```

end

**lemma** *less-than-char-of-char[code-unfold]*:

```

  ⟨(x, y) ∈ less-than-char ⟷ (of-char x :: nat) < of-char y⟩
  by (auto simp: less-than-char-def less-char-def)

```

**lemmas** [code] =  
*add-poly-l'.simps[unfolded var-order-rel-def]*

**export-code** *add-poly-l'* in *SML module-name test*

**definition** *full-checker-l*

```

  :: ⟨líst-polynom ⟹ (nat, líst-polynom) fmap ⟹ (-, string, nat) pac-step list ⟹

```



```

  (string code-status × -) nres)
where
  ⟨full-checker-l spec A st = do {
    spec' ← full-normalize-poly spec;
    (b, V, A) ← remap-polys-l spec' {} A;
    if is-cfailed b
    then RETURN (b, V, A)
    else do {
      let V = V ∪ vars-llist spec;
      PAC-checker-l spec' (V, A) b st
    }
  }⟩

context poly-embed
begin

term normalize-poly-spec
thm full-normalize-poly-diff-ideal[unfolded normalize-poly-spec-def[symmetric]]
abbreviation unsorted-fmap-polys-rel where
  ⟨unsorted-fmap-polys-rel ≡ ⟨nat-rel, fully-unsorted-poly-rel O mset-poly-rel⟩fmap-rel⟩

lemma full-checker-l-full-checker:
  assumes
    ⟨(A, B) ∈ unsorted-fmap-polys-rel⟩ and
    ⟨(st, st') ∈ ⟨pac-step-rel⟩list-rel⟩ and
    ⟨(spec, spec') ∈ fully-unsorted-poly-rel O mset-poly-rel⟩
  shows
    ⟨full-checker-l spec A st ≤ ↓ (code-status-status-rel ×r ⟨var-rel⟩set-rel ×r fmap-polys-rel) (full-checker
    spec' B st')⟩
proof -
  have [refine]:
    ⟨(spec, spec') ∈ sorted-poly-rel O mset-poly-rel ⟹
    (V, V') ∈ ⟨var-rel⟩set-rel ⟹
    remap-polys-l spec V A ≤ ↓ (code-status-status-rel ×r ⟨var-rel⟩set-rel ×r fmap-polys-rel)
    (remap-polys-change-all spec' V' B)⟩ for spec spec' V V'
  apply (rule remap-polys-l-remap-polys[THEN order-trans, OF assms(1)])
  apply assumption+
  apply (rule ref-two-step[OF order.refl])
  apply (rule remap-polys-spec[THEN order-trans])
  by (rule remap-polys-polynom-bool-remap-polys-change-all)

show ?thesis
  unfolding full-checker-l-def full-checker-def
  apply (refine-rcg remap-polys-l-remap-polys
    full-normalize-poly-diff-ideal[unfolded normalize-poly-spec-def[symmetric]]
    PAC-checker-l-PAC-checker)
  subgoal
    using assms(3) .
  subgoal by auto
  subgoal by (auto simp: is-cfailed-def is-failed-def)
  subgoal by auto
  apply (rule fully-unsorted-poly-rel-extend-vars)
  subgoal using assms(3) .

```

```

subgoal by auto
subgoal by auto
subgoal
  using assms(2) by (auto simp: p2rel-def)
subgoal by auto
done
qed

```

**lemma** *full-checker-l-full-checker'*:

```

⟨(uncurry2 full-checker-l, uncurry2 full-checker) ∈
((fully-unsorted-poly-rel O mset-poly-rel) ×r unsorted-fmap-polys-rel) ×r ⟨pac-step-rel⟩list-rel →f
⟨(code-status-status-rel ×r ⟨var-rel⟩set-rel ×r fmap-polys-rel)⟩nres-rel⟩
apply (intro frefI nres-relI)
using full-checker-l-full-checker by force

```

**end**

**definition** *remap-polys-l2* :: *⟨llist-polynom ⇒ string set ⇒ (nat, llist-polynom) fmap ⇒ - nres⟩* **where**

```

⟨remap-polys-l2 spec = (λV A. do{
  n ← upper-bound-on-dom A;
  b ← RETURN (n ≥ 264);
  if b
  then do {
    c ← remap-polys-l-dom-err;
    RETURN (error-msg (0 :: nat) c, V, fmempty)
  }
  else do {
    (b, V, A) ← nfoldli ([0..n]) (λ-. True)
    (λi (b, V, A').
      if i ∈# dom-m A
      then do {
        ASSERT(fmlookup A i ≠ None);
        p ← full-normalize-poly (the (fmlookup A i));
        eq ← weak-equality-l p spec;
        V ← RETURN (V ∪ vars-llist (the (fmlookup A i)));
        RETURN(b ∨ eq, V, fmupd i p A')
      } else RETURN (b, V, A')
    )
    (False, V, fmempty);
    RETURN (if b then CFOUND else CSUCCESS, V, A)
  }
}⟩

```

**lemma** *remap-polys-l2-remap-polys-l*:

```

⟨remap-polys-l2 spec V A ≤ ↓ Id (remap-polys-l spec V A)⟩

```

**proof** –

```

have [refine]: ⟨(A, A') ∈ Id ⇒ upper-bound-on-dom A
  ≤ ↓ {(n, dom). dom = set [0..n] } (SPEC (λdom. set-mset (dom-m A') ⊆ dom ∧ finite dom))⟩ for
A A'
unfolding upper-bound-on-dom-def
apply (rule RES-refine)
apply (auto simp: upper-bound-on-dom-def)
done
have 1: ⟨inj-on id dom⟩ for dom

```

```

  by auto
have 2:  $\langle x \in \# \text{ dom-m } A \implies$ 
   $x' \in \# \text{ dom-m } A' \implies$ 
   $(x, x') \in \text{ nat-rel } \implies$ 
   $(A, A') \in \text{ Id } \implies$ 
   $\text{full-normalize-poly } (\text{the } (\text{fmlookup } A \ x))$ 
   $\leq \Downarrow \text{ Id}$ 
   $(\text{full-normalize-poly } (\text{the } (\text{fmlookup } A' \ x')))$ 
  for  $A \ A' \ x \ x'$ 
  by (auto)
have 3:  $\langle (n, \text{ dom}) \in \{(n, \text{ dom}). \text{ dom} = \text{set } [0..<n]\} \implies$ 
   $([0..<n], \text{ dom}) \in \langle \text{ nat-rel } \rangle \text{ list-set-rel } \rangle$  for  $n \ \text{ dom}$ 
by (auto simp: list-set-rel-def br-def)
have 4:  $\langle (p, q) \in \text{ Id } \implies$ 
   $\text{weak-equality-l } p \ \text{spec} \leq \Downarrow \text{ Id } (\text{weak-equality-l } q \ \text{spec}) \rangle$  for  $p \ q \ \text{spec}$ 
  by auto

have 6:  $\langle a = b \implies (a, b) \in \text{ Id } \rangle$  for  $a \ b$ 
  by auto
show ?thesis
  unfolding remap-polys-l2-def remap-polys-l-def
  apply (refine-rcg LFO-refine[where  $R = \langle \text{ Id } \times_r \langle \text{ Id } \rangle \text{ set-rel } \times_r \text{ Id } \rangle$ ])
  subgoal by auto
  subgoal by auto
  subgoal by auto
  apply (rule 3)
  subgoal by auto
  subgoal by (simp add: in-dom-m-lookup-iff)
  subgoal by (simp add: in-dom-m-lookup-iff)
  apply (rule 2)
  subgoal by auto
  subgoal by auto
  subgoal by auto
  subgoal by auto
  apply (rule 4; assumption)
  apply (rule 6)
  subgoal by auto
  subgoal by auto
  subgoal by auto
  subgoal by auto
  subgoal by auto
done
qed

end
theory PAC-Checker-Relation
  imports PAC-Checker WB-Sort Native-Word.Uint64
begin

```

## 11 Various Refinement Relations

When writing this, it was not possible to share the definition with the IsaSAT version.

**definition** *uint64-nat-rel* ::  $(\text{uint64} \times \text{nat}) \ \text{set}$  **where**  
 $\langle \text{uint64-nat-rel} = \text{br nat-of-uint64 } (\lambda \cdot. \text{ True}) \rangle$

**abbreviation** *uint64-nat-assn* **where**  
 $\langle \text{uint64-nat-assn} \equiv \text{pure } \text{uint64-nat-rel} \rangle$

**instantiation** *uint32* :: *hashable*  
**begin**

**definition** *hashcode-uint32* ::  $\langle \text{uint32} \Rightarrow \text{uint32} \rangle$  **where**  
 $\langle \text{hashcode-uint32 } n = n \rangle$

**definition** *def-hashmap-size-uint32* ::  $\langle \text{uint32 itself} \Rightarrow \text{nat} \rangle$  **where**  
 $\langle \text{def-hashmap-size-uint32} = (\lambda -. 16) \rangle$   
— same as *nat*

**instance**  
**by** *standard* (*simp add: def-hashmap-size-uint32-def*)  
**end**

**instantiation** *uint64* :: *hashable*  
**begin**

**definition** *hashcode-uint64* ::  $\langle \text{uint64} \Rightarrow \text{uint32} \rangle$  **where**  
 $\langle \text{hashcode-uint64 } n = (\text{uint32-of-nat } (\text{nat-of-uint64 } ((n) \text{ AND } ((2 :: \text{uint64})^{\wedge} 32 - 1)))) \rangle$

**definition** *def-hashmap-size-uint64* ::  $\langle \text{uint64 itself} \Rightarrow \text{nat} \rangle$  **where**  
 $\langle \text{def-hashmap-size-uint64} = (\lambda -. 16) \rangle$   
— same as *nat*

**instance**  
**by** *standard* (*simp add: def-hashmap-size-uint64-def*)  
**end**

**lemma** *word-nat-of-uint64-Rep-inject*[*simp*]:  $\langle \text{nat-of-uint64 } ai = \text{nat-of-uint64 } bi \longleftrightarrow ai = bi \rangle$   
**by** *transfer simp*

**instance** *uint64* :: *heap*  
**by** *standard* (*auto simp: inj-def exI*[*of - nat-of-uint64*])

**instance** *uint64* :: *semiring-numeral*  
**by** *standard*

**lemma** *nat-of-uint64-012*[*simp*]:  $\langle \text{nat-of-uint64 } 0 = 0 \rangle \langle \text{nat-of-uint64 } 2 = 2 \rangle \langle \text{nat-of-uint64 } 1 = 1 \rangle$   
**by** (*transfer, auto*)+

**definition** *uint64-of-nat-conv* **where**  
[*simp*]:  $\langle \text{uint64-of-nat-conv } (x :: \text{nat}) = x \rangle$

**lemma** *less-upper-bintrunc-id*:  $\langle n < 2^b \implies n \geq 0 \implies \text{bintrunc } b \ n = n \rangle$   
**unfolding** *uint32-of-nat-def*  
**by** (*simp add: no-bintr-alt1*)

**lemma** *nat-of-uint64-uint64-of-nat-id*:  $\langle n < 2^{64} \implies \text{nat-of-uint64 } (\text{uint64-of-nat } n) = n \rangle$   
**unfolding** *uint64-of-nat-def*  
**apply** *simp*  
**apply** *transfer*  
**apply** (*auto simp: unat-def*)  
**apply** *transfer*  
**by** (*auto simp: less-upper-bintrunc-id*)

**lemma** [*sepref-fr-rules*]:

$\langle (\text{return } o \text{ uint64-of-nat}, \text{RETURN } o \text{ uint64-of-nat-conv}) \in [\lambda a. a < 2^{64}]_a \text{ nat-assn}^k \rightarrow \text{uint64-nat-assn} \rangle$   
**by** *sepref-to-hoare*  
*(sepf-auto simp: uint64-nat-rel-def br-def nat-of-uint64-uint64-of-nat-id)*

**definition** *string-rel* ::  $\langle (\text{String.literal} \times \text{string}) \text{ set} \rangle$  **where**  
 $\langle \text{string-rel} = \{(x, y). y = \text{String.explode } x\} \rangle$

**abbreviation** *string-assn* ::  $\langle \text{string} \Rightarrow \text{String.literal} \Rightarrow \text{assn} \rangle$  **where**  
 $\langle \text{string-assn} \equiv \text{pure string-rel} \rangle$

**lemma** *eq-string-eq*:  
 $\langle ((=), (=)) \in \text{string-rel} \rightarrow \text{string-rel} \rightarrow \text{bool-rel} \rangle$   
**by** (*auto intro!: frefI simp: string-rel-def String.less-literal-def less-than-char-def rel2p-def literal.explode-inject*)

**lemmas** *eq-string-eq-hnr* =  
 $\text{eq-string-eq}[\text{sepref-import-param}]$

**definition** *string2-rel* ::  $\langle (\text{string} \times \text{string}) \text{ set} \rangle$  **where**  
 $\langle \text{string2-rel} \equiv \langle \text{Id} \rangle \text{list-rel} \rangle$

**abbreviation** *string2-assn* ::  $\langle \text{string} \Rightarrow \text{string} \Rightarrow \text{assn} \rangle$  **where**  
 $\langle \text{string2-assn} \equiv \text{pure string2-rel} \rangle$

**abbreviation** *monom-rel* **where**  
 $\langle \text{monom-rel} \equiv \langle \text{string-rel} \rangle \text{list-rel} \rangle$

**abbreviation** *monom-assn* **where**  
 $\langle \text{monom-assn} \equiv \text{list-assn string-assn} \rangle$

**abbreviation** *monomial-rel* **where**  
 $\langle \text{monomial-rel} \equiv \text{monom-rel} \times_r \text{int-rel} \rangle$

**abbreviation** *monomial-assn* **where**  
 $\langle \text{monomial-assn} \equiv \text{monom-assn} \times_a \text{int-assn} \rangle$

**abbreviation** *poly-rel* **where**  
 $\langle \text{poly-rel} \equiv \langle \text{monomial-rel} \rangle \text{list-rel} \rangle$

**abbreviation** *poly-assn* **where**  
 $\langle \text{poly-assn} \equiv \text{list-assn monomial-assn} \rangle$

**lemma** *poly-assn-alt-def*:  
 $\langle \text{poly-assn} = \text{pure poly-rel} \rangle$   
**by** (*simp add: list-assn-pure-conv*)

**abbreviation** *polys-assn* **where**  
 $\langle \text{polys-assn} \equiv \text{hm-fmap-assn uint64-nat-assn poly-assn} \rangle$

**lemma** *string-rel-string-assn*:  
 $\langle (\uparrow ((c, a) \in \text{string-rel})) = \text{string-assn } a \text{ } c \rangle$   
**by** (*auto simp: pure-app-eq*)

**lemma** *single-valued-string-rel*:

```

⟨single-valued string-rel⟩
by (auto simp: single-valued-def string-rel-def)

lemma IS-LEFT-UNIQUE-string-rel:
  ⟨IS-LEFT-UNIQUE string-rel⟩
  by (auto simp: IS-LEFT-UNIQUE-def single-valued-def string-rel-def
    literal.explode-inject)

lemma IS-RIGHT-UNIQUE-string-rel:
  ⟨IS-RIGHT-UNIQUE string-rel⟩
  by (auto simp: single-valued-def string-rel-def
    literal.explode-inject)

lemma single-valued-monom-rel: ⟨single-valued monom-rel⟩
  by (rule list-rel-sv)
  (auto intro!: frefI simp: string-rel-def
    rel2p-def single-valued-def p2rel-def)

lemma single-valued-monomial-rel:
  ⟨single-valued monomial-rel⟩
  using single-valued-monom-rel
  by (auto intro!: frefI simp:
    rel2p-def single-valued-def p2rel-def)

lemma single-valued-monom-rel': ⟨IS-LEFT-UNIQUE monom-rel⟩
  unfolding IS-LEFT-UNIQUE-def inv-list-rel-eq string2-rel-def
  by (rule list-rel-sv)+
  (auto intro!: frefI simp: string-rel-def
    rel2p-def single-valued-def p2rel-def literal.explode-inject)

lemma single-valued-monomial-rel':
  ⟨IS-LEFT-UNIQUE monomial-rel⟩
  using single-valued-monom-rel'
  unfolding IS-LEFT-UNIQUE-def inv-list-rel-eq
  by (auto intro!: frefI simp:
    rel2p-def single-valued-def p2rel-def)

lemma [safe-constraint-rules]:
  ⟨Sepref-Constraints.CONSTRAINT single-valued string-rel⟩
  ⟨Sepref-Constraints.CONSTRAINT IS-LEFT-UNIQUE string-rel⟩
  by (auto simp: CONSTRAINT-def single-valued-def
    string-rel-def IS-LEFT-UNIQUE-def literal.explode-inject)

lemma eq-string-monom-hnr[sepref-fr-rules]:
  ⟨(uncurry (return oo (=)), uncurry (RETURN oo (=))) ∈ monom-assnk *a monom-assnk →a bool-assn⟩
  using single-valued-monom-rel' single-valued-monom-rel
  unfolding list-assn-pure-conv
  by sepref-to-hoare
  (sep-auto simp: list-assn-pure-conv string-rel-string-assn
    single-valued-def IS-LEFT-UNIQUE-def
    dest!: mod-starD
    simp flip: inv-list-rel-eq)

```

