

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

1	Duplicate Free Multisets	2
1.1	More Lists	3
1.2	Generic Multiset	4
1.3	Other	4
1.4	More Theorem about Loops	17
2	Libraries	19
2.1	More Polynoms	19
2.2	More Ideals	23
3	Specification of the PAC checker	24
3.1	Ideals	24
3.2	PAC Format	26
4	Finite maps and multisets	28
4.1	Finite sets and multisets	28
4.2	Finite map and multisets	29
5	Hash-Map for finite mappings	31
5.1	Operations	32
5.2	Patterns	33
5.3	Mapping to Normal Hashmaps	33
6	Checker Algorithm	35
6.1	Specification	35
6.2	Algorithm	37
6.3	Full Checker	42

7	Polynomials of strings	43
7.1	Polynomials and Variables	43
7.2	Addition	44
7.3	Normalisation	45
7.4	Correctness	46
8	Terms	49
8.1	Ordering	49
8.2	Polynomials	49
9	Polynomialss as Lists	52
9.1	Addition	52
9.2	Multiplication	55
9.3	Normalisation	57
9.4	Multiplication and normalisation	60
9.5	Correctness	61
10	Executable Checker	62
10.1	Definitions	62
10.2	Correctness	69
11	Various Refinement Relations	75
12	Initial Normalisation of Polynoms	79
12.1	Sorting	79
12.2	Sorting applied to monomials	80
12.3	Lifting to polynomials	82
13	Code Synthesis of the Complete Checker	89
14	Correctness theorem	100
theory <i>Duplicate-Free-Multiset</i>		
imports <i>Nested-Multisets-Ordinals.Multiset-More</i>		
begin		

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*: $\langle \text{distinct-mset } C \implies \text{remdups-mset } C = C \rangle$
 $\langle \text{proof} \rangle$

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

lemma *distinct-mset-image-mset*:
 $\langle \text{distinct-mset } (\text{image-mset } f (\text{mset } xs)) \longleftrightarrow \text{distinct } (\text{map } f xs) \rangle$
 $\langle \text{proof} \rangle$

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 $\langle False \rangle$
 $\langle proof \rangle$
lemma *distinct-mset-mono*: $\langle D' \subseteq\# D \implies distinct\text{-}mset\ D \implies distinct\text{-}mset\ D' \rangle$
 $\langle proof \rangle$
lemma *distinct-mset-mono-strict*: $\langle D' \subset\# D \implies distinct\text{-}mset\ D \implies distinct\text{-}mset\ D' \rangle$
 $\langle proof \rangle$
lemma *distinct-set-mset-eq-iff*:
assumes $\langle distinct\text{-}mset\ M \rangle\ \langle distinct\text{-}mset\ N \rangle$
shows $\langle set\text{-}mset\ M = set\text{-}mset\ N \longleftrightarrow M = N \rangle$
 $\langle proof \rangle$
lemma *distinct-mset-union2*:
 $\langle distinct\text{-}mset\ (A + B) \implies distinct\text{-}mset\ B \rangle$
 $\langle proof \rangle$
lemma *distinct-mset-mset-set*: $\langle distinct\text{-}mset\ (mset\text{-}set\ A) \rangle$
 $\langle proof \rangle$
lemma *distinct-mset-inter-remdups-mset*:
assumes $dist$: $\langle distinct\text{-}mset\ A \rangle$
shows $\langle A \cap\# remdups\text{-}mset\ B = A \cap\# B \rangle$
 $\langle proof \rangle$
lemma *finite-mset-set-inter*:
 $\langle finite\ A \implies finite\ B \implies mset\text{-}set\ (A \cap B) = mset\text{-}set\ A \cap\# mset\text{-}set\ B \rangle$
 $\langle proof \rangle$
lemma *removeAll-notin*: $\langle a \notin\# A \implies removeAll\text{-}mset\ a\ A = A \rangle$
 $\langle proof \rangle$
lemma *same-mset-distinct-iff*:
 $\langle mset\ M = mset\ M' \implies distinct\ M \longleftrightarrow distinct\ M' \rangle$
 $\langle proof \rangle$

1.1 More Lists

lemma *in-set-conv-iff*:
 $\langle x \in set\ (take\ n\ xs) \longleftrightarrow (\exists i < n. i < length\ xs \wedge xs\ !\ i = x) \rangle$
 $\langle proof \rangle$
lemma *in-set-take-conv-nth*:
 $\langle x \in set\ (take\ n\ xs) \longleftrightarrow (\exists m < min\ n\ (length\ xs). xs\ !\ m = x) \rangle$
 $\langle proof \rangle$
lemma *in-set-remove1D*:
 $\langle a \in set\ (remove1\ x\ xs) \implies a \in set\ xs \rangle$
 $\langle proof \rangle$

1.2 Generic Multiset

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

1.3 Other

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

lemma *Collect-eq-comp'*: $\langle \{ (x, y). P \ x \ y \} \ O \ \{ (c, a). c = f \ a \} = \{ (x, a). P \ x \ (f \ a) \} \rangle$
 $\langle \text{proof} \rangle$

end

theory *WB-Sort*

imports *Refine-Imperative-HOL.HICF 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* :: $\langle ('b \Rightarrow 'b \Rightarrow \text{bool}) \Rightarrow ('a \Rightarrow 'b) \Rightarrow 'a \ \text{list} \Rightarrow \text{nat} \Rightarrow \text{nat} \Rightarrow \text{nat} \ \text{nres} \rangle$ **where**
 $\langle \text{choose-pivot} \ - \ - \ - \ lo \ hi = SPEC(\lambda k. k \geq lo \wedge k \leq hi) \rangle$

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

definition *isPartition-wrt* :: $\langle ('b \Rightarrow 'b \Rightarrow \text{bool}) \Rightarrow 'b \ \text{list} \Rightarrow \text{nat} \Rightarrow \text{nat} \Rightarrow \text{nat} \Rightarrow \text{bool} \rangle$ **where**
 $\langle \text{isPartition-wrt } R \ xs \ lo \ hi \ p \equiv (\forall \ i. i \geq lo \wedge i < p \longrightarrow R \ (xs!i) \ (xs!p)) \wedge (\forall \ j. j > p \wedge j \leq hi \longrightarrow R \ (xs!p) \ (xs!j)) \rangle$

lemma *isPartition-wrtI*:

$\langle (\bigwedge \ i. \llbracket i \geq lo; i < p \rrbracket \Longrightarrow R \ (xs!i) \ (xs!p)) \Longrightarrow (\bigwedge \ j. \llbracket j > p; j \leq hi \rrbracket \Longrightarrow R \ (xs!p) \ (xs!j)) \Longrightarrow$
 $\text{isPartition-wrt } R \ xs \ lo \ hi \ p \rangle$
 $\langle \text{proof} \rangle$

definition *isPartition* :: $\langle 'a :: \text{order list} \Rightarrow \text{nat} \Rightarrow \text{nat} \Rightarrow \text{nat} \Rightarrow \text{bool} \rangle$ **where**
 $\langle \text{isPartition } xs \ lo \ hi \ p \equiv \text{isPartition-wrt } (\leq) \ xs \ lo \ hi \ p \rangle$

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

$\langle \text{isPartition-map } R \ h \ xs \ i \ j \ k \equiv \text{isPartition-wrt } (\lambda a \ b. R \ (h \ a) \ (h \ b)) \ xs \ i \ j \ k \rangle$

lemma *isPartition-map-def'*:

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

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

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

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*:

$l \neq [] \implies \text{take } (\text{Suc } 0) \ l = [!0]$
 $0 < \text{length } l \implies \text{take } (\text{Suc } 0) \ l = [!0]$
 $\text{Suc } n \leq \text{length } l \implies \text{take } (\text{Suc } 0) \ l = [!0]$
 $\langle \text{proof} \rangle$

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

$\langle \text{proof} \rangle$

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

$\langle \text{proof} \rangle$

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

$\langle \text{proof} \rangle$

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

$\langle \text{proof} \rangle$

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

$\langle \text{proof} \rangle$

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

$\langle \text{proof} \rangle$

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'*:

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

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

$\langle \text{proof} \rangle$

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

$\langle \text{proof} \rangle$

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

$\langle \text{proof} \rangle$

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

$\langle \text{proof} \rangle$

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

$\langle \text{proof} \rangle$

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

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

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

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

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

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 $\langle R \ (xs!i) \ (xs!j) \rangle$
 $\langle \text{proof} \rangle$

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

lemma *sorted-sublist-wrt-nth-le'*:
assumes *ref*: $\langle \bigwedge x. R \ x \ x \rangle$
and $\langle \text{sorted-sublist-wrt } R \ xs \ lo \ hi \rangle$ **and** $\langle lo \leq hi \rangle$ **and** $\langle hi < \text{length } xs \rangle$
and $\langle lo \leq i \rangle$ **and** $\langle i \leq j \rangle$ **and** $\langle j \leq hi \rangle$
shows $\langle R \ (xs!i) \ (xs!j) \rangle$
 $\langle \text{proof} \rangle$

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

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

lemma *sublist-cons*: $\langle lo < hi \implies hi < \text{length } xs \implies \text{sublist } xs \ lo \ hi = xs!lo \ # \ \text{sublist } xs \ (\text{Suc } lo) \ hi \rangle$
 $\langle \text{proof} \rangle$

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

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**

$\langle \text{sorted-sublist-wrt } R \text{ } xs \text{ } (lo+1) \text{ } hi \rangle$ **and**
 $\langle lo \leq hi \rangle$ **and** $\langle hi < \text{length } xs \rangle$ **and** $\langle R \text{ } (xs!lo) \text{ } (xs!(lo+1)) \rangle$
shows $\langle \text{sorted-sublist-wrt } R \text{ } xs \text{ } lo \text{ } hi \rangle$
 $\langle \text{proof} \rangle$

lemma *sorted-sublist-map-cons*:

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

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

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

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

lemma *sorted-sublist-wrt-snoc*:

assumes *trans*: $\langle (\bigwedge x \ y \ z. \llbracket R \text{ } x \text{ } y; R \text{ } y \text{ } z \rrbracket \implies R \text{ } x \text{ } z) \rangle$ **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 $\langle \text{sorted-sublist-wrt } R \text{ } xs \text{ } lo \text{ } hi \rangle$
 $\langle \text{proof} \rangle$

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

lemma *sublist-split-part*: $\langle lo \leq hi \implies lo < p \implies p < hi \implies hi < \text{length } xs \implies \text{sublist } xs \text{ } lo \text{ } (p-1) @$
 $\text{ } xs!p \# \text{sublist } xs \text{ } (p+1) \text{ } hi = \text{sublist } xs \text{ } lo \text{ } hi \rangle$
 $\langle \text{proof} \rangle$

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

lemma *isPartition-wrt-trans*:

$\langle (\bigwedge x \ y \ z. \llbracket R \text{ } x \text{ } y; R \text{ } y \text{ } z \rrbracket \implies R \text{ } x \text{ } z) \implies$
 $\text{isPartition-wrt } R \text{ } xs \text{ } lo \text{ } hi \text{ } p \implies$
 $(\forall i \ j. lo \leq i \wedge i < p \wedge p < j \wedge j \leq hi \longrightarrow R \text{ } (xs!i) \text{ } (xs!j)) \rangle$
 $\langle \text{proof} \rangle$

lemma *isPartition-map-trans*:

$\langle (\bigwedge x \ y \ z. \llbracket R \text{ } (h \ x) \text{ } (h \ y); R \text{ } (h \ y) \text{ } (h \ z) \rrbracket \implies R \text{ } (h \ x) \text{ } (h \ z)) \implies$
 $hi < \text{length } xs \implies$
 $\text{isPartition-map } R \text{ } h \text{ } xs \text{ } lo \text{ } hi \text{ } p \implies$
 $(\forall i \ j. lo \leq i \wedge i < p \wedge p < j \wedge j \leq hi \longrightarrow R \text{ } (h \text{ } (xs!i)) \text{ } (h \text{ } (xs!j))) \rangle$
 $\langle \text{proof} \rangle$

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

$\langle lo \leq hi \implies lo < p \implies p < hi \implies hi < \text{length } xs \implies$
 $\text{isPartition-wrt } R \text{ } xs \text{ } lo \text{ } hi \text{ } p \implies$
 $\text{sorted-sublist-wrt } R \text{ } xs \text{ } lo \text{ } (p-1) \implies \text{sorted-sublist-wrt } R \text{ } xs \text{ } (p+1) \text{ } hi \implies$

$(\forall i j. lo \leq i \wedge i < p \wedge p < j \wedge j \leq hi \longrightarrow R (xs!i) (xs!j)) \Longrightarrow$
 $sorted_sublist_wrt R xs lo hi$
 $\langle proof \rangle$

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

$\langle (\bigwedge x y z. \llbracket R x y; R y z \rrbracket \Longrightarrow R x z) \Longrightarrow$
 $isPartition_wrt R xs lo hi p \Longrightarrow$
 $sorted_sublist_wrt R xs lo (p-1) \Longrightarrow sorted_sublist_wrt R xs (p+1) hi \Longrightarrow$
 $lo \leq hi \Longrightarrow hi < length xs \Longrightarrow lo < p \Longrightarrow p < hi \Longrightarrow hi < length xs \Longrightarrow$
 $sorted_sublist_wrt R xs lo hi \rangle$
 $\langle proof \rangle$

The main theorem to merge sorted lists

lemma *merge-sorted-wrt-partitions:*

$\langle isPartition_wrt R xs lo hi p \Longrightarrow$
 $sorted_sublist_wrt R xs lo (p - Suc 0) \Longrightarrow sorted_sublist_wrt R xs (Suc p) hi \Longrightarrow$
 $lo \leq hi \Longrightarrow lo \leq p \Longrightarrow p \leq hi \Longrightarrow hi < length xs \Longrightarrow$
 $(\forall i j. lo \leq i \wedge i < p \wedge p < j \wedge j \leq hi \longrightarrow R (xs!i) (xs!j)) \Longrightarrow$
 $sorted_sublist_wrt R xs lo hi \rangle$
 $\langle proof \rangle$

theorem *merge-sorted-map-partitions:*

$\langle (\bigwedge x y z. \llbracket R (h x) (h y); R (h y) (h z) \rrbracket \Longrightarrow R (h x) (h z)) \Longrightarrow$
 $isPartition_map R h xs lo hi p \Longrightarrow$
 $sorted_sublist_map R h xs lo (p - Suc 0) \Longrightarrow sorted_sublist_map R h xs (Suc p) hi \Longrightarrow$
 $lo \leq hi \Longrightarrow lo \leq p \Longrightarrow p \leq hi \Longrightarrow hi < length xs \Longrightarrow$
 $sorted_sublist_map R h xs lo hi \rangle$
 $\langle proof \rangle$

lemma *partition-wrt-extend:*

$\langle isPartition_wrt R xs lo' hi' p \Longrightarrow$
 $hi < length xs \Longrightarrow$
 $lo \leq lo' \Longrightarrow lo' \leq hi \Longrightarrow hi' \leq hi \Longrightarrow$
 $lo' \leq p \Longrightarrow p \leq hi' \Longrightarrow$
 $(\bigwedge i. lo \leq i \Longrightarrow i < lo' \Longrightarrow R (xs!i) (xs!p)) \Longrightarrow$
 $(\bigwedge j. hi' < j \Longrightarrow j \leq hi \Longrightarrow R (xs!p) (xs!j)) \Longrightarrow$
 $isPartition_wrt R xs lo hi p \rangle$
 $\langle proof \rangle$

lemma *partition-map-extend:*

$\langle isPartition_map R h xs lo' hi' p \Longrightarrow$
 $hi < length xs \Longrightarrow$
 $lo \leq lo' \Longrightarrow lo' \leq hi \Longrightarrow hi' \leq hi \Longrightarrow$
 $lo' \leq p \Longrightarrow p \leq hi' \Longrightarrow$
 $(\bigwedge i. lo \leq i \Longrightarrow i < lo' \Longrightarrow R (h (xs!i)) (h (xs!p))) \Longrightarrow$
 $(\bigwedge j. hi' < j \Longrightarrow j \leq hi \Longrightarrow R (h (xs!p)) (h (xs!j))) \Longrightarrow$
 $isPartition_map R h xs lo hi p \rangle$
 $\langle proof \rangle$

lemma *isPartition-empty:*

$\langle (\bigwedge j. \llbracket lo < j; j \leq hi \rrbracket \Longrightarrow R (xs ! lo) (xs ! j)) \Longrightarrow$
 $isPartition_wrt R xs lo hi lo \rangle$
 $\langle proof \rangle$

lemma *take-ext*:

$\langle (\forall i < k. xs!i = xs!i) \implies$
 $k < \text{length } xs \implies k < \text{length } xs' \implies$
 $\text{take } k \text{ } xs' = \text{take } k \text{ } xs \rangle$
 $\langle \text{proof} \rangle$

lemma *drop-ext'*:

$\langle (\forall i. i \geq k \wedge i < \text{length } xs \longrightarrow xs!i = xs!i) \implies$
 $0 < k \implies xs \neq [] \implies \text{— These corner cases will be dealt with in the next lemma}$
 $\text{length } xs' = \text{length } xs \implies$
 $\text{drop } k \text{ } xs' = \text{drop } k \text{ } xs \rangle$
 $\langle \text{proof} \rangle$

lemma *drop-ext*:

$\langle (\forall i. i \geq k \wedge i < \text{length } xs \longrightarrow xs!i = xs!i) \implies$
 $\text{length } xs' = \text{length } xs \implies$
 $\text{drop } k \text{ } xs' = \text{drop } k \text{ } xs \rangle$
 $\langle \text{proof} \rangle$

lemma *sublist-ext'*:

$\langle (\forall i. lo \leq i \wedge i \leq hi \longrightarrow xs!i = xs!i) \implies$
 $\text{length } xs' = \text{length } xs \implies$
 $lo \leq hi \implies \text{Suc } hi < \text{length } xs \implies$
 $\text{sublist } xs' \text{ } lo \text{ } hi = \text{sublist } xs \text{ } lo \text{ } hi \rangle$
 $\langle \text{proof} \rangle$

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

$\langle \text{proof} \rangle$

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

$\langle \text{proof} \rangle$

lemma *sublist-ext*:

$\langle (\forall i. lo \leq i \wedge i \leq hi \longrightarrow xs!i = xs!i) \implies$
 $\text{length } xs' = \text{length } xs \implies$
 $lo \leq hi \implies hi < \text{length } xs \implies$
 $\text{sublist } xs' \text{ } lo \text{ } hi = \text{sublist } xs \text{ } lo \text{ } hi \rangle$
 $\langle \text{proof} \rangle$

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** $\langle \text{length } xs' = \text{length } xs \rangle$
shows $\langle \text{sorted-sublist-wrt } R \text{ } xs' \text{ } lo \text{ } (lo' - \text{Suc } 0) \rangle$
 $\langle \text{proof} \rangle$

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

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

shows $\langle \text{sorted-sublist-map } R \ h \ xs' \ lo \ (lo' - \text{Suc } 0) \rangle$
 $\langle \text{proof} \rangle$

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

assumes $\langle \text{sorted-sublist-wrt } R \ xs \ (hi'+1) \ hi \rangle$ **and**
 $\langle lo \leq lo' \rangle$ **and** $\langle hi < \text{length } xs \rangle$ **and**
 $\langle \forall j. hi' < j \wedge j \leq hi \longrightarrow xs!j = xs!j \rangle$ **and** $\langle \text{length } xs' = \text{length } xs \rangle$
shows $\langle \text{sorted-sublist-wrt } R \ xs' \ (hi'+1) \ hi \rangle$
 $\langle \text{proof} \rangle$

lemma *sorted-map-upper-sublist-still-sorted*:

assumes $\langle \text{sorted-sublist-map } R \ h \ xs \ (hi'+1) \ hi \rangle$ **and**
 $\langle lo \leq lo' \rangle$ **and** $\langle hi < \text{length } xs \rangle$ **and**
 $\langle \forall j. hi' < j \wedge j \leq hi \longrightarrow xs!j = xs!j \rangle$ **and** $\langle \text{length } xs' = \text{length } xs \rangle$
shows $\langle \text{sorted-sublist-map } R \ h \ xs' \ (hi'+1) \ hi \rangle$
 $\langle \text{proof} \rangle$

The specification of the partition function

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

$\langle \text{partition-spec } R \ h \ xs \ lo \ hi \ xs' \ p \equiv$
 $\text{mset } xs' = \text{mset } xs \wedge \text{--- The list is a permutation}$
 $\text{isPartition-map } R \ h \ xs' \ lo \ hi \ p \wedge \text{--- We have a valid partition on the resulting list}$
 $lo \leq p \wedge p \leq hi \wedge \text{--- The partition index is in bounds}$
 $(\forall i. i < lo \longrightarrow xs!i = xs!i) \wedge (\forall i. hi < i \wedge i < \text{length } xs' \longrightarrow xs!i = xs!i) \rangle$ **--- Everything else is unchanged.**

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

$\langle x \in \text{set } (\text{take } n \ xs) \longleftrightarrow (\exists m < \min n \ (\text{length } xs). \ xs!m = x) \rangle$
 $\langle \text{proof} \rangle$

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

$\langle \text{proof} \rangle$

lemma *mathias*:

assumes
 $\text{Perm}: \langle \text{mset } xs' = \text{mset } xs \rangle$
and $I: \langle lo \leq i \rangle \langle i \leq hi \rangle \langle xs!i = x \rangle$
and $\text{Bounds}: \langle hi < \text{length } xs \rangle$
and $\text{Fix}: \langle \bigwedge i. i < lo \implies xs!i = xs!i \rangle \langle \bigwedge j. \llbracket hi < j; j < \text{length } xs \rrbracket \implies xs!j = xs!j \rangle$
shows $\langle \exists j. lo \leq j \wedge j \leq hi \wedge xs!j = x \rangle$
 $\langle \text{proof} \rangle$

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 $\text{Perm}: \langle \text{mset } xs' = \text{mset } xs \rangle$
and $\text{Fix}: \langle \bigwedge i. i < lo \implies xs!i = xs!i \rangle \langle \bigwedge j. \llbracket hi < j; j < \text{length } xs \rrbracket \implies xs!j = xs!j \rangle$
and $\text{bounds}: \langle lo \leq hi \rangle \langle hi < \text{length } xs \rangle$
shows $\langle \text{set } (\text{sublist } xs' \ lo \ hi) \subseteq \text{set } (\text{sublist } xs \ lo \ hi) \rangle$
 $\langle \text{proof} \rangle$

lemma *mset-sublist-eq*:

assumes $\langle mset\ xs' = mset\ xs \rangle$
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*: $\langle lo \leq hi \rangle \langle hi < length\ xs \rangle$
shows $\langle set\ (sublist\ xs'\ lo\ hi) = set\ (sublist\ xs\ lo\ hi) \rangle$
 $\langle proof \rangle$