```

definition term-order-rel' where
  [simp]:  $\langle \text{term-order-rel}'\ x\ y = ((x, y) \in \text{term-order-rel}) \rangle$ 

lemma term-order-rel[def-pat-rules]:
   $\langle (\in)\$(x,y)\$\text{term-order-rel} \equiv \text{term-order-rel}'\$x\$y \rangle$ 
  by auto

lemma term-order-rel-alt-def:
   $\langle \text{term-order-rel} = \text{lexord}\ (p2rel\ \text{char}.\text{lexordp}) \rangle$ 
  by (auto simp: p2rel-def char.lexordp-conv-lexord var-order-rel-def intro!: arg-cong[of - - lexord])

instantiation char :: linorder
begin
  definition less-char where [symmetric, simp]: less-char = PAC-Polynoms-Term.less-char
  definition less-eq-char where [symmetric, simp]: less-eq-char = PAC-Polynoms-Term.less-eq-char
instance
  apply standard
  using char.linorder-axioms
  by (auto simp: class.linorder-def class.order-def class.preorder-def
    less-eq-char-def less-than-char-def class.order-axioms-def
    class.linorder-axioms-def p2rel-def less-char-def)
end

instantiation list :: (linorder) linorder
begin
  definition less-list where less-list = lexordp (<)
  definition less-eq-list where less-eq-list = lexordp-eq

instance
  apply standard
  apply (auto simp: less-list-def less-eq-list-def List.lexordp-def
    lexordp-conv-lexord lexordp-into-lexordp-eq lexordp-antisym
    antisym-def lexordp-eq-refl lexordp-eq-linear intro: lexordp-eq-trans
    dest: lexordp-eq-antisym)
  apply (metis lexordp-antisym lexordp-conv-lexord lexordp-eq-conv-lexord)
  using lexordp-conv-lexord lexordp-conv-lexordp-eq apply blast
  done

end

lemma term-order-rel'-alt-def-lexord:
   $\langle \text{term-order-rel}'\ x\ y = \text{ord-class.lexordp}\ x\ y \rangle$  and
  term-order-rel'-alt-def:
   $\langle \text{term-order-rel}'\ x\ y \longleftrightarrow x < y \rangle$ 
proof –
  show
   $\langle \text{term-order-rel}'\ x\ y = \text{ord-class.lexordp}\ x\ y \rangle$ 
   $\langle \text{term-order-rel}'\ x\ y \longleftrightarrow x < y \rangle$ 
  unfolding less-than-char-of-char[symmetric, abs-def]
  by (auto simp: lexordp-conv-lexord less-eq-list-def
    less-list-def lexordp-def var-order-rel-def
    rel2p-def term-order-rel-alt-def p2rel-def)

```

qed

**lemma** *list-rel-list-rel-order-iff*:

**assumes**  $\langle (a, b) \in \langle \text{string-rel} \rangle \text{list-rel} \rangle \langle (a', b') \in \langle \text{string-rel} \rangle \text{list-rel} \rangle$   
**shows**  $\langle a < a' \iff b < b' \rangle$

**proof**

**have**  $H: \langle (a, b) \in \langle \text{string-rel} \rangle \text{list-rel} \implies$   
 $(a, cs) \in \langle \text{string-rel} \rangle \text{list-rel} \implies b = cs \rangle$  **for**  $cs$   
**using** *single-valued-monom-rel' IS-RIGHT-UNIQUE-string-rel*  
**unfolding** *string2-rel-def*  
**by** (*subst (asm) list-rel-sv-iff[symmetric]*)  
 $(\text{auto simp: single-valued-def})$

**assume**  $\langle a < a' \rangle$

**then consider**

$u \ u' \text{ where } \langle a' = a @ u \# u' \rangle \mid$   
 $u \ aa \ v \ w \ aaa \text{ where } \langle a = u @ aa \# v \rangle \langle a' = u @ aaa \# w \rangle \langle aa < aaa \rangle$   
**by** (*subst (asm) less-list-def*)  
 $(\text{auto simp: lexord-def List.lexordp-def}$   
 $\text{list-rel-append1 list-rel-split-right-iff})$

**then show**  $\langle b < b' \rangle$

**proof cases**

**case 1**

**then show**  $\langle b < b' \rangle$

**using** *assms*

**by** (*subst less-list-def*)

$(\text{auto simp: lexord-def List.lexordp-def}$   
 $\text{list-rel-append1 list-rel-split-right-iff dest: } H)$

**next**

**case 2**

**then obtain**  $u' \ aa' \ v' \ w' \ aaa'$  **where**

$\langle b = u' @ aa' \# v' \rangle \langle b' = u' @ aaa' \# w' \rangle$

$\langle (aa, aa') \in \text{string-rel} \rangle$

$\langle (aaa, aaa') \in \text{string-rel} \rangle$

**using** *assms*

**apply** ( $\text{auto simp: lexord-def List.lexordp-def}$   
 $\text{list-rel-append1 list-rel-split-right-iff dest: } H)$

**by** (*metis (no-types, hide-lams) H list-rel-append2 list-rel-simp(4)*)

**with**  $\langle aa < aaa \rangle$  **have**  $\langle aa' < aaa' \rangle$

**by** ( $\text{auto simp: string-rel-def less-literal.rep-eq less-list-def}$   
 $\text{lexordp-conv-lexord lexordp-def char.lexordp-conv-lexord}$   
 $\text{simp flip: lexord-code less-char-def}$   
 $\text{PAC-Polynoms-Term.less-char-def})$

**then show**  $\langle b < b' \rangle$

**using**  $\langle b = u' @ aa' \# v' \rangle \langle b' = u' @ aaa' \# w' \rangle$

**by** (*subst less-list-def*)

$(\text{fastforce simp: lexord-def List.lexordp-def}$   
 $\text{list-rel-append1 list-rel-split-right-iff})$

qed

**next**

**have**  $H: \langle (a, b) \in \langle \text{string-rel} \rangle \text{list-rel} \implies$   
 $(a', b) \in \langle \text{string-rel} \rangle \text{list-rel} \implies a = a' \rangle$  **for**  $a \ a' \ b$

**using** *single-valued-monom-rel'*

**by** ( $\text{auto simp: single-valued-def IS-LEFT-UNIQUE-def}$   
 $\text{simp flip: inv-list-rel-eq})$

**assume**  $\langle b < b' \rangle$



```

then consider
  u u' where  $\langle b' = b @ u \# u' \rangle \mid$ 
  u aa v w aaa where  $\langle b = u @ aa \# v \rangle \langle b' = u @ aaa \# w \rangle \langle aa < aaa \rangle$ 
  by (subst (asm) less-list-def)
  (auto simp: lexord-def List.lexordp-def
    list-rel-append1 list-rel-split-right-iff)
then show  $\langle a < a' \rangle$ 
proof cases
  case 1
  then show  $\langle a < a' \rangle$ 
  using assms
  by (subst less-list-def)
  (auto simp: lexord-def List.lexordp-def
    list-rel-append2 list-rel-split-left-iff dest: H)
next
  case 2
  then obtain u' aa' v' w' aaa' where
     $\langle a = u' @ aa' \# v' \rangle \langle a' = u' @ aaa' \# w' \rangle$ 
     $\langle aa', aa \rangle \in \text{string-rel}$ 
     $\langle aaa', aaa \rangle \in \text{string-rel}$ 
  using assms
  by (auto simp: lexord-def List.lexordp-def
    list-rel-append2 list-rel-split-left-iff dest: H)
  with  $\langle aa < aaa \rangle$  have  $\langle aa' < aaa' \rangle$ 
  by (auto simp: string-rel-def less-literal.rep-eq less-list-def
    lexordp-conv-lexord lexordp-def char.lexordp-conv-lexord
    simp flip: lexord-code less-char-def
    PAC-Polynoms-Term.less-char-def)
  then show  $\langle a < a' \rangle$ 
  using  $\langle a = u' @ aa' \# v' \rangle \langle a' = u' @ aaa' \# w' \rangle$ 
  by (subst less-list-def)
  (fastforce simp: lexord-def List.lexordp-def
    list-rel-append1 list-rel-split-right-iff)
qed
qed

```

```

lemma string-rel-le[sepref-import-param]:
  shows  $\langle ((<), (<)) \in \langle \text{string-rel} \rangle \text{list-rel} \rightarrow \langle \text{string-rel} \rangle \text{list-rel} \rightarrow \text{bool-rel} \rangle$ 
  by (auto intro!: fun-relI simp: list-rel-list-rel-order-iff)

```

```

lemma [sepref-import-param]:
  assumes  $\langle \text{CONSTRAINT IS-LEFT-UNIQUE } R \rangle \langle \text{CONSTRAINT IS-RIGHT-UNIQUE } R \rangle$ 
  shows  $\langle (\text{remove1}, \text{remove1}) \in R \rightarrow \langle R \rangle \text{list-rel} \rightarrow \langle R \rangle \text{list-rel} \rangle$ 
  apply (intro fun-relI)
  subgoal premises p for x y xs ys
  using p(2) p(1) assms
  by (induction xs ys rule: list-rel-induct)
  (auto simp: IS-LEFT-UNIQUE-def single-valued-def)
done

```

```

instantiation pac-step :: (heap, heap, heap) heap
begin

```

```

instance
proof standard
  obtain f :: ⟨'a ⇒ nat⟩ where
    f: ⟨inj f⟩
    by blast
  obtain g :: ⟨nat × nat × nat × nat × nat ⇒ nat⟩ where
    g: ⟨inj g⟩
    by blast
  obtain h :: ⟨'b ⇒ nat⟩ where
    h: ⟨inj h⟩
    by blast
  obtain i :: ⟨'c ⇒ nat⟩ where
    i: ⟨inj i⟩
    by blast
  have [iff]: ⟨g a = g b ⟷ a = b⟩⟨h a'' = h b'' ⟷ a'' = b''⟩ ⟨f a' = f b' ⟷ a' = b'⟩
    ⟨i a''' = i b''' ⟷ a''' = b'''⟩ for a b a' b' a'' b'' a''' b'''
    using f g h i unfolding inj-def by blast+
  let ?f = ⟨λx :: ('a, 'b, 'c) pac-step.
    g (case x of
      | Add a b c d ⇒ (0, i a, i b, i c, f d)
      | Del a ⇒ (1, i a, 0, 0, 0)
      | Mult a b c d ⇒ (2, i a, f b, i c, f d)
      | Extension a b c ⇒ (3, i a, f c, 0, h b))⟩
  have ⟨inj ?f⟩
  apply (auto simp: inj-def)
  apply (case-tac x; case-tac y)
  apply auto
  done
  then show ⟨∃ f :: ('a, 'b, 'c) pac-step ⇒ nat. inj f⟩
  by blast
qed

end

end

theory PAC-Checker-Init
  imports PAC-Checker WB-Sort PAC-Checker-Relation
begin

```

## 12 Initial Normalisation of Polynoms

### 12.1 Sorting

Adapted from the theory *HOL-ex.MergeSort* by Tobias. We did not change much, but we refine it to executable code and try to improve efficiency.

```

fun merge :: - ⇒ 'a list ⇒ 'a list ⇒ 'a list
where
  merge f (x#xs) (y#ys) =
    (if f x y then x # merge f xs (y#ys) else y # merge f (x#xs) ys)
| merge f xs [] = xs
| merge f [] ys = ys

lemma mset-merge [simp]:
  mset (merge f xs ys) = mset xs + mset ys

```

```

by (induct f xs ys rule: merge.induct) (simp-all add: ac-simps)

lemma set-merge [simp]:
  set (merge f xs ys) = set xs ∪ set ys
by (induct f xs ys rule: merge.induct) auto

lemma sorted-merge:
  transp f  $\implies$  ( $\bigwedge x y. f x y \vee f y x$ )  $\implies$ 
    sorted-wrt f (merge f xs ys)  $\longleftrightarrow$  sorted-wrt f xs  $\wedge$  sorted-wrt f ys
  apply (induct f xs ys rule: merge.induct)
  apply (auto simp add: ball-Un not-le less-le dest: transpD)
  apply blast
  apply (blast dest: transpD)
done

fun msort ::  $\alpha \Rightarrow 'a \text{ list} \Rightarrow 'a \text{ list}$ 
where
  msort f [] = []
| msort f [x] = [x]
| msort f xs = merge f
  (msort f (take (size xs div 2) xs))
  (msort f (drop (size xs div 2) xs))

fun swap-ternary ::  $\langle \alpha \Rightarrow \text{nat} \Rightarrow \text{nat} \Rightarrow ('a \times 'a \times 'a) \Rightarrow ('a \times 'a \times 'a) \rangle$  where
  swap-ternary f m n =
    (if (m = 0  $\wedge$  n = 1)
     then ( $\lambda(a, b, c). \text{if } f a b \text{ then } (a, b, c)$ 
          else (b,a,c))
    else if (m = 0  $\wedge$  n = 2)
     then ( $\lambda(a, b, c). \text{if } f a c \text{ then } (a, b, c)$ 
          else (c,b,a))
    else if (m = 1  $\wedge$  n = 2)
     then ( $\lambda(a, b, c). \text{if } f b c \text{ then } (a, b, c)$ 
          else (a,c,b))
    else ( $\lambda(a, b, c). (a,b,c)$ ))

fun msort2 ::  $\alpha \Rightarrow 'a \text{ list} \Rightarrow 'a \text{ list}$ 
where
  msort2 f [] = []
| msort2 f [x] = [x]
| msort2 f [x,y] = (if f x y then [x,y] else [y,x])
| msort2 f xs = merge f
  (msort f (take (size xs div 2) xs))
  (msort f (drop (size xs div 2) xs))

lemmas [code del] =
  msort2.simps

declare msort2.simps[simp del]
lemmas [code] =
  msort2.simps[unfolded swap-ternary.simps, simplified]

declare msort2.simps[simp]

lemma msort-msort2:

```

```

fixes  $xs :: \langle 'a :: \text{linorder list} \rangle$ 
shows  $\langle \text{msort } (\leq) \text{ } xs = \text{msort2 } (\leq) \text{ } xs \rangle$ 
apply (induction  $\langle (\leq) :: 'a \Rightarrow 'a \Rightarrow \text{bool} \rangle$   $xs$  rule:  $\text{msort2.induct}$ )
apply (auto dest:  $\text{transpD}$ )
done

```

```

lemma sorted-msort:
   $\text{transp } f \implies (\bigwedge x y. f x y \vee f y x) \implies$ 
     $\text{sorted-wrt } f \text{ } (\text{msort } f \text{ } xs)$ 
by (induct  $f \text{ } xs$  rule:  $\text{msort.induct}$ ) (simp-all add:  $\text{sorted-merge}$ )

```

```

lemma mset-msort[simp]:
   $\text{mset } (\text{msort } f \text{ } xs) = \text{mset } xs$ 
by (induct  $f \text{ } xs$  rule:  $\text{msort.induct}$ )
  (simp-all, metis  $\text{append-take-drop-id mset.simps(2) mset-append}$ )

```

## 12.2 Sorting applied to monomials

```

lemma merge-coeffs-alt-def:
   $\langle (\text{RETURN } o \text{ merge-coeffs}) \text{ } p =$ 
     $\text{RECT}_T(\lambda f p.$ 
      (case  $p$  of
         $[] \Rightarrow \text{RETURN } []$ 
      |  $[-] \Rightarrow \text{RETURN } p$ 
      |  $((xs, n) \# (ys, m) \# p) \Rightarrow$ 
        (if  $xs = ys$ 
          then if  $n + m \neq 0$  then  $f ((xs, n + m) \# p)$  else  $f p$ 
          else do  $\{p \leftarrow f ((ys, m) \# p); \text{RETURN } ((xs, n) \# p)\}$ )))
     $p \rangle$ 
apply (induction  $p$  rule:  $\text{merge-coeffs.induct}$ )
subgoal by (subst  $\text{RECT-unfold}$ , refine-mono) auto
subgoal by (subst  $\text{RECT-unfold}$ , refine-mono) auto
subgoal for  $x \text{ } p \text{ } y \text{ } q$ 
  by (subst  $\text{RECT-unfold}$ , refine-mono)
  (smt case-prod-conv list.simps(5) merge-coeffs.simps(3) nres-monad1
    push-in-let-conv(2))
done

```

```

lemma hn-invalid-recover:
   $\langle \text{is-pure } R \implies \text{hn-invalid } R = (\lambda x y. R x y * \text{true}) \rangle$ 
   $\langle \text{is-pure } R \implies \text{invalid-assn } R = (\lambda x y. R x y * \text{true}) \rangle$ 
by (auto simp:  $\text{is-pure-conv invalid-pure-recover hn-ctxt-def intro! ext}$ )

```

```

lemma safe-poly-vars:
shows
  [safe-constraint-rules]:
     $\text{is-pure } (\text{poly-assn})$  and
  [safe-constraint-rules]:
     $\text{is-pure } (\text{monom-assn})$  and
  [safe-constraint-rules]:
     $\text{is-pure } (\text{monomial-assn})$  and
  [safe-constraint-rules]:
     $\text{is-pure string-assn}$ 
by (auto intro!: pure-prod list-assn-pure simp: prod-assn-pure-conv)

```

```

lemma invalid-assn-distrib:

```

```

⟨invalid-assn monom-assn ×a invalid-assn int-assn = invalid-assn (monom-assn ×a int-assn)⟩
  apply (simp add: invalid-pure-recover hn-invalid-recover
    safe-constraint-rules)
  apply (subst hn-invalid-recover)
  apply (rule safe-poly-vars(2))
  apply (subst hn-invalid-recover)
  apply (rule safe-poly-vars)
  apply (auto intro!: ext)
done

```

**lemma** *WTF-RF-recover*:

```

⟨hn-ctxt (invalid-assn monom-assn ×a invalid-assn int-assn) xb
  x'a ∨A
  hn-ctxt monomial-assn xb x'a ⟹t
  hn-ctxt (monomial-assn) xb x'a⟩
by (smt assn-aci(5) hn-ctxt-def invalid-assn-distrib invalid-pure-recover is-pure-conv
  merge-thms(4) merge-true-star reorder-enttI safe-poly-vars(3) star-aci(2) star-aci(3))

```

**lemma** *WTF-RF*:

```

⟨hn-ctxt (invalid-assn monom-assn ×a invalid-assn int-assn) xb x'a *
  (hn-invalid poly-assn la l'a * hn-invalid int-assn a2' a2 *
    hn-invalid monom-assn a1' a1 *
    hn-invalid poly-assn l l' *
    hn-invalid monomial-assn xa x' *
    hn-invalid poly-assn ax px) ⟹t
  hn-ctxt (monomial-assn) xb x'a *
  hn-ctxt poly-assn
    la l'a *
    hn-ctxt poly-assn l l' *
    (hn-invalid int-assn a2' a2 *
      hn-invalid monom-assn a1' a1 *
      hn-invalid monomial-assn xa x' *
      hn-invalid poly-assn ax px)⟩
⟨hn-ctxt (invalid-assn monom-assn ×a invalid-assn int-assn) xa x' *
  (hn-ctxt poly-assn l l' * hn-invalid poly-assn ax px) ⟹t
  hn-ctxt (monomial-assn) xa x' *
  hn-ctxt poly-assn l l' *
  hn-ctxt poly-assn ax px *
  emp⟩
by sepref-dbg-trans-step+

```

The refinement frameword is completely lost here when synthesizing the constants – it does not understant what is pure (actually everything) and what must be destroyed.

**sepref-definition** *merge-coeffs-impl*

```

is ⟨RETURN o merge-coeffs⟩
:: ⟨poly-assnd →a poly-assn⟩
supply [[goals-limit=1]]
unfolding merge-coeffs-alt-def
  HOL-list.fold-custom-empty poly-assn-alt-def
apply (rewrite in ⟨·⟩ annotate-assn[where A=⟨poly-assn⟩])
apply sepref-dbg-preproc
apply sepref-dbg-cons-init
apply sepref-dbg-id
apply sepref-dbg-monadify
apply sepref-dbg-opt-init

```

```

apply (rule WTF-RF | sepref-dbg-trans-step)+
apply sepref-dbg-opt
apply sepref-dbg-cons-solve
apply sepref-dbg-cons-solve
apply sepref-dbg-constraints
done

```

**definition** *full-quicksort-poly* **where**

```

⟨full-quicksort-poly = full-quicksort-ref (λx y. x = y ∨ (x, y) ∈ term-order-rel) fst⟩

```

**lemma** *down-eq-id-list-rel*:  $\Downarrow(\langle Id \rangle list\text{-}rel) \ x = x$

**by** *auto*

**definition** *quicksort-poly*::  $\langle nat \Rightarrow nat \Rightarrow llist\text{-}polynom \Rightarrow (llist\text{-}polynom) \ nres \rangle$  **where**

```

⟨quicksort-poly x y z = quicksort-ref (≤) fst (x, y, z)⟩

```

**term** *partition-between-ref*

**definition** *partition-between-poly* ::  $\langle nat \Rightarrow nat \Rightarrow llist\text{-}polynom \Rightarrow (llist\text{-}polynom \times nat) \ nres \rangle$  **where**

```

⟨partition-between-poly = partition-between-ref (≤) fst⟩

```

**definition** *partition-main-poly* ::  $\langle nat \Rightarrow nat \Rightarrow llist\text{-}polynom \Rightarrow (llist\text{-}polynom \times nat) \ nres \rangle$  **where**

```

⟨partition-main-poly = partition-main (≤) fst⟩

```

**lemma** *string-list-trans*:

```

⟨(xa :: char list list, ya) ∈ lexord (lexord {(x, y). x < y}) ⟹

```

```

(ya, z) ∈ lexord (lexord {(x, y). x < y}) ⟹

```

```

(xa, z) ∈ lexord (lexord {(x, y). x < y})⟩

```

**by** (smt less-char-def char.less-trans less-than-char-def lexord-partial-trans p2rel-def)

**lemma** *full-quicksort-sort-poly-spec*:

```

⟨(full-quicksort-poly, sort-poly-spec) ∈ ⟨Id⟩list-rel →f ⟨⟨Id⟩list-rel⟩nres-rel⟩

```

**proof** –

**have** *xs*:  $\langle xs, xs \rangle \in \langle Id \rangle list\text{-}rel$  **and**  $\Downarrow(\langle Id \rangle list\text{-}rel) \ x = x$  **for** *x* *xs*

**by** *auto*

**show** *?thesis*

**apply** (intro frefI nres-relI)

**unfolding** *full-quicksort-poly-def*

**apply** (rule full-quicksort-ref-full-quicksort[THEN fref-to-Down-curry, THEN order-trans])

**subgoal**

**by** (auto simp: rel2p-def var-order-rel-def p2rel-def Relation.total-on-def

dest: string-list-trans)

**subgoal**

**using** total-on-lexord-less-than-char-linear[unfolded var-order-rel-def]

**apply** (auto simp: rel2p-def var-order-rel-def p2rel-def Relation.total-on-def less-char-def)

**done**

**subgoal by fast**

**apply** (rule xs)

**apply** (subst down-eq-id-list-rel)

**unfolding** sorted-wrt-map sort-poly-spec-def

**apply** (rule full-quicksort-correct-sorted[where  $R = \langle \lambda x y. x = y \vee (x, y) \in term\text{-}order\text{-}rel \rangle$  **and**

$h = \langle fst \rangle$ ,

THEN order-trans])

**subgoal**

**by** (auto simp: rel2p-def var-order-rel-def p2rel-def Relation.total-on-def dest: string-list-trans)

```

subgoal for x y
  using total-on-lexord-less-than-char-linear[unfolded var-order-rel-def]
  apply (auto simp: rel2p-def var-order-rel-def p2rel-def Relation.total-on-def
    less-char-def)
  done
subgoal
  by (auto simp: rel2p-def p2rel-def)
done
qed

```

## 12.3 Lifting to polynomials

**definition** *merge-sort-poly* ::  $\langle \cdot \rangle$  **where**  
 $\langle \text{merge-sort-poly} = \text{msort } (\lambda a b. \text{fst } a \leq \text{fst } b) \rangle$

**definition** *merge-monoms-poly* ::  $\langle \cdot \rangle$  **where**  
 $\langle \text{merge-monoms-poly} = \text{msort } (\leq) \rangle$

**definition** *merge-poly* ::  $\langle \cdot \rangle$  **where**  
 $\langle \text{merge-poly} = \text{merge } (\lambda a b. \text{fst } a \leq \text{fst } b) \rangle$

**definition** *merge-monoms* ::  $\langle \cdot \rangle$  **where**  
 $\langle \text{merge-monoms} = \text{merge } (\leq) \rangle$

**definition** *msort-poly-impl* ::  $\langle (\text{String.literal list} \times \text{int}) \text{ list} \Rightarrow \cdot \rangle$  **where**  
 $\langle \text{msort-poly-impl} = \text{msort } (\lambda a b. \text{fst } a \leq \text{fst } b) \rangle$

**definition** *msort-monoms-impl* ::  $\langle (\text{String.literal list}) \Rightarrow \cdot \rangle$  **where**  
 $\langle \text{msort-monoms-impl} = \text{msort } (\leq) \rangle$

**lemma** *msort-poly-impl-alt-def*:

```

 $\langle \text{msort-poly-impl } xs =$ 
  (case xs of
    []  $\Rightarrow$  []
  | [a]  $\Rightarrow$  [a]
  | [a,b]  $\Rightarrow$  if  $\text{fst } a \leq \text{fst } b$  then [a,b] else [b,a]
  | xs  $\Rightarrow$  merge-poly
    (msort-poly-impl (take ((length xs) div 2) xs))
    (msort-poly-impl (drop ((length xs) div 2) xs)))
unfolding msort-poly-impl-def
apply (auto split: list.splits simp: merge-poly-def)
done

```

**lemma** *le-term-order-rel'*:

```

 $\langle (\leq) = (\lambda x y. x = y \vee \text{term-order-rel}' x y) \rangle$ 
apply (intro ext)
apply (auto simp add: less-list-def less-eq-list-def
  lexordp-eq-conv-lexord lexordp-def)
using term-order-rel'-alt-def-lexord term-order-rel'-def apply blast
using term-order-rel'-alt-def-lexord term-order-rel'-def apply blast
done

```

**fun** *lexord-eq* **where**

```

 $\langle \text{lexord-eq } [] - = \text{True} \rangle$  |
 $\langle \text{lexord-eq } (x \# xs) (y \# ys) = (x < y \vee (x = y \wedge \text{lexord-eq } xs ys)) \rangle$  |
 $\langle \text{lexord-eq } - - = \text{False} \rangle$ 

```

```

lemma [simp]:
  ⟨lexord-eq [] [] = True⟩
  ⟨lexord-eq (a # b) [] = False⟩
  ⟨lexord-eq [] (a # b) = True⟩
apply auto
done

lemma var-order-rel':
  ⟨(≤) = (λx y. x = y ∨ (x,y) ∈ var-order-rel)⟩
by (intro ext)
  (auto simp add: less-list-def less-eq-list-def
    lexordp-eq-conv-lexord lexordp-def var-order-rel-def
    lexordp-conv-lexord p2rel-def)

lemma var-order-rel'':
  ⟨(x,y) ∈ var-order-rel ⟷ x < y⟩
by (metis leD less-than-char-linear lexord-linear neq-iff var-order-rel' var-order-rel-antisym var-order-rel-def)

lemma lexord-eq-alt-def1:
  ⟨a ≤ b = lexord-eq a b⟩ for a b :: ⟨String.literal list⟩
unfolding le-term-order-rel'
apply (induction a b rule: lexord-eq.induct)
apply (auto simp: var-order-rel'' less-eq-list-def)
done

lemma lexord-eq-alt-def2:
  ⟨(RETURN oo lexord-eq) xs ys =
    RECT (λf (xs, ys).
      case (xs, ys) of
        ([], -) ⇒ RETURN True
      | (x # xs, y # ys) ⇒
        if x < y then RETURN True
        else if x = y then f (xs, ys) else RETURN False
      | - ⇒ RETURN False)
    (xs, ys)⟩
apply (subst eq-commute)
apply (induction xs ys rule: lexord-eq.induct)
subgoal by (subst RECT-unfold, refine-mono) auto
subgoal by (subst RECT-unfold, refine-mono) auto
subgoal by (subst RECT-unfold, refine-mono) auto
done

definition var-order' where
  [simp]: ⟨var-order' = var-order⟩

lemma var-order-rel[def-pat-rules]:
  ⟨(∈)$(x,y)$var-order-rel ≡ var-order'$(x,y)$⟩
by (auto simp: p2rel-def rel2p-def)

lemma var-order-rel-alt-def:
  ⟨var-order-rel = p2rel char.lexordp⟩
apply (auto simp: p2rel-def char.lexordp-conv-lexord var-order-rel-def)

```