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

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

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

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

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

Convert Pure to HOL

lemma *quicksort-postI*:
 $\langle \llbracket mset\ xs' = mset\ xs; sorted-sublist-map\ R\ h\ xs'\ lo\ hi; (\bigwedge i. \llbracket i < lo \rrbracket \implies xs!i = xs!i); (\bigwedge j. \llbracket hi < j; j < length\ xs \rrbracket \implies xs!j = xs!j) \rrbracket \implies quicksort-post\ R\ h\ lo\ hi\ xs\ xs' \rangle$
 $\langle proof \rangle$

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*: $\langle quicksort-pre\ R\ h\ xs0\ lo\ hi\ xs \rangle$
and *part*: $\langle partition-spec\ R\ h\ xs\ lo\ hi\ xs'\ p \rangle$
and *ifs*: $\langle p-1 \leq lo \rangle \langle hi \leq p+1 \rangle$
shows $\langle quicksort-post\ R\ h\ lo\ hi\ xs\ xs' \rangle$
 $\langle proof \rangle$

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

lemma quicksort-post-set:
 assumes $\langle \text{quicksort-post } R \ h \ lo \ hi \ xs \ xs' \rangle$
 and bounds: $\langle lo \leq hi \rangle \langle hi < \text{length } xs \rangle$
 shows $\langle \text{set } (\text{sublist } xs' \ lo \ hi) = \text{set } (\text{sublist } xs \ lo \ hi) \rangle$
 $\langle \text{proof} \rangle$

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: $\langle \text{quicksort-pre } R \ h \ xs0 \ lo \ hi \ xs \rangle$
 and part: $\langle \text{partition-spec } R \ h \ xs \ lo \ hi \ xs' \ p \rangle$
 and ifs: $\langle p - Suc \ 0 \leq lo \rangle \langle \neg hi \leq Suc \ p \rangle$
 and IH1': $\langle \text{quicksort-post } R \ h \ (Suc \ p) \ hi \ xs' \ xs'' \rangle$
 shows $\langle \text{quicksort-post } R \ h \ lo \ hi \ xs \ xs'' \rangle$
 $\langle \text{proof} \rangle$

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

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

lemma quicksort-correct-case5:
 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: $\langle \neg p - Suc \ 0 \leq lo \rangle \langle hi \leq Suc \ p \rangle$
 and IH1': $\langle \text{quicksort-post } R \ h \ lo \ (p - Suc \ 0) \ xs' \ xs'' \rangle$
 shows $\langle \text{quicksort-post } R \ h \ lo \ hi \ xs \ xs'' \rangle$
 $\langle \text{proof} \rangle$

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: $\langle \text{quicksort-pre } R \ h \ xs0 \ lo \ hi \ xs \rangle$

and part: $\langle \text{partition-spec } R \ h \ xs \ lo \ hi \ xs' \ p \rangle$
and ifs: $\langle \neg p - Suc \ 0 \leq lo \rangle \langle \neg hi \leq Suc \ p \rangle$
and IH1: $\langle \text{quicksort-post } R \ h \ lo \ (p - Suc \ 0) \ xs' \ xs'' \rangle$
shows $\langle \text{quicksort-pre } R \ h \ xs0 \ (Suc \ p) \ hi \ xs'' \rangle$
 $\langle \text{proof} \rangle$

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:* $\langle \bigwedge x \ y \ z. \llbracket R \ (h \ x) \ (h \ y); R \ (h \ y) \ (h \ z) \rrbracket \implies R \ (h \ x) \ (h \ z) \rangle$ **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: $\langle \neg p - Suc \ 0 \leq lo \rangle \langle \neg hi \leq Suc \ p \rangle$
and IH1': $\langle \text{quicksort-post } R \ h \ lo \ (p - Suc \ 0) \ xs' \ xs'' \rangle$
and IH2': $\langle \text{quicksort-post } R \ h \ (Suc \ p) \ hi \ xs'' \ xs''' \rangle$
shows $\langle \text{quicksort-post } R \ h \ lo \ hi \ xs \ xs''' \rangle$
 $\langle \text{proof} \rangle$

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: $\langle lo0 \leq hi0 \rangle \langle hi0 < length \ xs0 \rangle$
shows $\langle \text{quicksort } R \ h \ (lo0, hi0, xs0) \leq \Downarrow Id \ (SPEC(\lambda xs. \text{quicksort-post } R \ h \ lo0 \ hi0 \ xs0 \ xs)) \rangle$
 $\langle \text{proof} \rangle$

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

$\langle \text{partition-main-inv } R \ h \ lo \ hi \ xs0 \ p \equiv$
 $\text{case } p \text{ of } (i, j, xs) \Rightarrow$
 $j < length \ xs \wedge j \leq hi \wedge i < length \ xs \wedge lo \leq i \wedge i \leq j \wedge mset \ xs = mset \ xs0 \wedge$
 $(\forall k. k \geq lo \wedge k < i \longrightarrow R \ (h \ (xs!k)) \ (h \ (xs!hi))) \wedge$ — 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
 $(\forall k. k \geq j \wedge k < length \ xs \longrightarrow xs!k = xs0!k)$ — All elements from j are unchanged (including everything above hi)
 \rangle

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

$\langle \text{partition-main } R \ h \ lo \ hi \ xs0 = \text{do } \{$
 $\text{ASSERT}(hi < length \ xs0);$
 $\text{pivot} \leftarrow \text{RETURN } (h \ (xs0 ! hi));$
 $(i, j, xs) \leftarrow \text{WHILE}_T \text{partition-main-inv } R \ h \ lo \ hi \ xs0$ — We loop from $j = lo$ to $j = hi - (1::'c)$.
 \rangle

```

( $\lambda(i,j,xs). j < hi$ )
( $\lambda(i,j,xs). do \{$ 
   $ASSERT(i < length\ xs \wedge j < length\ xs);$ 
   $if\ R\ (h\ (xs!j))\ pivot$ 
   $then\ RETURN\ (i+1, j+1, swap\ xs\ i\ j)$ 
   $else\ RETURN\ (i, j+1, xs)$ 
 $\})$ 
( $lo, lo, xs0$ ); —  $i$  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 $bounds: \langle hi < length\ xs \rangle \langle lo \leq hi \rangle$ **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 $\langle partition-main\ R\ h\ lo\ hi\ xs \leq SPEC(\lambda(xs', p). mset\ xs = mset\ xs' \wedge$
 $lo \leq p \wedge p \leq hi \wedge isPartition-map\ R\ h\ xs'\ lo\ hi\ p \wedge (\forall i. i < lo \longrightarrow xs!i = xs!i) \wedge (\forall i. hi < i \wedge i < length\ xs' \longrightarrow xs!i = xs!i)) \rangle$
 $\langle proof \rangle$

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

```

 $\langle partition-between\ R\ h\ lo\ hi\ xs0 = do \{$ 
   $ASSERT(hi < length\ xs0 \wedge lo \leq hi);$ 
   $k \leftarrow choose-pivot\ R\ h\ xs0\ lo\ hi;$  — choice of pivot
   $ASSERT(k < length\ xs0);$ 
   $xs \leftarrow RETURN\ (swap\ xs0\ k\ hi);$  — 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-between-correct*:

assumes $\langle hi < length\ xs \rangle$ **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 $\langle partition-between\ R\ h\ lo\ hi\ xs \leq SPEC(uncurry\ (partition-spec\ R\ h\ xs\ lo\ hi)) \rangle$
 $\langle proof \rangle$

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

definition *choose-pivot3* **where**

```

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

```

$\text{else if } (R \text{ start end} \wedge R \text{ end mid}) \vee (R \text{ mid end} \wedge R \text{ end start}) \text{ then RETURN hi}$
 else RETURN lo
 \rangle

— 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 $\langle \text{choose-pivot3 } R \ h \ xs \ lo \ hi \leq \Downarrow Id \ (\text{choose-pivot } R \ h \ xs \ lo \ hi) \rangle$
 $\langle \text{proof} \rangle$

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

definition *partition-between-ref*

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

where

$\langle \text{partition-between-ref } R \ h \ lo \ hi \ xs0 = \text{do} \{$
 $\text{ASSERT}(hi < \text{length } xs0 \wedge hi < \text{length } xs0 \wedge lo \leq hi);$
 $k \leftarrow \text{choose-pivot3 } R \ h \ xs0 \ lo \ hi; \text{ — choice of pivot}$
 $\text{ASSERT}(k < \text{length } xs0);$
 $xs \leftarrow \text{RETURN } (\text{swap } xs0 \ k \ hi); \text{ — move the pivot to the last position, before we start the actual}$
 loop
 $\text{ASSERT}(\text{length } xs = \text{length } xs0);$
 $\text{partition-main } R \ h \ lo \ hi \ xs$
 $\} \rangle$

lemma *partition-main-ref'*:

$\langle \text{partition-main } R \ h \ lo \ hi \ xs$
 $\leq \Downarrow ((\lambda a \ b \ c \ d. Id) \ a \ b \ c \ d) \ (\text{partition-main } R \ h \ lo \ hi \ xs) \rangle$
 $\langle \text{proof} \rangle$

lemma *Down-id-eq*:

$\langle \Downarrow Id \ x = x \rangle$
 $\langle \text{proof} \rangle$

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

$\langle \text{partition-between-ref } R \ h \ lo \ hi \ xs \leq (\text{partition-between } R \ h \ lo \ hi \ xs) \rangle$
 $\langle \text{proof} \rangle$

Technical lemma for sepref

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

$\langle (\text{uncurry2 } (\text{partition-between-ref } R \ h), \text{uncurry2 } (\text{partition-between } R \ h)) \in$
 $(\text{nat-rel} \times_r \text{nat-rel}) \times_r \langle Id \rangle \text{list-rel} \rightarrow_f \langle \langle Id \rangle \text{list-rel} \times_r \text{nat-rel} \rangle \text{nres-rel} \rangle$
 $\langle \text{proof} \rangle$

Example instantiation for pivot

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

$\langle \text{choose-pivot3-impl} = \text{choose-pivot3 } (\leq) \text{ id} \rangle$

lemma *partition-between-ref-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*: $\langle \bigwedge x \ y. R \ (h \ x) \ (h \ y) \vee R \ (h \ y) \ (h \ x) \rangle$

and *bounds*: $\langle hi < \text{length } xs \rangle \langle lo \leq hi \rangle$
shows $\langle \text{partition-between-ref } R \ h \ lo \ hi \ xs \leq SPEC \ (\text{uncurry } (\text{partition-spec } R \ h \ xs \ lo \ hi)) \rangle$
 $\langle \text{proof} \rangle$

Refined quicksort algorithm: We use the refined partition function.

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

$\langle \text{quicksort-ref } R \ h = (\lambda(lo, hi, xs0).$

$\text{do } \{$

$RECT \ (\lambda f \ (lo, hi, xs). \text{do } \{$

$ASSERT(lo \leq hi \wedge hi < \text{length } xs0 \wedge mset \ xs = mset \ xs0);$

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

$ASSERT(mset \ xs = mset \ xs0 \wedge p \geq lo \wedge p < \text{length } xs0);$

$xs \leftarrow (if \ p-1 \leq lo \text{ then } RETURN \ xs \text{ else } f \ (lo, p-1, xs));$

$ASSERT(mset \ xs = mset \ xs0);$

$if \ hi \leq p+1 \text{ then } RETURN \ xs \text{ else } f \ (p+1, hi, xs)$

$\}) \ (lo, hi, xs0)$

$\}) \rangle$

lemma *fref-to-Down-curry2*:

$\langle (\text{uncurry2 } f, \text{uncurry2 } g) \in [P]_f \ A \rightarrow \langle B \rangle \text{nres-rel} \implies$

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

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

$\langle \text{proof} \rangle$

lemma *fref-to-Down-curry*:

$\langle (f, g) \in [P]_f \ A \rightarrow \langle B \rangle \text{nres-rel} \implies$

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

$f \ x \leq \Downarrow B \ (g \ x')) \rangle$

$\langle \text{proof} \rangle$

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 $\langle \text{quicksort-ref } R \ h \ xs0 \leq \Downarrow Id \ (\text{quicksort } R \ h \ xs0) \rangle$

$\langle \text{proof} \rangle$

definition *full-quicksort* **where**

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

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

$\langle \text{full-quicksort-ref } R \ h \ xs \equiv$

$if \ List.null \ xs \text{ then } RETURN \ xs$

$else \ \text{quicksort-ref } R \ h \ (0, \text{length } xs - 1, xs) \rangle$

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

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

lemma *sublist-entire*:
 $\langle \text{sublist } xs \ 0 \ (\text{length } xs - 1) = xs \rangle$
 $\langle \text{proof} \rangle$

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

lemma *sorted-sublist-map-entire*:
assumes $\langle \text{sorted-sublist-map } R \ h \ xs \ 0 \ (\text{length } xs - 1) \rangle$
shows $\langle \text{sorted-wrt } (\lambda x \ y. R \ (h \ x) \ (h \ y)) \ xs \rangle$
 $\langle \text{proof} \rangle$

Final correctness lemma

theorem *full-quicksort-correct-sorted*:
assumes
 $\text{trans: } \langle \bigwedge x \ y \ z. \llbracket R \ (h \ x) \ (h \ y); R \ (h \ y) \ (h \ z) \rrbracket \implies R \ (h \ x) \ (h \ z) \rangle$ **and** $\text{lin: } \langle \bigwedge x \ y. x \neq y \implies R \ (h \ x) \ (h \ y) \vee R \ (h \ y) \ (h \ x) \rangle$
shows $\langle \text{full-quicksort } R \ h \ xs \leq \Downarrow \text{Id} \ (\text{SPEC}(\lambda xs'. \text{mset } xs' = \text{mset } xs \wedge \text{sorted-wrt } (\lambda x \ y. R \ (h \ x) \ (h \ y)) \ xs')) \rangle$
 $\langle \text{proof} \rangle$

lemma *full-quicksort-correct*:
assumes
 $\text{trans: } \langle \bigwedge x \ y \ z. \llbracket R \ (h \ x) \ (h \ y); R \ (h \ y) \ (h \ z) \rrbracket \implies R \ (h \ x) \ (h \ z) \rangle$ **and**
 $\text{lin: } \langle \bigwedge x \ y. R \ (h \ x) \ (h \ y) \vee R \ (h \ y) \ (h \ x) \rangle$
shows $\langle \text{full-quicksort } R \ h \ xs \leq \Downarrow \text{Id} \ (\text{SPEC}(\lambda xs'. \text{mset } xs' = \text{mset } xs)) \rangle$
 $\langle \text{proof} \rangle$

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*:
 $\langle \Downarrow \text{Id } x = x \rangle$
 $\langle \text{proof} \rangle$

lemma *while-upt-while-direct1*:
 $b \geq a \implies$
 $\text{do } \{$

$(-, \sigma) \leftarrow \text{WHILE}_T (\text{FOREACH-cond } c) (\lambda x. \text{do } \{ \text{ASSERT } (\text{FOREACH-cond } c \ x); \text{FOREACH-body } f \ x \})$
 $([a..<b], \sigma);$
 $\text{RETURN } \sigma$
 $\} \leq \text{do } \{$
 $(-, \sigma) \leftarrow \text{WHILE}_T (\lambda(i, x). i < b \wedge c \ x) (\lambda(i, x). \text{do } \{ \text{ASSERT } (i < b); \sigma' \leftarrow f \ i \ x; \text{RETURN } (i+1, \sigma')$
 $\}) (a, \sigma);$
 $\text{RETURN } \sigma$
 $\}$
 $\langle \text{proof} \rangle$

lemma *while-upt-while-direct2*:

$b \geq a \implies$
 $\text{do } \{$
 $(-, \sigma) \leftarrow \text{WHILE}_T (\text{FOREACH-cond } c) (\lambda x. \text{do } \{ \text{ASSERT } (\text{FOREACH-cond } c \ x); \text{FOREACH-body } f \ x \})$
 $([a..<b], \sigma);$
 $\text{RETURN } \sigma$
 $\} \geq \text{do } \{$
 $(-, \sigma) \leftarrow \text{WHILE}_T (\lambda(i, x). i < b \wedge c \ x) (\lambda(i, x). \text{do } \{ \text{ASSERT } (i < b); \sigma' \leftarrow f \ i \ x; \text{RETURN } (i+1, \sigma')$
 $\}) (a, \sigma);$
 $\text{RETURN } \sigma$
 $\}$
 $\langle \text{proof} \rangle$

lemma *while-upt-while-direct*:

$b \geq a \implies$
 $\text{do } \{$
 $(-, \sigma) \leftarrow \text{WHILE}_T (\text{FOREACH-cond } c) (\lambda x. \text{do } \{ \text{ASSERT } (\text{FOREACH-cond } c \ x); \text{FOREACH-body } f \ x \})$
 $([a..<b], \sigma);$
 $\text{RETURN } \sigma$
 $\} = \text{do } \{$
 $(-, \sigma) \leftarrow \text{WHILE}_T (\lambda(i, x). i < b \wedge c \ x) (\lambda(i, x). \text{do } \{ \text{ASSERT } (i < b); \sigma' \leftarrow f \ i \ x; \text{RETURN } (i+1, \sigma')$
 $\}) (a, \sigma);$
 $\text{RETURN } \sigma$
 $\}$
 $\langle \text{proof} \rangle$

lemma *while-nfoldli*:

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

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

\leq
 $(\text{WHILE}_T^I$
 $(\text{FOREACH-cond } c) (\lambda x. \text{do } \{ \text{ASSERT } (\text{FOREACH-cond } c \ x); \text{FOREACH-body } f \ x \}) (l, \sigma)$
 $\gg=$
 $(\lambda(-, \sigma). \text{RETURN } \sigma))$
 $\langle \text{proof} \rangle$

lemma *while-eq-nfoldli*: $\text{do } \{$

```

  ( $\cdot, \sigma$ )  $\leftarrow$  WHILET (FOREACH-cond  $c$ ) ( $\lambda x. \text{do } \{ \text{ASSERT } (\text{FOREACH-cond } c \ x); \text{FOREACH-body } f \ x \}$ ) ( $l, \sigma$ );
  RETURN  $\sigma$ 
} = nfoldli  $l \ c \ f \ \sigma$ 
<proof>

```

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 Const₀-add:

```

<Const0 (a + b) = Const0 a + Const0 b>
<proof>

```

lemma Const-mult:

```

<Const (a * b) = Const a * Const b>
<proof>

```

lemma Const₀-mult:

```

<Const0 (a * b) = Const0 a * Const0 b>
<proof>

```

lemma Const0[simp]:

```

<Const 0 = 0>
<proof>

```

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

```

<Const (-n) = - Const n>
<proof>

```

lemma [simp]: <Const₀ 0 = 0>

```

<MPoly 0 = 0>
<proof>

```

lemma Const-add:

```

<Const (a + b) = Const a + Const b>
<proof>

```

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

```

<proof>

```

lemma degree-uminus[simp]:

```

<degree (-A) x' = degree A x'>
<proof>

```

lemma *degree-sum-notin*:

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

lemma *degree-notin-vars*:

$\langle x \notin (\text{vars } B) \implies \text{degree } (B :: 'a :: \{\text{monoid-add}\} \text{ mpoly}) \ x = 0 \rangle$
 $\langle \text{proof} \rangle$

lemma *not-in-vars-coeff0*:

$\langle x \notin \text{vars } p \implies \text{MPoly-Type.coeff } p \ (\text{monomial } (\text{Suc } 0) \ x) = 0 \rangle$
 $\langle \text{proof} \rangle$

lemma *keys-mapping-sum-add*:

$\langle \text{finite } A \implies \text{keys } (\text{mapping-of } (\sum v \in A. f \ v)) \subseteq \bigcup (\text{keys } ' \text{ mapping-of } ' f ' \text{ UNIV}) \rangle$
 $\langle \text{proof} \rangle$

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 $\langle \text{vars } (\sum v \mid f \ v \neq 0. f \ v * v) \subseteq \bigcup (\text{vars } ' \{v. f \ v \neq 0\}) \cup \bigcup (\text{vars } ' f ' \{v. f \ v \neq 0\}) \rangle$
 $\langle \text{is } \langle ?A \subseteq ?B \rangle \rangle$
 $\langle \text{proof} \rangle$

lemma *vars-in-right-only*:

$\langle x \in \text{vars } q \implies x \notin \text{vars } p \implies x \in \text{vars } (p + q) \rangle$
 $\langle \text{proof} \rangle$

lemma *[simp]*:

$\langle \text{vars } 0 = \{\} \rangle$
 $\langle \text{proof} \rangle$

lemma *vars-Un-nointer*:

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

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

lemma *polynom-sum-monoms*:

fixes $p :: \langle 'a :: \{\text{comm-monoid-add}, \text{cancel-comm-monoid-add}\} \text{ mpoly} \rangle$
shows
 $\langle p = (\sum x \in \text{keys } (\text{mapping-of } p). \text{MPoly-Type.monom } x \ (\text{MPoly-Type.coeff } p \ x)) \rangle$
 $\langle \text{keys } (\text{mapping-of } p) \subseteq I \implies \text{finite } I \implies p = (\sum x \in I. \text{MPoly-Type.monom } x \ (\text{MPoly-Type.coeff } p \ x)) \rangle$
 $\langle \text{proof} \rangle$

lemma *vars-mult-monom*:

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

lemma *in-mapping-mult-single*:

$\langle x \in (\lambda x. \text{lookup } x \ x') ' \text{keys } (A * (\text{Var}_0 \ x' :: (\text{nat} \Rightarrow_0 \text{nat}) \Rightarrow_0 'b :: \{\text{monoid-mult}, \text{zero-neq-one}, \text{semiring-0}\})) \rangle$