```

using char.lexordp-conv-lexord apply auto
done

lemma var-order-rel-var-order:
   $\langle (x, y) \in \text{var-order-rel} \longleftrightarrow \text{var-order } x \ y \rangle$ 
  by (auto simp: rel2p-def)

lemma var-order-string-le[sepref-import-param]:
   $\langle ((<), \text{var-order}') \in \text{string-rel} \rightarrow \text{string-rel} \rightarrow \text{bool-rel} \rangle$ 
  apply (auto intro!: freI simp: string-rel-def String.less-literal-def
    rel2p-def linorder.lexordp-conv-lexord[OF char.linorder-axioms,
    unfolded less-eq-char-def] var-order-rel-def
    p2rel-def
    simp flip: PAC-Polynoms-Term.less-char-def)
  using char.lexordp-conv-lexord apply auto
done

lemma [sepref-import-param]:
   $\langle ((\leq), (\leq)) \in \text{monom-rel} \rightarrow \text{monom-rel} \rightarrow \text{bool-rel} \rangle$ 
  apply (intro fun-relI)
  using list-rel-list-rel-order-iff by fastforce

lemma [sepref-import-param]:
   $\langle ((<), (<)) \in \text{string-rel} \rightarrow \text{string-rel} \rightarrow \text{bool-rel} \rangle$ 
  unfolding string-rel-def less-literal.rep-eq less-than-char-def
    less-eq-list-def PAC-Polynoms-Term.less-char-def[symmetric]
  apply (intro fun-relI)
  apply (auto simp: string-rel-def less-literal.rep-eq PAC-Polynoms-Term.less-char-def
    less-list-def char.lexordp-conv-lexord lexordp-eq-refl
    lexord-code lexordp-eq-conv-lexord less-char-def[abs-def])
  apply (metis PAC-Checker-Relation.less-char-def char.lexordp-conv-lexord less-list-def p2rel-def var-order-rel''
    var-order-rel-def)
  apply (metis PAC-Checker-Relation.less-char-def char.lexordp-conv-lexord less-list-def p2rel-def var-order-rel''
    var-order-rel-def)
done

lemma [sepref-import-param]:
   $\langle ((\leq), (\leq)) \in \text{string-rel} \rightarrow \text{string-rel} \rightarrow \text{bool-rel} \rangle$ 
  unfolding string-rel-def less-eq-literal.rep-eq less-than-char-def
    less-eq-list-def PAC-Polynoms-Term.less-char-def[symmetric]
  by (intro fun-relI)
  (auto simp: string-rel-def less-eq-literal.rep-eq less-than-char-def
    less-eq-list-def char.lexordp-eq-conv-lexord lexordp-eq-refl
    lexord-code lexordp-eq-conv-lexord
    simp flip: less-char-def[abs-def])

sepref-register lexord-eq
sepref-definition lexord-eq-term
  is  $\langle \text{uncurry } (\text{RETURN } \text{oo } \text{lexord-eq}) \rangle$ 
   $:: (\text{monom-assn}^k *_a \text{monom-assn}^k \rightarrow_a \text{bool-assn})$ 
  supply [[goals-limit=1]]
  unfolding lexord-eq-alt-def2
  by sepref

declare lexord-eq-term.refine[sepref-fr-rules]

```

**lemmas** [code del] = msort-poly-impl-def msort-monoms-impl-def

**lemmas** [code] =

msort-poly-impl-def[unfolded lexord-eq-alt-def1[abs-def]]  
 msort-monoms-impl-def[unfolded msort-msort2]

**lemma** term-order-rel-trans:

⟨ (a, aa) ∈ term-order-rel ⟹  
 (aa, ab) ∈ term-order-rel ⟹ (a, ab) ∈ term-order-rel ⟩  
**by** (metis PAC-Checker-Relation.less-char-def p2rel-def string-list-trans var-order-rel-def)

**lemma** merge-sort-poly-sort-poly-spec:

⟨ (RETURN o merge-sort-poly, sort-poly-spec) ∈ ⟨Id⟩list-rel →<sub>f</sub> ⟨⟨Id⟩list-rel⟩nres-rel ⟩  
**unfolding** sort-poly-spec-def merge-sort-poly-def  
**apply** (intro frefI nres-relI)  
**using** total-on-lexord-less-than-char-linear var-order-rel-def  
**by** (auto intro!: sorted-msort simp: sorted-wrt-map rel2p-def  
 le-term-order-rel' transp-def dest: term-order-rel-trans)

**lemma** msort-alt-def:

⟨ RETURN o (msort f) =  
 RECT (λg xs.  
 case xs of  
 [] ⇒ RETURN []  
 | [x] ⇒ RETURN [x]  
 | - ⇒ do {  
 a ← g (take (size xs div 2) xs);  
 b ← g (drop (size xs div 2) xs);  
 RETURN (merge f a b)} ⟩  
**apply** (intro ext)  
**unfolding** comp-def  
**apply** (induct-tac f x rule: msort.induct)  
**subgoal by** (subst RECT-unfold, refine-mono) auto  
**subgoal by** (subst RECT-unfold, refine-mono) auto  
**subgoal**  
**by** (subst RECT-unfold, refine-mono)  
 (smt let-to-bind-conv list.simps(5) msort.simps(3))  
**done**

**lemma** monomial-rel-order-map:

⟨ (x, a, b) ∈ monomial-rel ⟹  
 (y, aa, bb) ∈ monomial-rel ⟹  
 fst x ≤ fst y ⟷ a ≤ aa ⟩  
**apply** (cases x; cases y)  
**apply** auto  
**using** list-rel-list-rel-order-iff **by** fastforce+

**lemma** step-rewrite-pure:

**fixes** K :: ⟨('olbl × 'lbl) set⟩  
**shows**  
 ⟨ pure (p2rel (⟨K, V, R⟩pac-step-rel-raw)) = pac-step-rel-assn (pure K) (pure V) (pure R) ⟩  
 ⟨ monomial-assn = pure (monom-rel ×<sub>r</sub> int-rel) ⟩ **and**  
 poly-assn-list:

$\langle \text{poly-assn} = \text{pure } (\langle \text{monom-rel} \times_r \text{int-rel} \rangle \text{list-rel}) \rangle$

**subgoal**  
**apply** (intro ext)  
**apply** (case-tac x; case-tac xa)  
**apply** (auto simp: relAPP-def p2rel-def pure-def)  
**done**  
**subgoal** H  
**apply** (intro ext)  
**apply** (case-tac x; case-tac xa)  
**by** (simp add: list-assn-pure-conv)  
**subgoal**  
**unfolding** H  
**by** (simp add: list-assn-pure-conv relAPP-def)  
**done**

**lemma** safe-pac-step-rel-assn[safe-constraint-rules]:  
 $\text{is-pure } K \implies \text{is-pure } V \implies \text{is-pure } R \implies \text{is-pure } (\text{pac-step-rel-assn } K \ V \ R)$   
**by** (auto simp: step-rewrite-pure(1)[symmetric] is-pure-conv)

**lemma** merge-poly-merge-poly:  
 $\langle (\text{merge-poly}, \text{merge-poly})$   
 $\in \text{poly-rel} \rightarrow \text{poly-rel} \rightarrow \text{poly-rel} \rangle$   
**unfolding** merge-poly-def  
**apply** (intro fun-relI)  
**subgoal for** a a' aa a'a  
**apply** (induction  $\langle (\lambda(a :: \text{String.literal list} \times \text{int})$   
 $(b :: \text{String.literal list} \times \text{int}). \text{fst } a \leq \text{fst } b) \rangle \ a \ aa$   
arbitrary: a' a'a  
rule: merge.induct)  
**subgoal**  
**by** (auto elim!: list-relE3 list-relE4 list-relE list-relE2  
simp: monomial-rel-order-map)  
**subgoal**  
**by** (auto elim!: list-relE3 list-relE)  
**subgoal**  
**by** (auto elim!: list-relE3 list-relE4 list-relE list-relE2)  
**done**  
**done**

**lemmas** [fcomp-norm-unfold] =  
poly-assn-list[symmetric]  
step-rewrite-pure(1)

**lemma** merge-poly-merge-poly2:  
 $\langle (a, b) \in \text{poly-rel} \implies (a', b') \in \text{poly-rel} \implies$   
 $(\text{merge-poly } a \ a', \text{merge-poly } b \ b') \in \text{poly-rel} \rangle$   
**using** merge-poly-merge-poly  
**unfolding** fun-rel-def  
**by** auto

**lemma** list-rel-takeD:  
 $\langle (a, b) \in \langle R \rangle \text{list-rel} \implies (n, n') \in \text{Id} \implies (\text{take } n \ a, \text{take } n' \ b) \in \langle R \rangle \text{list-rel} \rangle$   
**by** (simp add: list-rel-eq-listrel listrel-iff-nth relAPP-def)

**lemma** *list-rel-dropD*:  
 $\langle (a, b) \in \langle R \rangle \text{list-rel} \implies (n, n') \in \text{Id} \implies (\text{drop } n \ a, \text{drop } n' \ b) \in \langle R \rangle \text{list-rel} \rangle$   
**by** (*simp add: list-rel-eq-listrel listrel-iff-nth relAPP-def*)

**lemma** *merge-sort-poly*[*sepref-import-param*]:  
 $\langle (\text{msort-poly-impl}, \text{merge-sort-poly}) \in \text{poly-rel} \rightarrow \text{poly-rel} \rangle$   
**unfolding** *merge-sort-poly-def msort-poly-impl-def*  
**apply** (*intro fun-relI*)  
**subgoal for** *a a'*  
**apply** (*induction*  $\langle (\lambda(a :: \text{String.literal list} \times \text{int}) (b :: \text{String.literal list} \times \text{int}). \text{fst } a \leq \text{fst } b) \rangle a$   
*arbitrary: a'*  
*rule: msort.induct*)  
**subgoal**  
**by** *auto*  
**subgoal**  
**by** (*auto elim!: list-relE3 list-relE*)  
**subgoal premises** *p*  
**using** *p*  
**by** (*auto elim!: list-relE3 list-relE4 list-relE list-relE2*  
*simp: merge-poly-def[symmetric]*  
*intro!: list-rel-takeD list-rel-dropD*  
*intro!: merge-poly-merge-poly2 p(1)[simplified] p(2)[simplified],*  
*auto simp: list-rel-imp-same-length*)  
**done**  
**done**

**lemmas** [*sepref-fr-rules*] = *merge-sort-poly*[*FCOMP merge-sort-poly-sort-poly-spec*]

**sepref-definition** *partition-main-poly-impl*  
**is**  $\langle \text{uncurry2 } \text{partition-main-poly} \rangle$   
 $:: \langle \text{nat-assn}^k *_a \text{nat-assn}^k *_a \text{poly-assn}^k \rightarrow_a \text{prod-assn poly-assn nat-assn} \rangle$   
**unfolding** *partition-main-poly-def partition-main-def*  
*term-order-rel'-def[symmetric]*  
*term-order-rel'-alt-def*  
*le-term-order-rel'*  
**by** *sepref*

**declare** *partition-main-poly-impl.refine*[*sepref-fr-rules*]

**sepref-definition** *partition-between-poly-impl*  
**is**  $\langle \text{uncurry2 } \text{partition-between-poly} \rangle$   
 $:: \langle \text{nat-assn}^k *_a \text{nat-assn}^k *_a \text{poly-assn}^k \rightarrow_a \text{prod-assn poly-assn nat-assn} \rangle$   
**unfolding** *partition-between-poly-def partition-between-ref-def*  
*partition-main-poly-def[symmetric]*  
**unfolding** *choose-pivot3-def*  
*term-order-rel'-def[symmetric]*  
*term-order-rel'-alt-def choose-pivot-def*  
*lexord-eq-alt-def1*  
**by** *sepref*

**declare** *partition-between-poly-impl.refine*[*sepref-fr-rules*]

**sepref-definition** *quicksort-poly-impl*  
**is**  $\langle \text{uncurry2 } \text{quicksort-poly} \rangle$   
**::**  $\langle \text{nat-assign}^k *_{\alpha} \text{nat-assign}^k *_{\alpha} \text{poly-assign}^k \rightarrow_{\alpha} \text{poly-assign} \rangle$   
**unfolding** *partition-main-poly-def quicksort-ref-def quicksort-poly-def*  
*partition-between-poly-def[symmetric]*  
**by** *sepref*

**lemmas** [*sepref-fr-rules*] = *quicksort-poly-impl.refine*

**sepref-register** *quicksort-poly*  
**sepref-definition** *full-quicksort-poly-impl*  
**is**  $\langle \text{full-quicksort-poly} \rangle$   
**::**  $\langle \text{poly-assign}^k \rightarrow_{\alpha} \text{poly-assign} \rangle$   
**unfolding** *full-quicksort-poly-def full-quicksort-ref-def*  
*quicksort-poly-def[symmetric]*  
*le-term-order-rel'[symmetric]*  
*term-order-rel'-def[symmetric]*  
*List.null-def*  
**by** *sepref*

**lemmas** *sort-poly-spec-hnr* =  
*full-quicksort-poly-impl.refine[FCOMP full-quicksort-sort-poly-spec]*

**declare** *merge-coeffs-impl.refine[sepref-fr-rules]*

**sepref-definition** *normalize-poly-impl*  
**is**  $\langle \text{normalize-poly} \rangle$   
**::**  $\langle \text{poly-assign}^k \rightarrow_{\alpha} \text{poly-assign} \rangle$   
**supply**  $[[\text{goals-limit}=1]]$   
**unfolding** *normalize-poly-def*  
**by** *sepref*

**declare** *normalize-poly-impl.refine[sepref-fr-rules]*

**definition** *full-quicksort-vars* **where**  
 $\langle \text{full-quicksort-vars} = \text{full-quicksort-ref } (\lambda x y. x = y \vee (x, y) \in \text{var-order-rel}) \text{ id} \rangle$

**definition** *quicksort-vars* **::**  $\langle \text{nat} \Rightarrow \text{nat} \Rightarrow \text{string list} \Rightarrow (\text{string list}) \text{ nres} \rangle$  **where**  
 $\langle \text{quicksort-vars } x y z = \text{quicksort-ref } (\leq) \text{ id } (x, y, z) \rangle$

**definition** *partition-between-vars* **::**  $\langle \text{nat} \Rightarrow \text{nat} \Rightarrow \text{string list} \Rightarrow (\text{string list} \times \text{nat}) \text{ nres} \rangle$  **where**  
 $\langle \text{partition-between-vars} = \text{partition-between-ref } (\leq) \text{ id} \rangle$

**definition** *partition-main-vars* **::**  $\langle \text{nat} \Rightarrow \text{nat} \Rightarrow \text{string list} \Rightarrow (\text{string list} \times \text{nat}) \text{ nres} \rangle$  **where**  
 $\langle \text{partition-main-vars} = \text{partition-main } (\leq) \text{ id} \rangle$

**lemma** *total-on-lexord-less-than-char-linear2*:  
 $\langle xs \neq ys \implies (xs, ys) \notin \text{lexord } (\text{less-than-char}) \longleftrightarrow$   
 $(ys, xs) \in \text{lexord } \text{less-than-char} \rangle$   
**using** *lexord-linear[of less-than-char xs ys]*

```

using lexord-linear[of  $\langle \text{less-than-char} \rangle$ ] less-than-char-linear
apply (auto simp: Relation.total-on-def)
using lexord-irrefl[OF irrefl-less-than-char]
  antisym-lexord[OF antisym-less-than-char irrefl-less-than-char]
apply (auto simp: antisym-def)
done

```

**lemma** *string-trans*:

```

 $\langle (xa, ya) \in \text{lexord } \{(x::\text{char}, y::\text{char}). x < y\} \implies$ 
 $(ya, z) \in \text{lexord } \{(x::\text{char}, y::\text{char}). x < y\} \implies$ 
 $(xa, z) \in \text{lexord } \{(x::\text{char}, y::\text{char}). x < y\} \rangle$ 
by (smt less-char-def char.less-trans less-than-char-def lexord-partial-trans p2rel-def)

```

**lemma** *full-quicksort-sort-vars-spec*:

```

 $\langle (\text{full-quicksort-vars}, \text{sort-coeff}) \in \langle \text{Id} \rangle \text{list-rel} \rightarrow_f \langle \langle \text{Id} \rangle \text{list-rel} \rangle \text{nres-rel} \rangle$ 

```

**proof** –

```

have xs:  $\langle (xs, xs) \in \langle \text{Id} \rangle \text{list-rel} \rangle$  and  $\langle \Downarrow(\langle \text{Id} \rangle \text{list-rel}) \ x = x \rangle$  for x xs
by auto

```

**show** *?thesis*

```

apply (intro frefI nres-relI)

```

```

unfolding full-quicksort-vars-def

```

```

apply (rule full-quicksort-ref-full-quicksort[THEN fref-to-Down-curry, THEN order-trans])

```

**subgoal**

```

by (auto simp: rel2p-def var-order-rel-def p2rel-def Relation.total-on-def
  dest: string-trans)

```

**subgoal**

```

using total-on-lexord-less-than-char-linear2[unfolded var-order-rel-def]

```

```

apply (auto simp: rel2p-def var-order-rel-def p2rel-def Relation.total-on-def less-char-def)

```

**done**

**subgoal by** *fast*

```

apply (rule xs)

```

```

apply (subst down-eq-id-list-rel)

```

```

unfolding sorted-wrt-map sort-coeff-def

```

```

apply (rule full-quicksort-correct-sorted[where  $R = \langle (\lambda x y. x = y \vee (x, y) \in \text{var-order-rel}) \rangle$  and  $h$ 
  =  $\langle \text{id} \rangle$ ,

```

```

  THEN order-trans])

```

**subgoal**

```

by (auto simp: rel2p-def var-order-rel-def p2rel-def Relation.total-on-def dest: string-trans)

```

**subgoal for** *x y*

```

using total-on-lexord-less-than-char-linear2[unfolded var-order-rel-def]

```

```

by (auto simp: rel2p-def var-order-rel-def p2rel-def Relation.total-on-def
  less-char-def)

```

**subgoal**

```

by (auto simp: rel2p-def p2rel-def rel2p-def[abs-def])

```

**done**

**qed**

**sempref-definition** *partition-main-vars-impl*