\longleftrightarrow
 $x > 0 \wedge x - 1 \in (\lambda x. \text{lookup } x \ x') \text{ 'keys } (A)$
 $\langle \text{proof} \rangle$

lemma *Max-Suc-Suc-Max*:
 $\langle \text{finite } A \implies A \neq \{\} \implies \text{Max } (\text{insert } 0 \ (\text{Suc } 'A)) =$
 $\text{Suc } (\text{Max } (\text{insert } 0 \ A)) \rangle$
 $\langle \text{proof} \rangle$

lemma *[simp]*:
 $\langle \text{keys } (\text{Var}_0 \ x' :: ('a \Rightarrow_0 \text{nat}) \Rightarrow_0 'b :: \{\text{zero-neq-one}\}) = \{\text{Poly-Mapping.single } x' \ 1\} \rangle$
 $\langle \text{proof} \rangle$

lemma *degree-mult-Var*:
 $\langle \text{degree } (A * \text{Var } x') \ x' = (\text{if } A = 0 \text{ then } 0 \text{ else } \text{Suc } (\text{degree } A \ x')) \rangle \text{ for } A :: \langle \text{int mpoly} \rangle$
 $\langle \text{proof} \rangle$

lemma *degree-mult-Var'*:
 $\langle \text{degree } (\text{Var } x' * A) \ x' = (\text{if } A = 0 \text{ then } 0 \text{ else } \text{Suc } (\text{degree } A \ x')) \rangle \text{ for } A :: \langle \text{int mpoly} \rangle$
 $\langle \text{proof} \rangle$

lemma *degree-add-max*:
 $\langle \text{degree } (A + B) \ x \leq \max (\text{degree } A \ x) (\text{degree } B \ x) \rangle$
 $\langle \text{proof} \rangle$

lemma *degree-times-le*:
 $\langle \text{degree } (A * B) \ x \leq \text{degree } A \ x + \text{degree } B \ x \rangle$
 $\langle \text{proof} \rangle$

lemma *monomial-inj*:
 $\text{monomial } c \ s = \text{monomial } (d :: 'b :: \text{zero-neq-one}) \ t \longleftrightarrow (c = 0 \wedge d = 0) \vee (c = d \wedge s = t)$
 $\langle \text{proof} \rangle$

lemma *MPoly-monomial-power'*:
 $\langle \text{MPoly } (\text{monomial } 1 \ x') \wedge (n+1) = \text{MPoly } (\text{monomial } (1) (((\lambda x. x + x') \rightsquigarrow^n) \ x')) \rangle$
 $\langle \text{proof} \rangle$

lemma *MPoly-monomial-power*:
 $\langle n > 0 \implies \text{MPoly } (\text{monomial } 1 \ x') \wedge (n) = \text{MPoly } (\text{monomial } (1) (((\lambda x. x + x') \rightsquigarrow^{(n-1)}) \ x')) \rangle$
 $\langle \text{proof} \rangle$

lemma *vars-uminus[simp]*:
 $\langle \text{vars } (-p) = \text{vars } p \rangle$
 $\langle \text{proof} \rangle$

lemma *coeff-uminus[simp]*:
 $\langle \text{MPoly-Type.coeff } (-p) \ x = -\text{MPoly-Type.coeff } p \ x \rangle$
 $\langle \text{proof} \rangle$

definition *decrease-key*:: $'a \Rightarrow ('a \Rightarrow_0 'b :: \{\text{monoid-add, minus, one}\}) \Rightarrow ('a \Rightarrow_0 'b)$ **where**
 $\text{decrease-key } k0 \ f = \text{Abs-poly-mapping } (\lambda k. \text{if } k = k0 \wedge \text{lookup } f \ k \neq 0 \text{ then } \text{lookup } f \ k - 1 \text{ else } \text{lookup } f \ k)$

lemma *remove-key-lookup*:

lookup (*decrease-key* *k0 f*) *k* = (if *k* = *k0* \wedge *lookup f k* \neq 0 then *lookup f k* - 1 else *lookup f k*)
 ⟨*proof*⟩

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**

⟨*p* = *monom* (*monomial* (*Suc 0*) *x'*) 1 * *q* + *r*⟩ **and**

⟨*x'* \notin *vars r*⟩

⟨*proof*⟩

lemma *polynom-split-on-var2*:

fixes *p* :: ⟨*int mpoly*⟩

assumes ⟨*x'* \notin *vars s*⟩

obtains *q r* **where**

⟨*p* = (*monom* (*monomial* (*Suc 0*) *x'*) 1 - *s*) * *q* + *r*⟩ **and**

⟨*x'* \notin *vars r*⟩

⟨*proof*⟩

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'*) 1² - *monom* (*monomial* (*Suc 0*) *x'*) 1)⟩ **and**

⟨*x'* \notin *vars r*⟩ **and**

⟨*x'* \notin *vars q*⟩

⟨*proof*⟩

lemma *polynom-decomp-alien-var*:

fixes *q A b* :: ⟨*int mpoly*⟩

assumes

q: ⟨*q* = *A* * (*monom* (*monomial* (*Suc 0*) *x'*) 1) + *b*⟩ **and**

x: ⟨*x'* \notin *vars q*⟩ ⟨*x'* \notin *vars b*⟩

shows

⟨*A* = 0⟩ **and**

⟨*q* = *b*⟩

⟨*proof*⟩

lemma *polynom-decomp-alien-var2*:

fixes *q A b* :: ⟨*int mpoly*⟩

assumes

q: ⟨*q* = *A* * (*monom* (*monomial* (*Suc 0*) *x'*) 1 + *p*) + *b*⟩ **and**

x: ⟨*x'* \notin *vars q*⟩ ⟨*x'* \notin *vars b*⟩ ⟨*x'* \notin *vars p*⟩

shows

⟨*A* = 0⟩ **and**

⟨*q* = *b*⟩

⟨*proof*⟩

lemma *vars-unE*: ⟨*x* \in *vars* (*a* * *b*)⟩ \implies (*x* \in *vars* *a* \implies *thesis*) \implies (*x* \in *vars* *b* \implies *thesis*) \implies *thesis*⟩

⟨*proof*⟩

lemma *in-keys-minusI1*:

assumes $t \in \text{keys } p$ **and** $t \notin \text{keys } q$
shows $t \in \text{keys } (p - q)$
 $\langle \text{proof} \rangle$

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 $t \in \text{keys } (p - q)$
 $\langle \text{proof} \rangle$

lemma *in-vars-addE*:
 $\langle x \in \text{vars } (p + q) \Rightarrow (x \in \text{vars } p \Rightarrow \text{thesis}) \Rightarrow (x \in \text{vars } q \Rightarrow \text{thesis}) \Rightarrow \text{thesis} \rangle$
 $\langle \text{proof} \rangle$

lemma *lookup-monomial-If*:
 $\langle \text{lookup } (\text{monomial } v \ k) = (\lambda k'. \text{ if } k = k' \text{ then } v \text{ else } 0) \rangle$
 $\langle \text{proof} \rangle$

lemma *vars-mult-Var*:
 $\langle \text{vars } (\text{Var } x * p) = (\text{if } p = 0 \text{ then } \{\} \text{ else insert } x \ (\text{vars } p)) \rangle$ **for** $p :: \langle \text{int mpoly} \rangle$
 $\langle \text{proof} \rangle$

lemma *keys-mult-monomial*:
 $\langle \text{keys } (\text{monomial } (n :: \text{int}) \ k * \text{mapping-of } a) = (\text{if } n = 0 \text{ then } \{\} \text{ else } ((+) \ k) \ ' \text{keys } (\text{mapping-of } a)) \rangle$
 $\langle \text{proof} \rangle$

lemma *vars-mult-Const*:
 $\langle \text{vars } (\text{Const } n * a) = (\text{if } n = 0 \text{ then } \{\} \text{ else vars } a) \rangle$ **for** $a :: \langle \text{int mpoly} \rangle$
 $\langle \text{proof} \rangle$

lemma *coeff-minus*: $\text{coeff } p \ m - \text{coeff } q \ m = \text{coeff } (p - q) \ m$
 $\langle \text{proof} \rangle$

lemma *Const-1-eq-1*: $\langle \text{Const } (1 :: \text{int}) = (1 :: \text{int mpoly}) \rangle$
 $\langle \text{proof} \rangle$

lemma *[simp]*:
 $\langle \text{vars } (1 :: \text{int mpoly}) = \{\} \rangle$
 $\langle \text{proof} \rangle$

2.2 More Ideals

lemma
fixes $A :: \langle ((x \Rightarrow_0 \text{nat}) \Rightarrow_0 'a :: \text{comm-ring-1}) \text{ set} \rangle$
assumes $\langle p \in \text{ideal } A \rangle$
shows $\langle p * q \in \text{ideal } A \rangle$
 $\langle \text{proof} \rangle$

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'*:
 $\langle \text{More-Modules.ideal } (\text{insert } a \ S) = \{y. \exists x \ k. y = x + k * a \wedge x \in \text{More-Modules.ideal } S\} \rangle$

⟨proof⟩

lemma *ideal-mult-right-in*:

⟨ $a \in \text{ideal } A \implies a * b \in \text{More-Modules.ideal } A$ ⟩

⟨proof⟩

lemma *ideal-mult-right-in2*:

⟨ $a \in \text{ideal } A \implies b * a \in \text{More-Modules.ideal } A$ ⟩

⟨proof⟩

lemma [simp]: ⟨vars (Var x :: 'a :: {zero-neq-one} mpoly) = {x}⟩

⟨proof⟩

lemma *vars-minus-Var-subset*:

⟨vars (p' - Var x :: 'a :: {ab-group-add,one,zero-neq-one} mpoly) ⊆ V ⟹ vars p' ⊆ insert x V⟩

⟨proof⟩

lemma *vars-add-Var-subset*:

⟨vars (p' + Var x :: 'a :: {ab-group-add,one,zero-neq-one} mpoly) ⊆ V ⟹ vars p' ⊆ insert x V⟩

⟨proof⟩

lemma *coeff-monomila-in-varsD*:

⟨coeff p (monomial (Suc 0) x) ≠ 0 ⟹ x ∈ vars (p :: int mpoly)⟩

⟨proof⟩

lemma (in -) *coeff-MPoly-monomila[simp]*:

⟨Const (MPoly-Type.coeff (MPoly (monomial a m)) m) = Const a⟩

⟨proof⟩

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⟩

⟨proof⟩

lemma *pac-idealI1* [intro]:

⟨p ∈ A ⟹ p ∈ *pac-ideal* A⟩

⟨proof⟩

lemma *pac-idealI2* [intro]:

$\langle p \in \text{ideal } A \implies p \in \text{pac-ideal } A \rangle$
 $\langle \text{proof} \rangle$

lemma *pac-idealI3[intro]*:
 $\langle p \in \text{ideal } A \implies p * q \in \text{pac-ideal } A \rangle$
 $\langle \text{proof} \rangle$

lemma *pac-ideal-Xsq2-iff*:
 $\langle \text{Var } c \wedge 2 \in \text{pac-ideal } A \iff \text{Var } c \in \text{pac-ideal } A \rangle$
 $\langle \text{proof} \rangle$

lemma *diff-in-polynom-bool-pac-idealI*:
assumes *a1*: $p \in \text{pac-ideal } A$
assumes *a2*: $p - p' \in \text{More-Modules.ideal polynom-bool}$
shows $\langle p' \in \text{pac-ideal } A \rangle$
 $\langle \text{proof} \rangle$

lemma *diff-in-polynom-bool-pac-idealI2*:
assumes *a1*: $p \in A$
assumes *a2*: $p - p' \in \text{More-Modules.ideal polynom-bool}$
shows $\langle p' \in \text{pac-ideal } A \rangle$
 $\langle \text{proof} \rangle$

lemma *pac-ideal-alt-def*:
 $\langle \text{pac-ideal } A = \text{ideal } (A \cup \text{ideal polynom-bool}) \rangle$
 $\langle \text{proof} \rangle$

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_I 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_V 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_{V I} 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*:
 $\langle p \in \text{pac-ideal } (\text{set-mset } A) \implies \text{vars } p \subseteq C \implies p \in \text{restricted-ideal-to}_I \ C \ A \rangle$
 $\langle \text{proof} \rangle$

lemma *pac-ideal-insert-already-in*:
 $\langle pq \in \text{pac-ideal } (\text{set-mset } A) \implies \text{pac-ideal } (\text{insert } pq \ (\text{set-mset } A)) = \text{pac-ideal } (\text{set-mset } A) \rangle$
 $\langle \text{proof} \rangle$

lemma *pac-ideal-add*:
 $\langle p \in \# A \implies q \in \# A \implies p + q \in \text{pac-ideal } (\text{set-mset } A) \rangle$
 $\langle \text{proof} \rangle$

lemma *pac-ideal-mult*:
 $\langle p \in \# A \implies p * q \in \text{pac-ideal } (\text{set-mset } A) \rangle$

$\langle proof \rangle$

lemma *pac-ideal-mono*:

$\langle A \subseteq B \implies \text{pac-ideal } A \subseteq \text{pac-ideal } B \rangle$

$\langle proof \rangle$

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

inductive *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 *PAC-Format-induct-split* =

PAC-Format.induct[split-format(complete), of V A V' A' for V A V' A']

lemma *PAC-Format-induct[consumes 1, 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
 $\langle P \ \mathcal{V} \ A \ \mathcal{V}' \ A' \rangle$
 $\langle \text{proof} \rangle$