```

is  $\langle \text{uncurry2 } \text{partition-main-vars} \rangle$ 

```

```

 $:: (\text{nat-assn}^k *_{\text{a}} \text{nat-assn}^k *_{\text{a}} (\text{monom-assn})^k \rightarrow_{\text{a}} \text{prod-assn } (\text{monom-assn}) \text{ nat-assn})$ 

```

```

unfolding partition-main-vars-def partition-main-def

```

```

  var-order-rel-var-order

```

```

  var-order'-def[symmetric]

```

```

  term-order-rel'-alt-def

```

$le-term-order-rel'$   
 $id-apply$   
**by**  $sepref$

**declare**  $partition-main-vars-impl.refine[sepref-fr-rules]$

**sepref-definition**  $partition-between-vars-impl$

**is**  $\langle uncurry2\ partition-between-vars \rangle$   
 $:: \langle nat-assn^k *_{\alpha} nat-assn^k *_{\alpha} monom-assn^k \rightarrow_{\alpha} prod-assn\ monom-assn\ nat-assn \rangle$   
**unfolding**  $partition-between-vars-def\ partition-between-ref-def$   
 $partition-main-vars-def[symmetric]$   
**unfolding**  $choose-pivot3-def$   
 $term-order-rel'-def[symmetric]$   
 $term-order-rel'-alt-def\ choose-pivot-def$   
 $le-term-order-rel'\ id-apply$   
**by**  $sepref$

**declare**  $partition-between-vars-impl.refine[sepref-fr-rules]$

**sepref-definition**  $quicksort-vars-impl$

**is**  $\langle uncurry2\ quicksort-vars \rangle$   
 $:: \langle nat-assn^k *_{\alpha} nat-assn^k *_{\alpha} monom-assn^k \rightarrow_{\alpha} monom-assn \rangle$   
**unfolding**  $partition-main-vars-def\ quicksort-ref-def\ quicksort-vars-def$   
 $partition-between-vars-def[symmetric]$   
**by**  $sepref$

**lemmas**  $[sepref-fr-rules] = quicksort-vars-impl.refine$

**sepref-register**  $quicksort-vars$

**lemma**  $le-var-order-rel$ :

$\langle (\leq) = (\lambda x\ y.\ x = y \vee (x, y) \in var-order-rel) \rangle$   
**by**  $(intro\ ext)$   
 $(auto\ simp\ add: less-list-def\ less-eq-list-def\ rel2p-def$   
 $p2rel-def\ lexordp-conv-lexord\ p2rel-def\ var-order-rel-def$   
 $lexordp-eq-conv-lexord\ lexordp-def)$

**sepref-definition**  $full-quicksort-vars-impl$

**is**  $\langle full-quicksort-vars \rangle$   
 $:: \langle monom-assn^k \rightarrow_{\alpha} monom-assn \rangle$   
**unfolding**  $full-quicksort-vars-def\ full-quicksort-ref-def$   
 $quicksort-vars-def[symmetric]$   
 $le-var-order-rel[symmetric]$   
 $term-order-rel'-def[symmetric]$   
 $List.null-def$   
**by**  $sepref$

**lemmas**  $sort-vars-spec-hnr =$

$full-quicksort-vars-impl.refine[FCOMP\ full-quicksort-sort-vars-spec]$

**lemma**  $string-rel-order-map$ :

$\langle (x, a) \in string-rel \implies$   
 $(y, aa) \in string-rel \implies$

$x \leq y \longleftrightarrow a \leq aa$   
**unfolding** *string-rel-def less-eq-literal.rep-eq less-than-char-def*  
 $less-eq-list-def \text{ PAC-Polynoms-Term.less-char-def[symmetric]}$   
**by** (*auto simp: string-rel-def less-eq-literal.rep-eq less-than-char-def*  
 $less-eq-list-def \text{ char.lexordp-eq-conv-lexord lexordp-eq-refl}$   
 $lexord-code \text{ lexordp-eq-conv-lexord}$   
 $simp \text{ flip: less-char-def[abs-def]}$ )  
**lemma** *merge-monoms-merge-monoms*:  
 $\langle (merge-monoms, merge-monoms) \in monom-rel \rightarrow monom-rel \rightarrow monom-rel \rangle$   
**unfolding** *merge-monoms-def*  
**apply** (*intro fun-relI*)  
**subgoal for**  $a \ a' \ aa \ a'a$   
**apply** (*induction*  $\langle (\lambda(a :: String.literal)$   
 $(b :: String.literal). a \leq b) \rangle a \ aa$   
*arbitrary: a' a'a*  
*rule: merge.induct*)  
**subgoal**  
**by** (*auto elim!: list-relE3 list-relE4 list-relE list-relE2*  
 $simp: string-rel-order-map$ )  
**subgoal**  
**by** (*auto elim!: list-relE3 list-relE*)  
**subgoal**  
**by** (*auto elim!: list-relE3 list-relE4 list-relE list-relE2*)  
**done**  
**done**

**lemma** *merge-monoms-merge-monoms2*:  
 $\langle (a, b) \in monom-rel \implies (a', b') \in monom-rel \implies$   
 $(merge-monoms \ a \ a', merge-monoms \ b \ b') \in monom-rel \rangle$   
**using** *merge-monoms-merge-monoms*  
**unfolding** *fun-rel-def merge-monoms-def*  
**by** *auto*

**lemma** *msort-monoms-impl*:  
 $\langle (msort-monoms-impl, merge-monoms-poly)$   
 $\in monom-rel \rightarrow monom-rel \rangle$   
**unfolding** *msort-monoms-impl-def merge-monoms-poly-def*  
**apply** (*intro fun-relI*)  
**subgoal for**  $a \ a'$   
**apply** (*induction*  $\langle (\lambda(a :: String.literal)$   
 $(b :: String.literal). a \leq b) \rangle a$   
*arbitrary: a'*  
*rule: msort.induct*)  
**subgoal**  
**by** *auto*  
**subgoal**  
**by** (*auto elim!: list-relE3 list-relE*)  
**subgoal premises**  $p$   
**using**  $p$   
**by** (*auto elim!: list-relE3 list-relE4 list-relE list-relE2*  
 $simp: merge-monoms-def[symmetric] \text{ intro!: list-rel-takeD list-rel-dropD}$   
 $\text{intro!: merge-monoms-merge-monoms2 } p(1)[simplified] \ p(2)[simplified]$   
 $(simp-all \text{ add: list-rel-imp-same-length})$ )



```

done
done

lemma merge-sort-monoms-sort-monoms-spec:
  ⟨(RETURN o merge-monoms-poly, sort-coeff) ∈ ⟨Id⟩list-rel →f ⟨⟨Id⟩list-rel⟩nres-rel⟩
  unfolding merge-monoms-poly-def sort-coeff-def
  by (intro frefI nres-relI)
    (auto intro!: sorted-msort simp: sorted-wrt-map rel2p-def
      le-term-order-rel' transp-def rel2p-def[abs-def]
      simp flip: le-var-order-rel)

sepref-register sort-coeff
lemma [sepref-fr-rules]:
  ⟨(return o msort-monoms-impl, sort-coeff) ∈ monom-assnk →a monom-assn⟩
  using msort-monoms-impl[sepref-param, FCOMP merge-sort-monoms-sort-monoms-spec]
  by auto

sepref-definition sort-all-coeffs-impl
  is ⟨sort-all-coeffs⟩
  :: ⟨poly-assnk →a poly-assn⟩
  unfolding sort-all-coeffs-def
    HOL-list.fold-custom-empty
  by sepref

declare sort-all-coeffs-impl.refine[sepref-fr-rules]

lemma merge-coeffs0-alt-def:
  ⟨(RETURN o merge-coeffs0) p =
    RECT(λf p.
      (case p of
        [] ⇒ RETURN []
      | [p] => if snd p = 0 then RETURN [] else RETURN [p]
      | ((xs, n) # (ys, m) # p) ⇒
        (if xs = ys
          then if n + m ≠ 0 then f ((xs, n + m) # p) else f p
          else if n = 0 then
            do {p ← f ((ys, m) # p);
              RETURN p}
          else do {p ← f ((ys, m) # p);
              RETURN ((xs, n) # p}))))
    p)
  apply (subst eq-commute)
  apply (induction p rule: merge-coeffs0.induct)
  subgoal by (subst RECT-unfold, refine-mono) auto
  subgoal by (subst RECT-unfold, refine-mono) auto
  subgoal by (subst RECT-unfold, refine-mono) (auto simp: let-to-bind-conv)
done

```

Again, Sepref does not understand what is going here.

```

sepref-definition merge-coeffs0-impl
  is ⟨RETURN o merge-coeffs0⟩
  :: ⟨poly-assnk →a poly-assn⟩
  supply [[goals-limit=1]]
  unfolding merge-coeffs0-alt-def
    HOL-list.fold-custom-empty

```

```

apply sepref-dbg-preproc
apply sepref-dbg-cons-init
apply sepref-dbg-id
apply sepref-dbg-monadify
apply sepref-dbg-opt-init
apply (rule WTF-RF | sepref-dbg-trans-step)+
apply sepref-dbg-opt
apply sepref-dbg-cons-solve
apply sepref-dbg-cons-solve
apply sepref-dbg-constraints
done

```

```

declare merge-coeffs0-impl.refine[sepref-fr-rules]

```

```

sepref-definition fully-normalize-poly-impl
  is  $\langle \text{full-normalize-poly} \rangle$ 
  ::  $\langle \text{poly-assn}^k \rightarrow_a \text{poly-assn} \rangle$ 
  supply [[goals-limit=1]]
  unfolding full-normalize-poly-def
  by sepref

```

```

declare fully-normalize-poly-impl.refine[sepref-fr-rules]

```

```

end
theory PAC-Version
  imports Main
begin

```

This code was taken from IsaFoR and adapted to git.

```

local-setup (
  let
    val version = 2020-AFP
  (*   trim-line (#1 (Isabelle-System.bash-output (cd $ISAFOL/ && git rev-parse --short HEAD ||
echo unknown))) *)
  in
    Local-Theory.define
      ((binding  $\langle \text{version} \rangle$ , NoSyn),
       ((binding  $\langle \text{version-def} \rangle$ , []), HOLogic.mk-literal version)) #> #2
    end
)

```

```

declare version-def [code]

```

```

end
theory PAC-Checker-Synthesis
  imports PAC-Checker WB-Sort PAC-Checker-Relation
  PAC-Checker-Init More-Loops PAC-Version
begin

```

## 13 Code Synthesis of the Complete Checker

We here combine refine the full checker, using the initialisation provided in another file.

**abbreviation** *vars-assn* **where**

$\langle \text{vars-assn} \equiv \text{hs.assn string-assn} \rangle$

**fun** *vars-of-monom-in* **where**

$\langle \text{vars-of-monom-in } [] = \text{True} \rangle \mid$

$\langle \text{vars-of-monom-in } (x \# xs) \mathcal{V} \longleftrightarrow x \in \mathcal{V} \wedge \text{vars-of-monom-in } xs \mathcal{V} \rangle$

**fun** *vars-of-poly-in* **where**

$\langle \text{vars-of-poly-in } [] = \text{True} \rangle \mid$

$\langle \text{vars-of-poly-in } ((x, -) \# xs) \mathcal{V} \longleftrightarrow \text{vars-of-monom-in } x \mathcal{V} \wedge \text{vars-of-poly-in } xs \mathcal{V} \rangle$

**lemma** *vars-of-monom-in-alt-def*:

$\langle \text{vars-of-monom-in } xs \mathcal{V} \longleftrightarrow \text{set } xs \subseteq \mathcal{V} \rangle$

**by** (*induction xs*)

*auto*

**lemma** *vars-llist-alt-def*:

$\langle \text{vars-llist } xs \subseteq \mathcal{V} \longleftrightarrow \text{vars-of-poly-in } xs \mathcal{V} \rangle$

**by** (*induction xs*)

(*auto simp: vars-llist-def vars-of-monom-in-alt-def*)

**lemma** *vars-of-monom-in-alt-def2*:

$\langle \text{vars-of-monom-in } xs \mathcal{V} \longleftrightarrow \text{fold } (\lambda x b. b \wedge x \in \mathcal{V}) \ xs \ \text{True} \rangle$

**apply** (*subst foldr-fold[symmetric]*)

**subgoal by** *auto*

**subgoal by** (*induction xs*) *auto*

**done**

**sempref-definition** *vars-of-monom-in-impl*

**is**  $\langle \text{uncurry } (\text{RETURN } \text{oo } \text{vars-of-monom-in}) \rangle$

$:: \langle (\text{list-assn string-assn})^k *_a \text{vars-assn}^k \rightarrow_a \text{bool-assn} \rangle$

**unfolding** *vars-of-monom-in-alt-def2*

**by** *sempref*

**declare** *vars-of-monom-in-impl.refine[sempref-fr-rules]*

**lemma** *vars-of-poly-in-alt-def2*:

$\langle \text{vars-of-poly-in } xs \mathcal{V} \longleftrightarrow \text{fold } (\lambda(x, -) b. b \wedge \text{vars-of-monom-in } x \mathcal{V}) \ xs \ \text{True} \rangle$

**apply** (*subst foldr-fold[symmetric]*)

**subgoal by** *auto*

**subgoal by** (*induction xs*) *auto*

**done**

**sempref-definition** *vars-of-poly-in-impl*

**is**  $\langle \text{uncurry } (\text{RETURN } \text{oo } \text{vars-of-poly-in}) \rangle$

$:: \langle (\text{poly-assn})^k *_a \text{vars-assn}^k \rightarrow_a \text{bool-assn} \rangle$

**unfolding** *vars-of-poly-in-alt-def2*

**by** *sempref*

**declare** *vars-of-poly-in-impl.refine[sempref-fr-rules]*

**definition** *union-vars-monom*  $:: \langle \text{string list} \Rightarrow \text{string set} \Rightarrow \text{string set} \rangle$  **where**

$\langle \text{union-vars-monom } xs \mathcal{V} = \text{fold insert } xs \mathcal{V} \rangle$

**definition** *union-vars-poly* ::  $\langle \text{llist-polynom} \Rightarrow \text{string set} \Rightarrow \text{string set} \rangle$  **where**  
 $\langle \text{union-vars-poly } xs \ \mathcal{V} = \text{fold } (\lambda(xs, -) \ \mathcal{V}. \text{union-vars-monom } xs \ \mathcal{V}) \ xs \ \mathcal{V} \rangle$

**lemma** *union-vars-monom-alt-def*:  
 $\langle \text{union-vars-monom } xs \ \mathcal{V} = \mathcal{V} \cup \text{set } xs \rangle$   
**unfolding** *union-vars-monom-def*  
**apply** (*subst foldr-fold[symmetric]*)  
**subgoal for**  $x \ y$   
  **by** (*cases x; cases y*) *auto*  
**subgoal**  
  **by** (*induction xs*) *auto*  
**done**

**lemma** *union-vars-poly-alt-def*:  
 $\langle \text{union-vars-poly } xs \ \mathcal{V} = \mathcal{V} \cup \text{vars-llist } xs \rangle$   
**unfolding** *union-vars-poly-def*  
**apply** (*subst foldr-fold[symmetric]*)  
**subgoal for**  $x \ y$   
  **by** (*cases x; cases y*)  
  (*auto simp: union-vars-monom-alt-def*)  
**subgoal**  
  **by** (*induction xs*)  
  (*auto simp: vars-llist-def union-vars-monom-alt-def*)  
**done**

**sempref-definition** *union-vars-monom-impl*  
**is**  $\langle \text{uncurry } (\text{RETURN } oo \text{ union-vars-monom}) \rangle$   
 $:: \langle \text{monom-assn}^k *_a \text{vars-assn}^d \rightarrow_a \text{vars-assn} \rangle$   
**unfolding** *union-vars-monom-def*  
**by** *sempref*

**declare** *union-vars-monom-impl.refine[sempref-fr-rules]*

**sempref-definition** *union-vars-poly-impl*  
**is**  $\langle \text{uncurry } (\text{RETURN } oo \text{ union-vars-poly}) \rangle$   
 $:: \langle \text{poly-assn}^k *_a \text{vars-assn}^d \rightarrow_a \text{vars-assn} \rangle$   
**unfolding** *union-vars-poly-def*  
**by** *sempref*

**declare** *union-vars-poly-impl.refine[sempref-fr-rules]*

**hide-const** (**open**) *Autoref-Fix-Rel.CONSTRAINT*

**fun** *status-assn* **where**  
 $\langle \text{status-assn } - \text{ CSUCCESS } \text{CSUCCESS} = \text{emp} \rangle \mid$   
 $\langle \text{status-assn } - \text{ CFOUND } \text{CFOUND} = \text{emp} \rangle \mid$   
 $\langle \text{status-assn } R \ (\text{CFAILED } a) \ (\text{CFAILED } b) = R \ a \ b \rangle \mid$   
 $\langle \text{status-assn } - \text{ - } = \text{false} \rangle$

**lemma** *SUCCESS-hnr[sempref-fr-rules]*:  
 $\langle (\text{uncurry0 } (\text{return } \text{CSUCCESS}), \text{uncurry0 } (\text{RETURN } \text{CSUCCESS})) \in \text{unit-assn}^k \rightarrow_a \text{status-assn } R \rangle$   
**by** (*sempref-to-hoare*)  
  *sep-auto*

**lemma** *FOUND-hnr*[*sepref-fr-rules*]:  
 $\langle (\text{uncurry0 } (\text{return } \text{CFOUND}), \text{uncurry0 } (\text{RETURN } \text{CFOUND})) \in \text{unit-assn}^k \rightarrow_a \text{status-assn } R \rangle$   
**by** (*sepref-to-hoare*)  
*sep-auto*

**lemma** *is-success-hnr*[*sepref-fr-rules*]:  
 $\langle \text{CONSTRAINT is-pure } R \Rightarrow$   
 $((\text{return } o \text{ is-cfound}), (\text{RETURN } o \text{ is-cfound})) \in (\text{status-assn } R)^k \rightarrow_a \text{bool-assn} \rangle$   
**apply** (*sepref-to-hoare*)  
**apply** (*rename-tac xi x; case-tac xi; case-tac x*)  
**apply** *sep-auto+*  
**done**

**lemma** *is-cfailed-hnr*[*sepref-fr-rules*]:  
 $\langle \text{CONSTRAINT is-pure } R \Rightarrow$   
 $((\text{return } o \text{ is-cfailed}), (\text{RETURN } o \text{ is-cfailed})) \in (\text{status-assn } R)^k \rightarrow_a \text{bool-assn} \rangle$   
**apply** (*sepref-to-hoare*)  
**apply** (*rename-tac xi x; case-tac xi; case-tac x*)  
**apply** *sep-auto+*  
**done**

**lemma** *merge-cstatus-hnr*[*sepref-fr-rules*]:  
 $\langle \text{CONSTRAINT is-pure } R \Rightarrow$   
 $(\text{uncurry } (\text{return } oo \text{ merge-cstatus}), \text{uncurry } (\text{RETURN } oo \text{ merge-cstatus})) \in$   
 $(\text{status-assn } R)^k *_a (\text{status-assn } R)^k \rightarrow_a \text{status-assn } R \rangle$   
**apply** (*sepref-to-hoare*)  
**by** (*case-tac b; case-tac bi; case-tac a; case-tac ai; sep-auto simp: is-pure-conv pure-app-eq*)

**sepref-definition** *add-poly-impl*  
**is**  $\langle \text{add-poly-l} \rangle$   
 $:: \langle (\text{poly-assn} \times_a \text{poly-assn})^k \rightarrow_a \text{poly-assn} \rangle$   
**supply**  $[[\text{goals-limit}=1]]$   
**unfolding** *add-poly-l-def*  
*HOL-list.fold-custom-empty*  
*term-order-rel'-def[symmetric]*  
*term-order-rel'-alt-def*  
**by** *sepref*

**declare** *add-poly-impl.refine*[*sepref-fr-rules*]

**sepref-register** *mult-monomials*

**lemma** *mult-monoms-alt-def*:  
 $\langle (\text{RETURN } oo \text{ mult-monoms}) x y = \text{REC}_T$   
 $(\lambda f (p, q).$   
*case*  $(p, q)$  *of*  
 $([], -) \Rightarrow \text{RETURN } q$   
 $| (-, []) \Rightarrow \text{RETURN } p$   
 $| (x \# p, y \# q) \Rightarrow$   
 $(\text{if } x = y \text{ then do } \{$   
 $\quad pq \leftarrow f (p, q);$   
 $\quad \text{RETURN } (x \# pq)\}$   
 $\text{else if } (x, y) \in \text{var-order-rel}$

```

    then do {
      pq ← f (p, y # q);
      RETURN (x # pq)}
    else do {
      pq ← f (x # p, q);
      RETURN (y # pq)}}))
  (x, y)
apply (subst eq-commute)
apply (induction x y rule: mult-monomys.induct)
subgoal for p
  by (subst RECT-unfold, refine-mono) (auto split: list.splits)
subgoal for p
  by (subst RECT-unfold, refine-mono) (auto split: list.splits)
subgoal for x p y q
  by (subst RECT-unfold, refine-mono) (auto split: list.splits simp: let-to-bind-conv)
done

```

**sempref-definition** *mult-monomys-impl*

```

is ⟨uncurry (RETURN oo mult-monomys)⟩
:: ⟨(monom-assn)k *a (monom-assn)k →a (monom-assn)⟩
supply [[goals-limit=1]]
unfolding mult-poly-raw-def
  HOL-list.fold-custom-empty
  var-order'-def[symmetric]
  term-order-rel'-alt-def
  mult-monomys-alt-def
  var-order-rel-var-order
by sempref

```

**declare** *mult-monomys-impl.refine*[sempref-fr-rules]

**sempref-definition** *mult-monomials-impl*

```

is ⟨uncurry (RETURN oo mult-monomials)⟩
:: ⟨(monomial-assn)k *a (monomial-assn)k →a (monomial-assn)⟩
supply [[goals-limit=1]]
unfolding mult-monomials-def
  HOL-list.fold-custom-empty
  term-order-rel'-def[symmetric]
  term-order-rel'-alt-def
by sempref

```

**lemma** *map-append-alt-def2*:

```

⟨(RETURN o (map-append f b)) xs = RECT
  (λg xs. case xs of [] ⇒ RETURN b
    | x # xs ⇒ do {
      y ← g xs;
      RETURN (f x # y)
    }) xs⟩
apply (subst eq-commute)
apply (induction f b xs rule: map-append.induct)
subgoal by (subst RECT-unfold, refine-mono) auto
subgoal by (subst RECT-unfold, refine-mono) auto
done

```

**definition** *map-append-poly-mult* **where**  
 $\langle \text{map-append-poly-mult } x = \text{map-append } (\text{mult-monomials } x) \rangle$

**declare** *mult-monomials-impl.refine*[*sepref-fr-rules*]

**sepref-definition** *map-append-poly-mult-impl*  
**is**  $\langle \text{uncurry2 } (\text{RETURN } \text{ooo } \text{map-append-poly-mult}) \rangle$   
 $:: \langle \text{monomial-assn}^k *_a \text{poly-assn}^k *_a \text{poly-assn}^k \rightarrow_a \text{poly-assn} \rangle$   
**unfolding** *map-append-poly-mult-def*  
*map-append-alt-def2*  
**by** *sepref*

**declare** *map-append-poly-mult-impl.refine*[*sepref-fr-rules*]

TODO *foldl*  $(\lambda l \ x. \ l \ @ \ [?f \ x]) \ [] \ ?l = \text{map } ?f \ ?l$  is the worst possible implementation of *map*!

**sepref-definition** *mult-poly-raw-impl*  
**is**  $\langle \text{uncurry } (\text{RETURN } \text{oo } \text{mult-poly-raw}) \rangle$   
 $:: \langle \text{poly-assn}^k *_a \text{poly-assn}^k \rightarrow_a \text{poly-assn} \rangle$   
**supply**  $[[\text{goals-limit}=1]]$   
**supply**  $[[\text{eta-contract} = \text{false}, \text{show-abbrevs}=\text{false}]]$   
**unfolding** *mult-poly-raw-def*  
*HOL-list.fold-custom-empty*  
*term-order-rel'-def*[*symmetric*]  
*term-order-rel'-alt-def*  
*foldl-conv-fold*  
*fold-eq-nfoldli*  
*map-append-poly-mult-def*[*symmetric*]  
*map-append-alt-def*[*symmetric*]  
**by** *sepref*

**declare** *mult-poly-raw-impl.refine*[*sepref-fr-rules*]

**sepref-definition** *mult-poly-impl*  
**is**  $\langle \text{uncurry } \text{mult-poly-full} \rangle$   
 $:: \langle \text{poly-assn}^k *_a \text{poly-assn}^k \rightarrow_a \text{poly-assn} \rangle$   
**supply**  $[[\text{goals-limit}=1]]$   
**unfolding** *mult-poly-full-def*  
*HOL-list.fold-custom-empty*  
*term-order-rel'-def*[*symmetric*]  
*term-order-rel'-alt-def*  
**by** *sepref*

**declare** *mult-poly-impl.refine*[*sepref-fr-rules*]

**lemma** *inverse-monomial*:  
 $\langle \text{monom-rel}^{-1} \times_r \text{int-rel} = (\text{monom-rel} \times_r \text{int-rel})^{-1} \rangle$   
**by** (*auto*)

**lemma** *eq-poly-rel-eq*[*sepref-import-param*]:  
 $\langle ((=), (=)) \in \text{poly-rel} \rightarrow \text{poly-rel} \rightarrow \text{bool-rel} \rangle$   
**using** *list-rel-sv*[*of*  $\langle \text{monomial-rel} \rangle$ , *OF* *single-valued-monomial-rel*]  
**using** *list-rel-sv*[*OF* *single-valued-monomial-rel'*[*unfolded IS-LEFT-UNIQUE-def inv-list-rel-eq*]]

```

unfolding inv-list-rel-eq[symmetric]
by (auto intro!: frefI simp:
      rel2p-def single-valued-def p2rel-def
      simp del: inv-list-rel-eq)

sempref-definition weak-equality-l-impl
is ⟨uncurry weak-equality-l⟩
:: ⟨poly-assnk *a poly-assnk →a bool-assn⟩
supply [[goals-limit=1]]
unfolding weak-equality-l-def
by sempref

declare weak-equality-l-impl.refine[sempref-fr-rules]
sempref-register add-poly-l mult-poly-full

abbreviation raw-string-assn :: ⟨string ⇒ string ⇒ assn⟩ where
  ⟨raw-string-assn ≡ list-assn id-assn⟩

definition show-nat :: ⟨nat ⇒ string⟩ where
  ⟨show-nat i = show i⟩

lemma [sempref-import-param]:
  ⟨(show-nat, show-nat) ∈ nat-rel → ⟨Id⟩list-rel⟩
by (auto intro: fun-relI)

lemma status-assn-pure-conv:
  ⟨status-assn (id-assn) a b = id-assn a b⟩
by (cases a; cases b)
  (auto simp: pure-def)

lemma [sempref-fr-rules]:
  ⟨(uncurry3 (λx y. return oo (error-msg-not-equal-dom x y)), uncurry3 check-not-equal-dom-err) ∈
    poly-assnk *a poly-assnk *a poly-assnk *a poly-assnk →a raw-string-assn⟩
unfolding show-nat-def[symmetric] list-assn-pure-conv
  prod-assn-pure-conv check-not-equal-dom-err-def
by (sempref-to-hoare; sep-auto simp: error-msg-not-equal-dom-def)

lemma [sempref-fr-rules]:
  ⟨(return o (error-msg-notin-dom o nat-of-uint64), RETURN o error-msg-notin-dom)
    ∈ uint64-nat-assnk →a raw-string-assn⟩
  ⟨(return o (error-msg-reused-dom o nat-of-uint64), RETURN o error-msg-reused-dom)
    ∈ uint64-nat-assnk →a raw-string-assn⟩
  ⟨(uncurry (return oo (λi. error-msg (nat-of-uint64 i))), uncurry (RETURN oo error-msg))
    ∈ uint64-nat-assnk *a raw-string-assnk →a status-assn raw-string-assn⟩
  ⟨(uncurry (return oo error-msg), uncurry (RETURN oo error-msg))
    ∈ nat-assnk *a raw-string-assnk →a status-assn raw-string-assn⟩
unfolding error-msg-notin-dom-def list-assn-pure-conv list-rel-id-simp
unfolding status-assn-pure-conv
unfolding show-nat-def[symmetric]
by (sempref-to-hoare; sep-auto simp: uint64-nat-rel-def br-def; fail)+

sempref-definition check-addition-l-impl

```



```

is ⟨uncurry6 check-addition-l⟩
:: ⟨poly-assnk *a polys-assnk *a vars-assnk *a uint64-nat-assnk *a uint64-nat-assnk *a
    uint64-nat-assnk *a poly-assnk →a status-assn raw-string-assn⟩
supply [[goals-limit=1]]
unfolding mult-poly-full-def
  HOL-list.fold-custom-empty
  term-order-rel'-def[symmetric]
  term-order-rel'-alt-def
  check-addition-l-def
  in-dom-m-lookup-iff
  fmlookup'-def[symmetric]
  vars-llist-alt-def
by sepref

declare check-addition-l-impl.refine[sepref-fr-rules]

sepref-register check-mult-l-dom-err

definition check-mult-l-dom-err-impl where
  ⟨check-mult-l-dom-err-impl pd p ia i =
    (if pd then "The polynom with id " @ show (nat-of-uint64 p) @ " was not found" else "'') @
    (if ia then "The id of the resulting id " @ show (nat-of-uint64 i) @ " was already given" else "'')⟩

definition check-mult-l-mult-err-impl where
  ⟨check-mult-l-mult-err-impl p q pq r =
    "Multiplying " @ show p @ " by " @ show q @ " gives " @ show pq @ " and not " @ show r⟩

lemma [sepref-fr-rules]:
  ⟨(uncurry3 ((λx y. return oo (check-mult-l-dom-err-impl x y))),
    uncurry3 (check-mult-l-dom-err)) ∈ bool-assnk *a uint64-nat-assnk *a bool-assnk *a uint64-nat-assnk
  →a raw-string-assn⟩
  unfolding check-mult-l-dom-err-def check-mult-l-dom-err-impl-def list-assn-pure-conv
  apply sepref-to-hoare
  apply sep-auto
  done

lemma [sepref-fr-rules]:
  ⟨(uncurry3 ((λx y. return oo (check-mult-l-mult-err-impl x y))),
    uncurry3 (check-mult-l-mult-err)) ∈ poly-assnk *a poly-assnk *a poly-assnk *a poly-assnk →a raw-string-assn⟩
  unfolding check-mult-l-mult-err-def check-mult-l-mult-err-impl-def list-assn-pure-conv
  apply sepref-to-hoare
  apply sep-auto
  done