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

lemma extensions-are-safe-uminus:

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

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

lemma vars-subst-in-left-only:

$\langle x \notin \text{vars } p \implies x \in \text{vars } (p - \text{Var } x) \rangle$ **for** $p :: \langle \text{int mpoly} \rangle$
 $\langle \text{proof} \rangle$

lemma vars-subst-in-left-only-diff-iff:

$\langle x \notin \text{vars } p \implies \text{vars } (p - \text{Var } x) = \text{insert } x \ (\text{vars } p) \rangle$ **for** $p :: \langle \text{int mpoly} \rangle$
 $\langle \text{proof} \rangle$

lemma vars-subst-in-left-only-iff:

$\langle x \notin \text{vars } p \implies \text{vars } (p + \text{Var } x) = \text{insert } x \ (\text{vars } p) \rangle$ **for** $p :: \langle \text{int mpoly} \rangle$
 $\langle \text{proof} \rangle$

lemma coeff-add-right-notin:

$\langle x \notin \text{vars } p \implies \text{MPoly-Type.coeff } (\text{Var } x - p) \ (\text{monomial } (\text{Suc } 0) \ x) = 1 \rangle$

⟨proof⟩

lemma *coeff-add-left-notin*:

⟨ $x \notin \text{vars } p \implies \text{MPoly-Type.coeff } (p - \text{Var } x) (\text{monomial } (\text{Suc } 0) x) = -1 \rangle \text{ for } p :: \langle \text{int mpoly} \rangle$
 ⟨proof⟩

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

$\text{More-Modules.ideal } (\text{insert } r \ (A \cup \text{polynom-bool})) = \text{More-Modules.ideal } (\text{insert } s \ (A \cup \text{polynom-bool})) \rangle$
 ⟨proof⟩

lemma *PAC-Format-subset-ideal*:

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

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

lemma *restricted-ideal-to-restricted-ideal-to_ID*:

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

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

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

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* :: ⟨ $'a \text{ fset} \Rightarrow 'a \text{ multiset}$ ⟩ **where**

⟨ $\text{mset-fset } N \equiv \text{mset-set } (\text{fset } N) \rangle$

definition *fset-mset* :: ⟨ $'a \text{ multiset} \Rightarrow 'a \text{ fset}$ ⟩ **where**

⟨ $\text{fset-mset } N \equiv \text{Abs-fset } (\text{set-mset } N) \rangle$

lemma *fset-mset-mset-fset*: ⟨ $\text{fset-mset } (\text{mset-fset } N) = N \rangle$

⟨proof⟩

lemma *mset-fset-fset-mset[simp]*:

⟨ $\text{mset-fset } (\text{fset-mset } N) = \text{remdups-mset } N \rangle$

⟨proof⟩

lemma *in-mset-fset-fmember[simp]*: ⟨ $x \in\# \text{ mset-fset } N \longleftrightarrow x \in | N \rangle$

⟨proof⟩

lemma *in-fset-mset-mset[simp]*: ⟨ $x \in | \text{ fset-mset } N \longleftrightarrow x \in\# N \rangle$

⟨proof⟩

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

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

lemma *dom-m-fmdrop[simp]*: $\langle \text{dom-m } (\text{fmdrop } C \ N) = \text{remove1-mset } C \ (\text{dom-m } N) \rangle$

⟨proof⟩

lemma *dom-m-fmdrop-All*: $\langle \text{dom-m } (\text{fmdrop } C \ N) = \text{removeAll-mset } C \ (\text{dom-m } N) \rangle$

⟨proof⟩

lemma *dom-m-fmupd[simp]*: $\langle \text{dom-m } (\text{fmupd } k \ C \ N) = \text{add-mset } k \ (\text{remove1-mset } k \ (\text{dom-m } N)) \rangle$

⟨proof⟩

lemma *distinct-mset-dom*: $\langle \text{distinct-mset } (\text{dom-m } N) \rangle$

⟨proof⟩

lemma *in-dom-m-lookup-iff*: $\langle C \in \# \text{ dom-m } N' \longleftrightarrow \text{fmlookup } N' \ C \neq \text{None} \rangle$

⟨proof⟩

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

⟨proof⟩

lemma *fmupd-same[simp]*:

$$\langle x1 \in \# \text{ dom-m } x1aa \implies \text{fmupd } x1 \ (\text{the } (\text{fmlookup } x1aa \ x1)) \ x1aa = x1aa \rangle$$

⟨proof⟩

lemma *ran-m-fmempty[simp]*: $\langle \text{ran-m } \text{fmempty} = \{ \# \} \rangle$ and

$$\text{dom-m-fmempty[simp]: } \langle \text{dom-m } \text{fmempty} = \{ \# \} \rangle$$

⟨proof⟩

lemma *fmrestrict-set-fmupd*:

$$\langle a \in xs \implies \text{fmrestrict-set } xs \ (\text{fmupd } a \ C \ N) = \text{fmupd } a \ C \ (\text{fmrestrict-set } xs \ N) \rangle$$

$$\langle a \notin xs \implies \text{fmrestrict-set } xs \ (\text{fmupd } a \ C \ N) = \text{fmrestrict-set } xs \ N \rangle$$

⟨proof⟩

lemma *fset-fmdom-fmrestrict-set*:

$$\langle \text{fset } (\text{fmdom } (\text{fmrestrict-set } xs \ N)) = \text{fset } (\text{fmdom } N) \cap xs \rangle$$

⟨proof⟩

lemma *dom-m-fmrestrict-set*: $\langle \text{dom-m } (\text{fmrestrict-set } (\text{set } xs) \ N) = \text{mset } xs \cap \# \text{ dom-m } N \rangle$

⟨proof⟩

lemma *dom-m-fmrestrict-set'*: $\langle \text{dom-m } (\text{fmrestrict-set } xs \ N) = \text{mset-set } (xs \cap \text{set-mset } (\text{dom-m } N)) \rangle$

⟨proof⟩

lemma *indom-mI*: $\langle \text{fmlookup } m \ x = \text{Some } y \implies x \in \# \text{ dom-}m \ m \rangle$
 $\langle \text{proof} \rangle$

lemma *fmupd-fmdrop-id*:
assumes $\langle k \in \text{fmdom } N' \rangle$
shows $\langle \text{fmupd } k \ (\text{the } (\text{fmlookup } N' \ k)) \ (\text{fmdrop } k \ N') = N' \rangle$
 $\langle \text{proof} \rangle$

lemma *fm-member-split*: $\langle k \in \text{fmdom } N' \implies \exists N'' \ v. N' = \text{fmupd } k \ v \ N'' \wedge \text{the } (\text{fmlookup } N' \ k) = v \wedge k \notin \text{fmdom } N'' \rangle$
 $\langle \text{proof} \rangle$

lemma $\langle \text{fmdrop } k \ (\text{fmupd } k \ va \ N'') = \text{fmdrop } k \ N'' \rangle$
 $\langle \text{proof} \rangle$

lemma *fmmap-ext-fmdom*:
 $\langle (\text{fmdom } N = \text{fmdom } N') \implies (\bigwedge x. x \in \text{fmdom } N \implies \text{fmlookup } N \ x = \text{fmlookup } N' \ x) \implies N = N' \rangle$
 $\langle \text{proof} \rangle$

lemma *fmrestrict-set-insert-in*:
 $\langle xa \in \text{fset } (\text{fmdom } N) \implies \text{fmrestrict-set } (\text{insert } xa \ l1) \ N = \text{fmupd } xa \ (\text{the } (\text{fmlookup } N \ xa)) \ (\text{fmrestrict-set } l1 \ N) \rangle$
 $\langle \text{proof} \rangle$

lemma *fmrestrict-set-insert-notin*:
 $\langle xa \notin \text{fset } (\text{fmdom } N) \implies \text{fmrestrict-set } (\text{insert } xa \ l1) \ N = \text{fmrestrict-set } l1 \ N \rangle$
 $\langle \text{proof} \rangle$

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

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

lemma *fmlookup-restrict-set-id*: $\langle \text{fset } (\text{fmdom } N) \subseteq A \implies \text{fmrestrict-set } A \ N = N \rangle$
 $\langle \text{proof} \rangle$

lemma *fmlookup-restrict-set-id'*: $\langle \text{set-mset } (\text{dom-}m \ N) \subseteq A \implies \text{fmrestrict-set } A \ N = N \rangle$
 $\langle \text{proof} \rangle$

lemma *ran-m-mapsto-upd*:
assumes $NC: \langle C \in \# \text{ dom-}m \ N \rangle$
shows $\langle \text{ran-}m \ (\text{fmupd } C \ C' \ N) = \text{add-mset } C' \ (\text{remove1-mset } (\text{the } (\text{fmlookup } N \ C)) \ (\text{ran-}m \ N)) \rangle$
 $\langle \text{proof} \rangle$

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

assumes NC : $\langle C \notin \# \text{ dom-}m \ N \rangle$
shows $\langle \text{ran-}m \ (\text{fmupd } C \ C' \ N) = \text{add-mset } C' \ (\text{ran-}m \ N) \rangle$
 $\langle \text{proof} \rangle$

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

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

lemma *filter-mset-cong2*:

$\langle \bigwedge x. x \in \# M \implies f \ x = g \ x \implies M = N \implies \text{filter-mset } f \ M = \text{filter-mset } g \ N \rangle$
 $\langle \text{proof} \rangle$

lemma *ran-m-fmdrop*:

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

lemma *ran-m-fmdrop-notin*:

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

lemma *ran-m-fmdrop-If*:

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

lemma *dom-m-empty-iff[iff]*:

$\langle \text{dom-}m \ NU = \{ \# \} \longleftrightarrow NU = \text{fmempty} \rangle$
 $\langle \text{proof} \rangle$

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**

$[to\text{-}relAPP]$:

$\text{fmap-rel } K \ V \equiv \{ (m1, m2). \}$

$(\forall i \ j. i \in | \text{fmdom } m2 \longrightarrow (j, i) \in K \longrightarrow (\text{the } (\text{fmlookup } m1 \ j), \text{the } (\text{fmlookup } m2 \ i)) \in V) \wedge$
 $\text{fset } (\text{fmdom } m1) \subseteq \text{Domain } K \wedge \text{fset } (\text{fmdom } m2) \subseteq \text{Range } K \wedge$
 $(\forall i \ j. (i, j) \in K \longrightarrow j \in | \text{fmdom } m2 \longleftrightarrow i \in | \text{fmdom } m1) \}$

lemma *fmap-rel-alt-def*:

$\langle (K, V) \text{fmap-rel} \equiv$

$\{ (m1, m2). \}$

$(\forall i \ j. i \in \# \text{ dom-}m \ m2 \longrightarrow$

$(j, i) \in K \longrightarrow (\text{the } (\text{fmlookup } m1 \ j), \text{the } (\text{fmlookup } m2 \ i)) \in V) \wedge$

$\text{fset } (\text{fmdom } m1) \subseteq \text{Domain } K \wedge$

$\text{fset } (\text{fmdom } m2) \subseteq \text{Range } K \wedge$

$(\forall i j. (i, j) \in K \longrightarrow (j \in \# \text{ dom-}m \ m2) = (i \in \# \text{ dom-}m \ m1)))\}$
 \rangle
 $\langle \text{proof} \rangle$

lemma *fmap-rel-empty1-simp*[simp]:
 $(\text{fmempty}, m) \in \langle K, V \rangle \text{fmap-rel} \longleftrightarrow m = \text{fmempty}$
 $\langle \text{proof} \rangle$

lemma *fmap-rel-empty2-simp*[simp]:
 $(m, \text{fmempty}) \in \langle K, V \rangle \text{fmap-rel} \longleftrightarrow m = \text{fmempty}$
 $\langle \text{proof} \rangle$

sempref-decl-intf $(\text{'k}, \text{'v})$ *f-map* **is** $(\text{'k}, \text{'v})$ *fmap*