sepref-definition check-mult-l-impl
  is ⟨uncurry6 check-mult-l⟩
  :: ⟨poly-assnk *a polys-assnk *a vars-assnk *a uint64-nat-assnk *a poly-assnk *a uint64-nat-assnk *a
    poly-assnk →a status-assn raw-string-assn⟩
  supply [[goals-limit=1]]
  unfolding check-mult-l-def
    HOL-list.fold-custom-empty
    term-order-rel'-def[symmetric]
    term-order-rel'-alt-def
    in-dom-m-lookup-iff
    fmlookup'-def[symmetric]

```

*vars-llist-alt-def*  
by *sepref*

**declare** *check-mult-l-impl.refine[sepref-fr-rules]*

**definition** *check-ext-l-dom-err-impl* ::  $\langle \text{uint64} \Rightarrow \cdot \rangle$  **where**  
 $\langle \text{check-ext-l-dom-err-impl } p =$   
*"There is already a polynom with index "* @ *show (nat-of-uint64 p)* $\rangle$

**lemma** [*sepref-fr-rules*]:  
 $\langle (((\text{return } o \text{ (check-ext-l-dom-err-impl)})),$   
 $(\text{check-extension-l-dom-err})) \in \text{uint64-nat-assn}^k \rightarrow_a \text{raw-string-assn} \rangle$   
**unfolding** *check-extension-l-dom-err-def check-ext-l-dom-err-impl-def list-assn-pure-conv*  
**apply** *sepref-to-hoare*  
**apply** *sep-auto*  
**done**

**definition** *check-extension-l-no-new-var-err-impl* ::  $\langle \cdot \Rightarrow \cdot \rangle$  **where**  
 $\langle \text{check-extension-l-no-new-var-err-impl } p =$   
*"No new variable could be found in polynom "* @ *show p* $\rangle$

**lemma** [*sepref-fr-rules*]:  
 $\langle (((\text{return } o \text{ (check-extension-l-no-new-var-err-impl)})),$   
 $(\text{check-extension-l-no-new-var-err})) \in \text{poly-assn}^k \rightarrow_a \text{raw-string-assn} \rangle$   
**unfolding** *check-extension-l-no-new-var-err-impl-def check-extension-l-no-new-var-err-def*  
*list-assn-pure-conv*  
**apply** *sepref-to-hoare*  
**apply** *sep-auto*  
**done**

**definition** *check-extension-l-side-cond-err-impl* ::  $\langle \cdot \Rightarrow \cdot \rangle$  **where**  
 $\langle \text{check-extension-l-side-cond-err-impl } v \text{ } p \text{ } r \text{ } s =$   
*"Error while checking side conditions of extensions polynow, var is "* @ *show v* @  
*" polynom is "* @ *show p* @ *"side condition p\*p - p = "* @ *show s* @ *" and should be 0"* $\rangle$

**lemma** [*sepref-fr-rules*]:  
 $\langle ((\text{uncurry3 } (\lambda x \ y. \text{return } oo \text{ (check-extension-l-side-cond-err-impl } x \ y))),$   
 $\text{uncurry3 } (\text{check-extension-l-side-cond-err})) \in \text{string-assn}^k *_a \text{poly-assn}^k *_a \text{poly-assn}^k *_a \text{poly-assn}^k$   
 $\rightarrow_a \text{raw-string-assn} \rangle$   
**unfolding** *check-extension-l-side-cond-err-impl-def check-extension-l-side-cond-err-def*  
*list-assn-pure-conv*  
**apply** *sepref-to-hoare*  
**apply** *sep-auto*  
**done**

**definition** *check-extension-l-new-var-multiple-err-impl* ::  $\langle \cdot \Rightarrow \cdot \rangle$  **where**  
 $\langle \text{check-extension-l-new-var-multiple-err-impl } v \text{ } p =$   
*"Error while checking side conditions of extensions polynow, var is "* @ *show v* @  
*" but it either appears at least once in the polynom or another new variable is created "* @  
*show p* @ *" but should not."* $\rangle$

**lemma** [*sepref-fr-rules*]:  
 $\langle ((\text{uncurry } (\text{return } oo \text{ (check-extension-l-new-var-multiple-err-impl)})),$   
 $\text{uncurry } (\text{check-extension-l-new-var-multiple-err})) \in \text{string-assn}^k *_a \text{poly-assn}^k \rightarrow_a \text{raw-string-assn} \rangle$

**unfolding** *check-extension-l-new-var-multiple-err-impl-def*  
*check-extension-l-new-var-multiple-err-def*  
*list-assn-pure-conv*  
**apply** *sepref-to-hoare*  
**apply** *sep-auto*  
**done**

**sepref-register** *check-extension-l-dom-err fmlookup'*  
*check-extension-l-side-cond-err check-extension-l-no-new-var-err*  
*check-extension-l-new-var-multiple-err*

**definition** *uminus-poly* ::  $\langle \text{llist-polynom} \Rightarrow \text{llist-polynom} \rangle$  **where**  
 $\langle \text{uminus-poly } p' = \text{map } (\lambda(a, b). (a, - b)) p' \rangle$

**sepref-register** *uminus-poly*  
**lemma** [*sepref-import-param*]:  
 $\langle (\text{map } (\lambda(a, b). (a, - b)), \text{uminus-poly}) \in \text{poly-rel} \rightarrow \text{poly-rel} \rangle$   
**unfolding** *uminus-poly-def*  
**apply** (*intro fun-relI*)  
**subgoal for** *p p'*  
**by** (*induction p p' rule: list-rel-induct*)  
*auto*  
**done**

**sepref-register** *vars-of-poly-in*  
*weak-equality-l*

**lemma** [*safe-constraint-rules*]:  
 $\langle \text{Sepref-Constraints.CONSTRAINT single-valued (the-pure monomial-assn)} \rangle$  **and**  
*single-valued-the-monomial-assn*:  
 $\langle \text{single-valued (the-pure monomial-assn)} \rangle$   
 $\langle \text{single-valued } ((\text{the-pure monomial-assn})^{-1}) \rangle$   
**unfolding** *IS-LEFT-UNIQUE-def[symmetric]*  
**by** (*auto simp: step-rewrite-pure single-valued-monomial-rel single-valued-monomial-rel' Sepref-Constraints.CONSTRAINT*)

**sepref-definition** *check-extension-l-impl*  
**is**  $\langle \text{uncurry5 check-extension-l} \rangle$   
 $:: \langle \text{poly-assn}^k *_a \text{polys-assn}^k *_a \text{vars-assn}^k *_a \text{uint64-nat-assn}^k *_a \text{string-assn}^k *_a \text{poly-assn}^k \rightarrow_a \text{status-assn raw-string-assn} \rangle$   
**supply** *option.splits[split] single-valued-the-monomial-assn[simp]*  
**supply** *[[goals-limit=1]]*  
**unfolding**  
*HOL-list.fold-custom-empty*  
*term-order-rel'-def[symmetric]*  
*term-order-rel'-alt-def*  
*in-dom-m-lookup-iff*  
*fmlookup'-def[symmetric]*  
*vars-llist-alt-def*  
*check-extension-l-def*  
*not-not*  
*option.case-eq-if*  
*uminus-poly-def[symmetric]*  
*HOL-list.fold-custom-empty*  
**by** *sepref*

**declare** *check-extension-l-impl.refine*[sepref-fr-rules]

**sepref-definition** *check-del-l-impl*

**is**  $\langle \text{uncurry2 } \text{check-del-l} \rangle$   
 $:: \langle \text{poly-assn}^k *_{\text{a}} \text{polys-assn}^k *_{\text{a}} \text{uint64-nat-assn}^k \rightarrow_{\text{a}} \text{status-assn raw-string-assn} \rangle$   
**supply** [[goals-limit=1]]  
**unfolding** *check-del-l-def*  
*in-dom-m-lookup-iff*  
*fmlookup'-def*[symmetric]  
**by** *sepref*

**lemmas** [sepref-fr-rules] = *check-del-l-impl.refine*

**abbreviation** *pac-step-rel* **where**

$\langle \text{pac-step-rel} \equiv \text{p2rel } (\langle \text{Id}, \langle \text{monomial-rel} \rangle \text{list-rel}, \text{Id} \rangle \text{ pac-step-rel-raw}) \rangle$

**sepref-register** *PAC-Polynoms-Operations.normalize-poly*

*pac-src1 pac-src2 new-id pac-mult case-pac-step check-mult-l*  
*check-addition-l check-del-l check-extension-l*

**lemma** *pac-step-rel-assn-alt-def2*:

$\langle \text{hn-ctxt } (\text{pac-step-rel-assn nat-assn poly-assn id-assn}) \text{ b bi} =$   
 $\text{hn-val}$   
 $(\text{p2rel}$   
 $((\text{nat-rel}, \text{poly-rel}, \text{Id} :: (\text{string} \times -) \text{ set}) \text{ pac-step-rel-raw})) \text{ b bi} \rangle$   
**unfolding** *poly-assn-list hn-ctxt-def*  
**by** (*induction nat-assn poly-assn <id-assn :: string  $\Rightarrow$  -> b bi rule: pac-step-rel-assn.induct*)  
*(auto simp: p2rel-def hn-val-unfold pac-step-rel-raw.simps relAPP-def*  
*pure-app-eq)*

**lemma** *is-AddD-import*[sepref-fr-rules]:

**assumes**  $\langle \text{CONSTRAINT is-pure } K \rangle \langle \text{CONSTRAINT is-pure } V \rangle$

**shows**

$\langle (\text{return o pac-res}, \text{RETURN o pac-res}) \in [\lambda x. \text{is-Add } x \vee \text{is-Mult } x \vee \text{is-Extension } x]_{\text{a}}$   
 $(\text{pac-step-rel-assn } K \text{ V R})^k \rightarrow V \rangle$   
 $\langle (\text{return o pac-src1}, \text{RETURN o pac-src1}) \in [\lambda x. \text{is-Add } x \vee \text{is-Mult } x \vee \text{is-Del } x]_{\text{a}} (\text{pac-step-rel-assn}$   
 $K \text{ V R})^k \rightarrow K \rangle$   
 $\langle (\text{return o new-id}, \text{RETURN o new-id}) \in [\lambda x. \text{is-Add } x \vee \text{is-Mult } x \vee \text{is-Extension } x]_{\text{a}} (\text{pac-step-rel-assn}$   
 $K \text{ V R})^k \rightarrow K \rangle$   
 $\langle (\text{return o is-Add}, \text{RETURN o is-Add}) \in (\text{pac-step-rel-assn } K \text{ V R})^k \rightarrow_{\text{a}} \text{bool-assn} \rangle$   
 $\langle (\text{return o is-Mult}, \text{RETURN o is-Mult}) \in (\text{pac-step-rel-assn } K \text{ V R})^k \rightarrow_{\text{a}} \text{bool-assn} \rangle$   
 $\langle (\text{return o is-Del}, \text{RETURN o is-Del}) \in (\text{pac-step-rel-assn } K \text{ V R})^k \rightarrow_{\text{a}} \text{bool-assn} \rangle$   
 $\langle (\text{return o is-Extension}, \text{RETURN o is-Extension}) \in (\text{pac-step-rel-assn } K \text{ V R})^k \rightarrow_{\text{a}} \text{bool-assn} \rangle$   
**using** *assms*  
**by** (*sepref-to-hoare; sep-auto simp: pac-step-rel-assn-alt-def is-pure-conv ent-true-drop pure-app-eq*  
*split: pac-step.splits; fail*) $+$

**lemma** [sepref-fr-rules]:

$\langle \text{CONSTRAINT is-pure } K \implies$   
 $(\text{return o pac-src2}, \text{RETURN o pac-src2}) \in [\lambda x. \text{is-Add } x]_{\text{a}} (\text{pac-step-rel-assn } K \text{ V R})^k \rightarrow K \rangle$   
 $\langle \text{CONSTRAINT is-pure } V \implies$   
 $(\text{return o pac-mult}, \text{RETURN o pac-mult}) \in [\lambda x. \text{is-Mult } x]_{\text{a}} (\text{pac-step-rel-assn } K \text{ V R})^k \rightarrow V \rangle$

$\langle \text{CONSTRAINT } \text{is-pure } R \implies$   
 $(\text{return } o \text{ new-var}, \text{RETURN } o \text{ new-var}) \in [\lambda x. \text{is-Extension } x]_a (\text{pac-step-rel-assn } K \ V \ R)^k \rightarrow R \rangle$   
**by**  $(\text{sepref-to-hoare}; \text{sep-auto simp}; \text{pac-step-rel-assn-alt-def is-pure-conv ent-true-drop pure-app-eq}$   
 $\text{split}; \text{pac-step.splits}; \text{fail})+$

**lemma** *is-Mult-lastI*:

$\langle \neg \text{is-Add } b \implies \neg \text{is-Mult } b \implies \neg \text{is-Extension } b \implies \text{is-Del } b \rangle$   
**by**  $(\text{cases } b) \text{ auto}$

**sepref-register** *is-cfailed is-Del*

**definition** *PAC-checker-l-step'* :: - **where**

$\langle \text{PAC-checker-l-step}' a \ b \ c \ d = \text{PAC-checker-l-step } a \ (b, c, d) \rangle$

**lemma** *PAC-checker-l-step-alt-def*:

$\langle \text{PAC-checker-l-step } a \ bcd \ e = (\text{let } (b, c, d) = bcd \text{ in } \text{PAC-checker-l-step}' a \ b \ c \ d \ e) \rangle$   
**unfolding** *PAC-checker-l-step'-def* **by** *auto*

**sepref-decl-intf**  $(\text{'k}) \ \text{acode-status is } (\text{'k}) \ \text{code-status}$

**sepref-decl-intf**  $(\text{'k}, \text{'b}, \text{'lbl}) \ \text{apac-step is } (\text{'k}, \text{'b}, \text{'lbl}) \ \text{pac-step}$

**sepref-register** *merge-cstatus full-normalize-poly new-var is-Add*

**lemma** *poly-rel-the-pure*:

$\langle \text{poly-rel} = \text{the-pure poly-assn} \rangle$  **and**  
 $\text{nat-rel-the-pure}$ :  
 $\langle \text{nat-rel} = \text{the-pure nat-assn} \rangle$  **and**  
*WTF-RF*:  $\langle \text{pure } (\text{the-pure nat-assn}) = \text{nat-assn} \rangle$   
**unfolding** *poly-assn-list*  
**by** *auto*

**lemma** *[safe-constraint-rules]*:

$\langle \text{CONSTRAINT IS-LEFT-UNIQUE uint64-nat-rel} \rangle$  **and**  
 $\text{single-valued-uint64-nat-rel}[\text{safe-constraint-rules}]$ :  
 $\langle \text{CONSTRAINT single-valued uint64-nat-rel} \rangle$   
**by**  $(\text{auto simp}; \text{IS-LEFT-UNIQUE-def single-valued-def uint64-nat-rel-def br-def})$

**sepref-definition** *check-step-impl*

**is**  $\langle \text{uncurry4 PAC-checker-l-step}' \rangle$   
 $:: \langle \text{poly-assn}^k *_a (\text{status-assn raw-string-assn})^d *_a \text{vars-assn}^d *_a \text{polys-assn}^d *_a (\text{pac-step-rel-assn}$   
 $(\text{uint64-nat-assn}) \text{poly-assn } (\text{string-assn} :: \text{string} \Rightarrow -))^d \rightarrow_a$   
 $\text{status-assn raw-string-assn} \times_a \text{vars-assn} \times_a \text{polys-assn} \rangle$   
**supply**  $[[\text{goals-limit}=1]] \ \text{is-Mult-lastI}[\text{intro}] \ \text{single-valued-uint64-nat-rel}[\text{simp}]$   
**unfolding** *PAC-checker-l-step-def PAC-checker-l-step'-def*  
 $\text{pac-step.case-eq-if Let-def}$   
 $\text{is-success-alt-def}[\text{symmetric}]$   
 $\text{uminus-poly-def}[\text{symmetric}]$   
 $\text{HOL-list.fold-custom-empty}$   
**by** *sepref*

**declare** *check-step-impl.refine[sepref-fr-rules]*

**sepref-register** *PAC-checker-l-step PAC-checker-l-step' fully-normalize-poly-impl*

**definition** *PAC-checker-l'* **where**

$\langle \text{PAC-checker-l}' \ p \ \mathcal{V} \ A \ \text{status steps} = \text{PAC-checker-l} \ p \ (\mathcal{V}, A) \ \text{status steps} \rangle$

**lemma** *PAC-checker-l-alt-def*:

$\langle \text{PAC-checker-l} \ p \ \mathcal{V} \ A \ \text{status steps} =$   
 $(\text{let } (\mathcal{V}, A) = \mathcal{V} \ A \ \text{in } \text{PAC-checker-l}' \ p \ \mathcal{V} \ A \ \text{status steps}) \rangle$

**unfolding** *PAC-checker-l'-def* **by** *auto*

**sempref-definition** *PAC-checker-l-impl*

**is**  $\langle \text{uncurry4} \ \text{PAC-checker-l}' \rangle$

$:: \langle \text{poly-assn}^k *_a \text{vars-assn}^d *_a \text{polys-assn}^d *_a (\text{status-assn} \ \text{raw-string-assn})^d *_a$   
 $(\text{list-assn} \ (\text{pac-step-rel-assn} \ (\text{uint64-nat-assn}) \ \text{poly-assn} \ \text{string-assn}))^k \rightarrow_a$   
 $\text{status-assn} \ \text{raw-string-assn} \times_a \text{vars-assn} \times_a \text{polys-assn} \rangle$

**supply**  $[[\text{goals-limit}=1]]$  *is-Mult-lastI*[*intro*]

**unfolding** *PAC-checker-l-def* *is-success-alt-def*[*symmetric*] *PAC-checker-l-step-alt-def*  
*nres-bind-let-law*[*symmetric*] *PAC-checker-l'-def*

**apply** (*subst nres-bind-let-law*)

**by** *sempref*

**declare** *PAC-checker-l-impl.refine*[*sempref-fr-rules*]

**abbreviation** *polys-assn-input* **where**

$\langle \text{polys-assn-input} \equiv \text{iam-fmap-assn} \ \text{nat-assn} \ \text{poly-assn} \rangle$

**definition** *remap-polys-l-dom-err-impl*  $:: (-) \rightarrow$  **where**

$\langle \text{remap-polys-l-dom-err-impl} =$

*"Error during initialisation. Too many polynoms where provided. If this happens," @*  
*"please report the example to the authors, because something went wrong during " @*  
*"code generation (code generation to arrays is likely to be broken)."*  $\rangle$

**lemma** [*sempref-fr-rules*]:

$\langle ((\text{uncurry0} \ (\text{return} \ (\text{remap-polys-l-dom-err-impl}))),$   
 $\text{uncurry0} \ (\text{remap-polys-l-dom-err})) \in \text{unit-assn}^k \rightarrow_a \text{raw-string-assn} \rangle$

**unfolding** *remap-polys-l-dom-err-def*

*remap-polys-l-dom-err-def*

*list-assn-pure-conv*

**by** *sempref-to-hoare sep-auto*

MLton is not able to optimise the calls to *pow*.

**lemma** *pow-2-64*:  $\langle (2::\text{nat}) \wedge 64 = 18446744073709551616 \rangle$

**by** *auto*

**sempref-register** *upper-bound-on-dom op-fmap-empty*

**sempref-definition** *remap-polys-l-impl*

**is**  $\langle \text{uncurry2} \ \text{remap-polys-l2} \rangle$

$:: \langle \text{poly-assn}^k *_a \text{vars-assn}^d *_a \text{polys-assn-input}^d \rightarrow_a$   
 $\text{status-assn} \ \text{raw-string-assn} \times_a \text{vars-assn} \times_a \text{polys-assn} \rangle$

**supply**  $[[\text{goals-limit}=1]]$  *is-Mult-lastI*[*intro*] *indom-mI*[*dest*]

**unfolding** *remap-polys-l2-def* *op-fmap-empty-def*[*symmetric*] *while-eq-nfoldli*[*symmetric*]

*while-upt-while-direct* *pow-2-64*

*in-dom-m-lookup-iff*

*fmlookup'-def*[*symmetric*]

*union-vars-poly-alt-def*[*symmetric*]

**apply** (*rewrite at*  $\langle \text{fmupd} \sqsupset \text{uint64-of-nat-conv-def}[\text{symmetric}] \rangle$ )

```

apply (subst while-upt-while-direct)
apply simp
apply (rewrite at ⟨op-fmap-empty⟩ annotate-assn[where A=⟨polys-assn⟩])
by sepref

```

```

lemma remap-polys-l2-remap-polys-l:
  ⟨(uncurry2 remap-polys-l2, uncurry2 remap-polys-l) ∈ (Id ×r ⟨Id⟩set-rel) ×r Id →f ⟨Id⟩nres-rel⟩
apply (intro frefI fun-relI nres-relI)
using remap-polys-l2-remap-polys-l by auto

```

```

lemma [sepref-fr-rules]:
  ⟨(uncurry2 remap-polys-l-impl,
    uncurry2 remap-polys-l) ∈ poly-assnk *a vars-assnd *a polys-assn-inputd →a
    status-assn raw-string-assn ×a vars-assn ×a polys-assn⟩
using hfcomp-tcomp-pre[OF remap-polys-l2-remap-polys-l remap-polys-l-impl.refine]
by (auto simp: hrp-comp-def hfprod-def)

```

```

sepref-register remap-polys-l

```

```

sepref-definition full-checker-l-impl
  is ⟨uncurry2 full-checker-l
  :: ⟨poly-assnk *a polys-assn-inputd *a (list-assn (pac-step-rel-assn (uint64-nat-assn) poly-assn string-assn))k
  →a
    status-assn raw-string-assn ×a vars-assn ×a polys-assn⟩
  supply [[goals-limit=1]] is-Mult-lastI[intro]
  unfolding full-checker-l-def hs.fold-custom-empty
    union-vars-poly-alt-def[symmetric]
    PAC-checker-l-alt-def
  by sepref

```

```

sepref-definition PAC-update-impl
  is ⟨uncurry2 (RETURN ooo fmupd)
  :: ⟨nat-assnk *a poly-assnk *a (polys-assn-input)d →a polys-assn-input⟩
  unfolding comp-def
  by sepref

```

```

sepref-definition PAC-empty-impl
  is ⟨uncurry0 (RETURN fmempty)
  :: ⟨unit-assnk →a polys-assn-input⟩
  unfolding op-iam-fmap-empty-def[symmetric] pat-fmap-empty
  by sepref

```

```

sepref-definition empty-vars-impl
  is ⟨uncurry0 (RETURN { })
  :: ⟨unit-assnk →a vars-assn⟩
  unfolding hs.fold-custom-empty
  by sepref

```

This is a hack for performance. There is no need to recheck that that a char is valid when working on chars coming from strings... It is not that important in most cases, but in our case the preformance difference is really large.

```

definition unsafe-asciis-of-literal :: ⟨-⟩ where
  ⟨unsafe-asciis-of-literal xs = String.asciis-of-literal xs⟩

```

```

definition unsafe-asciis-of-literal' :: ⟨-⟩ where

```

[simp, symmetric, code]:  $\langle \text{unsafe-asciis-of-literal}' = \text{unsafe-asciis-of-literal} \rangle$

#### code-printing

**constant** *unsafe-asciis-of-literal'*  $\rightarrow$

(SML)  $!(\text{List.map } (fn\ c \Rightarrow \text{let val } k = \text{Char.ord } c \text{ in } \text{IntInf.fromInt } k \text{ end}) \ /o\ \text{String.explode})$

Now comes the big and ugly and unsafe hack.

Basically, we try to avoid the conversion to IntInf when calculating the hash. The performance gain is roughly 40%, which is a LOT and definitively something we need to do. We are aware that the SML semantic encourages compilers to optimise conversions, but this does not happen here, corroborating our early observation on the verified SAT solver IsaSAT.x

#### definition *raw-explode* where

[simp]:  $\langle \text{raw-explode} = \text{String.explode} \rangle$

#### code-printing

**constant** *raw-explode*  $\rightarrow$

(SML)  $\text{String.explode}$

#### definition $\langle \text{hashcode-literal}'\ s \equiv$

$\text{foldl } (\lambda h\ x.\ h * 33 + \text{uint32-of-int } (\text{of-char } x))\ 5381$

$(\text{raw-explode } s) \rangle$

#### lemmas [code] =

*hashcode-literal-def*[*unfolded String.explode-code*

*unsafe-asciis-of-literal-def*[*symmetric*]]

#### definition *uint32-of-char* where

[symmetric, code-unfold]:  $\langle \text{uint32-of-char } x = \text{uint32-of-int } (\text{int-of-char } x) \rangle$

#### code-printing

**constant** *uint32-of-char*  $\rightarrow$

(SML)  $!(\text{Word32.fromInt } /o\ (\text{Char.ord}))$

#### lemma [code]: $\langle \text{hashcode } s = \text{hashcode-literal}'\ s \rangle$

**unfolding** *hashcode-literal-def* *hashcode-list-def*

**apply** (auto simp: *unsafe-asciis-of-literal-def* *hashcode-list-def*

*String.asciis-of-literal-def* *hashcode-literal-def* *hashcode-literal'-def*)

**done**

We do not include

**export-code** *PAC-checker-l-impl* *PAC-update-impl* *PAC-empty-impl* *the-error-is-cfailed-is-cfound*

*int-of-integer* *Del* *Add* *Mult* *nat-of-integer* *String.implode* *remap-polys-l-impl*

*fully-normalize-poly-impl* *union-vars-poly-impl* *empty-vars-impl*

*full-checker-l-impl* *check-step-impl* *CSUCCESS*

*Extension* *hashcode-literal'* *version*

**in** *SML-imp* **module-name** *PAC-Checker*

**file-prefix** *checker*

#### compile-generated-files -

##### external-files

$\langle \text{code}/\text{parser.sml} \rangle$

$\langle \text{code}/\text{pasteque.sml} \rangle$

$\langle \text{code}/\text{pasteque.mlb} \rangle$

**where**  $\langle \text{fn } \text{dir} \Rightarrow$



```

let
  val exec = Generated-Files.execute (Path.append dir (Path.basic code));
  val - = exec ⟨rename file⟩ mv checker.ML checker.sml
  val - =
    exec ⟨Compilation⟩
      (File.bash-path path ⟨$ISABELLE-MLTON⟩ ^ ^
        -const 'MLton.safe false' -verbose 1 -default-type int64 -output pasteqe ^
        -codegen native -inline 700 -cc-opt -O3 pasteqe.mlb);
in () end

```

## 14 Correctness theorem

**context** *poly-embed*  
**begin**

**definition** *full-poly-assn* **where**  
 ⟨full-poly-assn = hr-comp poly-assn (fully-unsorted-poly-rel O mset-poly-rel)⟩

**definition** *full-poly-input-assn* **where**  
 ⟨full-poly-input-assn = hr-comp  
 (hr-comp polys-assn-input  
 ((nat-rel, fully-unsorted-poly-rel O mset-poly-rel) fmap-rel))  
 polys-rel⟩

**definition** *fully-pac-assn* **where**  
 ⟨fully-pac-assn = (list-assn  
 (hr-comp (pac-step-rel-assn uint64-nat-assn poly-assn string-assn)  
 (p2rel  
 ((nat-rel,  
 fully-unsorted-poly-rel O  
 mset-poly-rel, var-rel) pac-step-rel-raw))))⟩

**definition** *code-status-assn* **where**  
 ⟨code-status-assn = hr-comp (status-assn raw-string-assn)  
 code-status-status-rel⟩

**definition** *full-vars-assn* **where**  
 ⟨full-vars-assn = hr-comp (hs.assn string-assn)  
 ((var-rel) set-rel)⟩

**lemma** *polys-rel-full-polys-rel*:  
 ⟨polys-rel-full = Id ×<sub>r</sub> polys-rel⟩  
**by** (auto simp: polys-rel-full-def)

**definition** *full-polys-assn* :: ⟨-⟩ **where**  
 ⟨full-polys-assn = hr-comp (hr-comp polys-assn  
 ((nat-rel,  
 sorted-poly-rel O mset-poly-rel) fmap-rel))  
 polys-rel⟩

Below is the full correctness theorems. It basically states that:

1. assuming that the input polynomials have no duplicate variables

Then:

1. if the checker returns *CFOUND*, the spec is in the ideal and the PAC file is correct
2. if the checker returns *CSUCCESS*, the PAC file is correct (but there is no information on the spec, aka checking failed)
3. if the checker return *CFAILED err*, then checking failed (and *err might* give you an indication of the error, but the correctness theorem does not say anything about that).

The input parameters are:

4. the specification polynom represented as a list
5. the input polynoms as hash map (as an array of option polynom)
6. a representation of the PAC proofs.

**lemma** *PAC-full-correctness*:

$$\begin{aligned} & \langle (\text{uncurry2 } \text{full-checker-l-impl}, \\ & \quad \text{uncurry2 } (\lambda \text{spec } A \cdot \text{PAC-checker-specification spec } A)) \\ & \in (\text{full-poly-assn})^k *_a (\text{full-poly-input-assn})^d *_a (\text{fully-pac-assn})^k \rightarrow_a \text{hr-comp} \\ & \quad (\text{code-status-assn} \times_a \text{full-vars-assn} \times_a \text{hr-comp polys-assn} \\ & \quad \quad (\langle \text{nat-rel}, \text{sorted-poly-rel } O \text{ mset-poly-rel} \rangle \text{fmap-rel})) \\ & \quad \{((st, G), st', G'). \\ & \quad \quad st = st' \wedge (st \neq \text{FAILED} \longrightarrow (G, G') \in \text{Id} \times_r \text{polys-rel})\} \rangle \end{aligned}$$

**using**

```
full-checker-l-impl.refine[FCOMP full-checker-l-full-checker',
  FCOMP full-checker-spec',
  unfolded full-poly-assn-def[symmetric]
  full-poly-input-assn-def[symmetric]
  fully-pac-assn-def[symmetric]
  code-status-assn-def[symmetric]
  full-vars-assn-def[symmetric]
  polys-rel-full-polys-rel
  hr-comp-prod-conv
  full-polys-assn-def[symmetric]]
hr-comp-Id2
by auto
```

It would be more efficient to move the parsing to Isabelle, as this would be more memory efficient (and also reduce the TCB). But now comes the fun part: It cannot work. A stream (of a file) is consumed by side effects. Assume that this would work. The code could look like:

*Let (read-file file) f*

This code is equal to (in the HOL sense of equality): *let - = read-file file in Let (read-file file) f*  
 However, as an hypothetical *read-file* changes the underlying stream, we would get the next token. Remark that this is already a weird point of ML compilers. Anyway, I see currently two solutions to this problem:

1. The meta-argument: use it only in the Refinement Framework in a setup where copies are disallowed. Basically, this works because we can express the non-duplication constraints on the type level. However, we cannot forbid people from expressing things directly at the HOL level.
2. On the target language side, model the stream as the stream and the position. Reading takes two arguments. First, the position to read. Second, the stream (and the current

position) to read. If the position to read does not match the current position, return an error. This would fit the correctness theorem of the code generation (roughly “if it terminates without exception, the answer is the same”), but it is still unsatisfactory.

**end**

**definition**  $\varphi :: \langle \text{string} \Rightarrow \text{nat} \rangle$  **where**  
 $\langle \varphi = (\text{SOME } \varphi. \text{bij } \varphi) \rangle$

**lemma**  $\text{bij-}\varphi$ :  $\langle \text{bij } \varphi \rangle$   
**using**  $\text{someI}[\text{of } \langle \lambda \varphi :: \text{string} \Rightarrow \text{nat}. \text{bij } \varphi \rangle]$   
**unfolding**  $\varphi\text{-def}[\text{symmetric}]$   
**using**  $\text{poly-embed-EX}$   
**by**  $\text{auto}$

**global-interpretation**  $\text{PAC}$ :  $\text{poly-embed}$  **where**  
 $\varphi = \varphi$   
**apply**  $\text{standard}$   
**apply**  $(\text{use } \text{bij-}\varphi \text{ in } \langle \text{auto simp: bij-def} \rangle)$   
**done**

The full correctness theorem is  $(\text{uncurry2 } \text{full-checker-l-impl}, \text{uncurry2 } (\lambda \text{spec } A -. \text{PAC-checker-specification spec } A)) \in \text{PAC.full-poly-assn}^k *_a \text{PAC.full-poly-input-assn}^d *_a \text{PAC.fully-pac-assn}^k \rightarrow_a \text{hr-comp}$   
 $(\text{PAC.code-status-assn} \times_a \text{PAC.full-vars-assn} \times_a \text{hr-comp polys-assn } (\langle \text{nat-rel}, \text{sorted-poly-rel}$   
 $O \text{ PAC.mset-poly-rel} \rangle \text{fmap-rel})) \{((st, G), st', G'). st = st' \wedge (st \neq \text{FAILED} \longrightarrow (G, G') \in$   
 $\text{Id} \times_r \text{polys-rel})\}.$

**end**

## References

- [1] D. Kaufmann, M. Fleury, and A. Biere. The proof checkers pacheck and pasteque for the practical algebraic calculus. In O. Strichman and A. Ivrii, editors, *Formal Methods in Computer-Aided Design, FMCAD 2020, September 21-24, 2020*. IEEE, 2020.