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

5.1 Operations

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

sempref-decl-op *fmap-is-empty*: $(=) \text{fmempty} :: \langle K, V \rangle \text{fmap-rel} \rightarrow \text{bool-rel}$
 $\langle \text{proof} \rangle$

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})
 $\langle \text{proof} \rangle$

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})
 $\langle \text{proof} \rangle$

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})
 $\langle \text{proof} \rangle$

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})
 $\langle \text{proof} \rangle$

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

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

$a \in \# \text{ dom-}m \text{ aa}$
 $\langle \text{proof} \rangle$

lemma *fmap-rel-fmlookup-rel*:

$\langle (a, a') \in K \implies (aa, a'a) \in \langle K, V \rangle \text{fmap-rel} \implies$
 $(\text{fmlookup } aa \text{ } a, \text{fmlookup } a'a \text{ } a') \in \langle V \rangle \text{option-rel} \rangle$
 $\langle \text{proof} \rangle$

sempref-decl-op *fmap-lookup*: $\text{fmlookup} :: \langle K, V \rangle \text{fmap-rel} \rightarrow K \rightarrow \langle V \rangle \text{option-rel}$
 $\langle \text{proof} \rangle$

lemma *in-fdom-alt*: $k \in \# \text{ dom-}m \text{ } m \longleftrightarrow \neg \text{is-None } (\text{fmlookup } m \text{ } k)$
 $\langle \text{proof} \rangle$

sempref-decl-op *fmap-contains-key*: $\lambda k \text{ } m. k \in \# \text{ dom-}m \text{ } m :: K \rightarrow \langle K, V \rangle \text{fmap-rel} \rightarrow \text{bool-rel}$
 $\langle \text{proof} \rangle$

5.2 Patterns

lemma *pat-fmap-empty*[*pat-rules*]: $\text{fmempty} \equiv \text{op-fmap-empty}$ $\langle \text{proof} \rangle$

lemma *pat-map-is-empty*[*pat-rules*]:

$(=) \text{ } \$m\$ \text{fmempty} \equiv \text{op-fmap-is-empty} \m
 $(=) \text{ } \$ \text{fmempty} \$m \equiv \text{op-fmap-is-empty} \m
 $(=) \text{ } \$(\text{dom-}m \$m) \$\{ \# \} \equiv \text{op-fmap-is-empty} \m
 $(=) \text{ } \$\{ \# \} \$(\text{dom-}m \$m) \equiv \text{op-fmap-is-empty} \m
 $\langle \text{proof} \rangle$

lemma *op-map-contains-key*[*pat-rules*]:

$(\in \#) \$ k \$ (\text{dom-}m \$m) \equiv \text{op-fmap-contains-key} \$'k \$'m$
 $\langle \text{proof} \rangle$

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**

$\langle \text{map-fmap-rel} = \text{br map-of-fmap } (\lambda a. \text{finite } (\text{dom } a)) \rangle$

lemma *fmdrop-set-None*:

$\langle (\text{op-map-delete}, \text{fmdrop}) \in \text{Id} \rightarrow \text{map-fmap-rel} \rightarrow \text{map-fmap-rel} \rangle$
 $\langle \text{proof} \rangle$

lemma *map-upd-fmupd*:

$\langle (\text{op-map-update}, \text{fmupd}) \in \text{Id} \rightarrow \text{Id} \rightarrow \text{map-fmap-rel} \rightarrow \text{map-fmap-rel} \rangle$
 $\langle \text{proof} \rangle$

Technically *op-map-lookup* has the arguments in the wrong direction.

definition *fmlookup'* **where**

[*simp*]: $\langle \text{fmlookup}' A k = \text{fmlookup } k A \rangle$

lemma [*def-pat-rules*]:

$\langle ((\in \#) \$ k \$ (\text{dom-}m \$A)) \equiv \text{Not} \$ (\text{is-None} \$ (\text{fmlookup}' k \$A)) \rangle$

$\langle \text{proof} \rangle$

lemma *op-map-lookup-fmlookup*:

$\langle (op\text{-}map\text{-}lookup, fmlookup') \in Id \rightarrow map\text{-}fmap\text{-}rel \rightarrow \langle Id \rangle option\text{-}rel \rangle$
 $\langle \text{proof} \rangle$

abbreviation *hm-fmap-assn* **where**

$\langle hm\text{-}fmap\text{-}assn\ K\ V \equiv hr\text{-}comp\ (hm.assn\ K\ V)\ map\text{-}fmap\text{-}rel \rangle$

lemmas *fmap-delete-hnr* [*sepref-fr-rules*] =
 $hm.delete\text{-}hnr[FCOMP\ fmdrop\text{-}set\text{-}None]$

lemmas *fmap-update-hnr* [*sepref-fr-rules*] =
 $hm.update\text{-}hnr[FCOMP\ map\text{-}upd\text{-}fmupd]$

lemmas *fmap-lookup-hnr* [*sepref-fr-rules*] =
 $hm.lookup\text{-}hnr[FCOMP\ op\text{-}map\text{-}lookup\text{-}fmlookup]$

lemma *fmempty-empty*:

$\langle (uncurry0\ (RETURN\ op\text{-}map\text{-}empty),\ uncurry0\ (RETURN\ fmempty)) \in unit\text{-}rel \rightarrow_f \langle map\text{-}fmap\text{-}rel \rangle nres\text{-}rel \rangle$
 $\langle \text{proof} \rangle$

lemmas [*sepref-fr-rules*] =

$hm.empty\text{-}hnr[FCOMP\ fmempty\text{-}empty,\ unfolded\ op\text{-}fmap\text{-}empty\text{-}def[symmetric]]$

abbreviation *iam-fmap-assn* **where**

$\langle iam\text{-}fmap\text{-}assn\ K\ V \equiv hr\text{-}comp\ (iam.assn\ K\ V)\ map\text{-}fmap\text{-}rel \rangle$

lemmas *iam-fmap-delete-hnr* [*sepref-fr-rules*] =
 $iam.delete\text{-}hnr[FCOMP\ fmdrop\text{-}set\text{-}None]$

lemmas *iam-ffmap-update-hnr* [*sepref-fr-rules*] =
 $iam.update\text{-}hnr[FCOMP\ map\text{-}upd\text{-}fmupd]$

lemmas *iam-ffmap-lookup-hnr* [*sepref-fr-rules*] =
 $iam.lookup\text{-}hnr[FCOMP\ op\text{-}map\text{-}lookup\text{-}fmlookup]$

definition *op-iam-fmap-empty* **where**

$\langle op\text{-}iam\text{-}fmap\text{-}empty = fmempty \rangle$

lemma *iam-fmempty-empty*:

$\langle (uncurry0\ (RETURN\ op\text{-}map\text{-}empty),\ uncurry0\ (RETURN\ op\text{-}iam\text{-}fmap\text{-}empty)) \in unit\text{-}rel \rightarrow_f \langle map\text{-}fmap\text{-}rel \rangle nres\text{-}rel \rangle$
 $\langle \text{proof} \rangle$

lemmas [*sepref-fr-rules*] =

$iam.empty\text{-}hnr[FCOMP\ fmempty\text{-}empty,\ unfolded\ op\text{-}iam\text{-}fmap\text{-}empty\text{-}def[symmetric]]$

definition *upper-bound-on-dom* **where**

$\langle upper\text{-}bound\text{-}on\text{-}dom\ A = SPEC(\lambda n. \forall i \in \#(dom\text{-}m\ A). i < n) \rangle$

lemma *[sepreffr-rules]*:
 $\langle ((\text{Array.len}), \text{upper-bound-on-dom}) \in (\text{iam-fmap-assn nat-assn } V)^k \rightarrow_a \text{nat-assn} \rangle$
 $\langle \text{proof} \rangle$

lemma *fmap-rel-nat-rel-dom-m[simp]*:
 $\langle (A, B) \in \langle \text{nat-rel}, R \rangle \text{fmap-rel} \implies \text{dom-m } A = \text{dom-m } B \rangle$
 $\langle \text{proof} \rangle$

lemma *ref-two-step'*:
 $\langle A \leq B \implies \Downarrow R A \leq \Downarrow R B \rangle$
 $\langle \text{proof} \rangle$

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*:
 $\langle \text{is-success } a \longleftrightarrow a = \text{SUCCESS} \rangle$
 $\langle \text{proof} \rangle$

datatype (*'a*, *'b*, *'lbs*) *pac-step* =
Add (*pac-src1*: *'lbs*) (*pac-src2*: *'lbs*) (*new-id*: *'lbs*) (*pac-res*: *'a*) |
Mult (*pac-src1*: *'lbs*) (*pac-mult*: *'a*) (*new-id*: *'lbs*) (*pac-res*: *'a*) |
Extension (*new-id*: *'lbs*) (*new-var*: *'b*) (*pac-res*: *'a*) |
Del (*pac-src1*: *'lbs*)

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

$\langle \text{PAC-checker-specification spec } A = \text{SPEC}(\lambda(b, \mathcal{V}, B).$
 $(\neg \text{is-failed } b \longrightarrow \text{restricted-ideal-to}_I (\bigcup (\text{vars } \text{'set-mset } A) \cup \text{vars spec}) B \subseteq \text{restricted-ideal-to}_I$
 $(\bigcup (\text{vars } \text{'set-mset } A) \cup \text{vars spec}) A) \wedge$
 $(\text{is-found } b \longrightarrow \text{spec} \in \text{pac-ideal } (\text{set-mset } A))) \rangle$

definition *PAC-checker-specification-spec*

$\vdash \langle \text{int-poly} \Rightarrow \text{pac-state} \Rightarrow (\text{status} \times \text{pac-state}) \Rightarrow \text{bool} \rangle$
where
 $\langle \text{PAC-checker-specification-spec spec} = (\lambda(\mathcal{V}, A) (b, B). (\neg \text{is-failed } b \longrightarrow \bigcup (\text{vars } \text{'set-mset } A) \subseteq \mathcal{V}) \wedge$
 $(\text{is-success } b \longrightarrow \text{PAC-Format}^{**}(\mathcal{V}, A) B) \wedge$
 $(\text{is-found } b \longrightarrow \text{PAC-Format}^{**}(\mathcal{V}, A) B \wedge \text{spec} \in \text{pac-ideal } (\text{set-mset } A))) \rangle$

abbreviation *PAC-checker-specification2*

$\vdash \langle \text{int-poly} \Rightarrow (\text{nat set} \times \text{int-poly multiset}) \Rightarrow (\text{status} \times (\text{nat set} \times \text{int-poly multiset})) \text{ nres} \rangle$
where
 $\langle \text{PAC-checker-specification2 spec } A \equiv \text{SPEC}(\text{PAC-checker-specification-spec spec } A) \rangle$

definition *PAC-checker-specification-step-spec*

$\vdash \langle \text{pac-state} \Rightarrow \text{int-poly} \Rightarrow \text{pac-state} \Rightarrow (\text{status} \times \text{pac-state}) \Rightarrow \text{bool} \rangle$
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*

$\vdash \langle \text{pac-state} \Rightarrow \text{int-poly} \Rightarrow \text{pac-state} \Rightarrow (\text{status} \times \text{pac-state}) \text{ nres} \rangle$
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* $\vdash \langle \cdot \rangle$ **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*:

$\langle \text{normalize-poly-spec } p = \text{SPEC } (\lambda r. r - p \in \text{ideal polynom-bool} \wedge \text{vars } r \subseteq \text{vars } p) \rangle$
 $\langle \text{proof} \rangle$

definition *mult-poly-spec* $\vdash \langle \text{int mpoly} \Rightarrow \text{int mpoly} \Rightarrow \text{int mpoly nres} \rangle$ **where**

$\langle \text{mult-poly-spec } p \ q = \text{SPEC } (\lambda r. p * q - r \in \text{ideal polynom-bool}) \rangle$

definition *check-add* $\vdash \langle (\text{nat}, \text{int mpoly}) \text{ fmap} \Rightarrow \text{nat set} \Rightarrow \text{nat} \Rightarrow \text{nat} \Rightarrow \text{nat} \Rightarrow \text{int mpoly} \Rightarrow \text{bool nres} \rangle$ **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* $\vdash \langle (\text{nat}, \text{int mpoly}) \text{ fmap} \Rightarrow \text{nat set} \Rightarrow \text{nat} \Rightarrow \text{int mpoly} \Rightarrow \text{nat} \Rightarrow \text{int mpoly} \Rightarrow \text{bool nres} \rangle$ **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* $\vdash \langle (\text{nat}, \text{int mpoly}) \text{ fmap} \Rightarrow \text{nat set} \Rightarrow \text{nat} \Rightarrow \text{nat} \Rightarrow \text{int mpoly} \Rightarrow (\text{bool nres}) \rangle$ **where**

$\langle \text{check-extension } A \ \mathcal{V} \ i \ v \ p =$

$SPEC(\lambda b. b \longrightarrow (i \notin \# \text{ dom-}m \ A \wedge$
 $(v \notin \mathcal{V} \wedge$
 $(p + \text{Var } v)^2 - (p + \text{Var } v) \in \text{ideal polynom-bool} \wedge$
 $\text{vars } (p + \text{Var } v) \subseteq \mathcal{V})))$

fun *merge-status* **where**

$\langle \text{merge-status } \text{FAILED} \rangle = \text{FAILED} \mid$
 $\langle \text{merge-status } - \text{ FAILED} \rangle = \text{FAILED} \mid$
 $\langle \text{merge-status } \text{FOUND} \rangle = \text{FOUND} \mid$
 $\langle \text{merge-status } - \text{ FOUND} \rangle = \text{FOUND} \mid$
 $\langle \text{merge-status } - \rangle = \text{SUCCESS}$

type-synonym *fpac-step* = $\langle \text{nat set} \times (\text{nat}, \text{int-poly}) \text{ fmap} \rangle$

definition *check-del* :: $\langle (\text{nat}, \text{int mpoly}) \text{ fmap} \Rightarrow \text{nat} \Rightarrow \text{bool nres} \rangle$ **where**

$\langle \text{check-del } A \ p =$
 $SPEC(\lambda b. b \longrightarrow \text{True}) \rangle$

6.2 Algorithm

definition *PAC-checker-step*

$:: \langle \text{int-poly} \Rightarrow (\text{status} \times \text{fpac-step}) \Rightarrow (\text{int-poly}, \text{nat}, \text{nat}) \text{ pac-step} \Rightarrow$
 $(\text{status} \times \text{fpac-step}) \text{ nres} \rangle$

where

$\langle \text{PAC-checker-step} = (\lambda \text{spec } (\text{stat}, (\mathcal{V}, A)) \text{ st. case st of}$
 $\text{Add } - - - \Rightarrow$
 $\text{do } \{$
 $\quad r \leftarrow \text{normalize-poly-spec } (\text{pac-res st});$
 $\quad \text{eq} \leftarrow \text{check-add } A \ \mathcal{V} \ (\text{pac-src1 st}) \ (\text{pac-src2 st}) \ (\text{new-id st}) \ r;$
 $\quad \text{st}' \leftarrow SPEC(\lambda \text{st}'. (\neg \text{is-failed st}' \wedge \text{is-found st}' \longrightarrow r - \text{spec} \in \text{ideal polynom-bool}));$
 $\quad \text{if eq}$
 $\quad \text{then RETURN } (\text{merge-status stat st}',$
 $\quad \quad \mathcal{V}, \text{fmupd } (\text{new-id st}) \ r \ A)$
 $\quad \text{else RETURN } (\text{FAILED}, (\mathcal{V}, A))$
 $\}$
 $\mid \text{Del } - \Rightarrow$
 $\text{do } \{$
 $\quad \text{eq} \leftarrow \text{check-del } A \ (\text{pac-src1 st});$
 $\quad \text{if eq}$
 $\quad \text{then RETURN } (\text{stat}, (\mathcal{V}, \text{fmdrop } (\text{pac-src1 st}) \ A))$
 $\quad \text{else RETURN } (\text{FAILED}, (\mathcal{V}, A))$
 $\}$
 $\mid \text{Mult } - - - \Rightarrow$
 $\text{do } \{$
 $\quad r \leftarrow \text{normalize-poly-spec } (\text{pac-res st});$
 $\quad q \leftarrow \text{normalize-poly-spec } (\text{pac-mult st});$
 $\quad \text{eq} \leftarrow \text{check-mult } A \ \mathcal{V} \ (\text{pac-src1 st}) \ q \ (\text{new-id st}) \ r;$
 $\quad \text{st}' \leftarrow SPEC(\lambda \text{st}'. (\neg \text{is-failed st}' \wedge \text{is-found st}' \longrightarrow r - \text{spec} \in \text{ideal polynom-bool}));$
 $\quad \text{if eq}$
 $\quad \text{then RETURN } (\text{merge-status stat st}',$
 $\quad \quad \mathcal{V}, \text{fmupd } (\text{new-id st}) \ r \ A)$
 $\quad \text{else RETURN } (\text{FAILED}, (\mathcal{V}, A))$
 $\}$
 $\mid \text{Extension } - - - \Rightarrow$
 $\text{do } \{$
 $\quad r \leftarrow \text{normalize-poly-spec } (\text{pac-res st} - \text{Var } (\text{new-var st}));$

```

    (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* :: $\langle (nat, int\ mpoly) fmap \times - \rangle set$ **where**
 $\langle polys-rel = \{(A, B). B = (ran-m\ A)\} \rangle$

definition *polys-rel-full* :: $\langle (nat\ set \times (nat, int\ mpoly) fmap) \times - \rangle set$ **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)
  <proof>

```

lemma *polys-rel-in-dom-inD*:

```

  (A, B) ∈ polys-rel ⇒
  x12 ∈ #dom-m A ⇒
  the (fmlookup A x12) ∈ #B
  <proof>

```

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** (V, B) (V, remove1-mset (the (fmlookup A x12)) (add-mset r B))
  (r - x14 ∈ More-Modules.ideal polynom-bool ⇒
  (A, B) ∈ polys-rel ⇒
  the (fmlookup A x11) + the (fmlookup A x12) - r ∈ More-Modules.ideal polynom-bool ⇒
  x11 ∈ #dom-m A ⇒
  x12 ∈ #dom-m A ⇒
  vars r ⊆ V ⇒
  PAC-Format** (V, B) (V, add-mset r B))
  (r - x14 ∈ More-Modules.ideal polynom-bool ⇒
  (A, B) ∈ polys-rel ⇒
  x11 ∈ #dom-m A ⇒
  x12 ∈ #dom-m A ⇒

```

$\langle \text{the (fmlookup } A \ x11) + \text{the (fmlookup } A \ x12) - r \in \text{More-Modules.ideal polynom-bool} \implies$
 $\text{vars } r \subseteq \mathcal{V} \implies$
 $x11 \neq x12 \implies$
 $\text{PAC-Format}^{**} (\mathcal{V}, B)$
 $(\mathcal{V}, \text{add-mset } r \ B - \{\# \text{the (fmlookup } A \ x11), \text{the (fmlookup } A \ x12) \# \}) \rangle$
 $\langle (A, B) \in \text{polys-rel} \implies$
 $r - x34 \in \text{More-Modules.ideal polynom-bool} \implies$
 $x11 \in \# \text{ dom-m } A \implies$
 $\text{the (fmlookup } A \ x11) * x32 - r \in \text{More-Modules.ideal polynom-bool} \implies$
 $\text{vars } x32 \subseteq \mathcal{V} \implies$
 $\text{vars } r \subseteq \mathcal{V} \implies$
 $\text{PAC-Format}^{**} (\mathcal{V}, B) (\mathcal{V}, \text{add-mset } r \ B) \rangle$
 $\langle (A, B) \in \text{polys-rel} \implies$
 $r - x34 \in \text{More-Modules.ideal polynom-bool} \implies$
 $x11 \in \# \text{ dom-m } A \implies$
 $\text{the (fmlookup } A \ x11) * x32 - r \in \text{More-Modules.ideal polynom-bool} \implies$
 $\text{vars } x32 \subseteq \mathcal{V} \implies$
 $\text{vars } r \subseteq \mathcal{V} \implies$
 $\text{PAC-Format}^{**} (\mathcal{V}, B) (\mathcal{V}, \text{remove1-mset (the (fmlookup } A \ x11)) (add-mset } r \ B)) \rangle$
 $\langle (A, B) \in \text{polys-rel} \implies$
 $x12 \in \# \text{ dom-m } A \implies$
 $\text{PAC-Format}^{**} (\mathcal{V}, B) (\mathcal{V}, \text{remove1-mset (the (fmlookup } A \ x12)) } B) \rangle$
 $\langle (A, B) \in \text{polys-rel} \implies$
 $(p' + \text{Var } x)^2 - (p' + \text{Var } x) \in \text{ideal polynom-bool} \implies$
 $x \notin \mathcal{V} \implies$
 $x \notin \text{vars}(p' + \text{Var } x) \implies$
 $\text{vars}(p' + \text{Var } x) \subseteq \mathcal{V} \implies$
 $\text{PAC-Format}^{**} (\mathcal{V}, B)$
 $(\text{insert } x \ \mathcal{V}, \text{add-mset } p' \ B) \rangle$
 $\langle \text{proof} \rangle$

abbreviation $\text{status-rel} :: \langle (\text{status} \times \text{status}) \text{ set} \rangle$ **where**
 $\langle \text{status-rel} \equiv \text{Id} \rangle$

lemma $\text{is-merge-status}[simp]$:
 $\langle \text{is-failed (merge-status } a \ st') \longleftrightarrow \text{is-failed } a \vee \text{is-failed } st' \rangle$
 $\langle \text{is-found (merge-status } a \ st') \longleftrightarrow \neg \text{is-failed } a \wedge \neg \text{is-failed } st' \wedge (\text{is-found } a \vee \text{is-found } st') \rangle$
 $\langle \text{is-success (merge-status } a \ st') \longleftrightarrow (\text{is-success } a \wedge \text{is-success } st') \rangle$
 $\langle \text{proof} \rangle$

lemma $\text{status-rel-merge-status}$:
 $\langle (\text{merge-status } a \ b, \text{SUCCESS}) \notin \text{status-rel} \longleftrightarrow$
 $(a = \text{FAILED}) \vee (b = \text{FAILED}) \vee$
 $a = \text{FOUND} \vee (b = \text{FOUND}) \rangle$
 $\langle \text{proof} \rangle$

lemma Ex-status-iff :
 $\langle (\exists a. P \ a) \longleftrightarrow P \ \text{SUCCESS} \vee P \ \text{FOUND} \vee (P \ (\text{FAILED})) \rangle$
 $\langle \text{proof} \rangle$

lemma is-failed-alt-def :
 $\langle \text{is-failed } st' \longleftrightarrow \neg \text{is-success } st' \wedge \neg \text{is-found } st' \rangle$
 $\langle \text{proof} \rangle$

lemma *merge-status-eq-iff*[simp]:

$\langle \text{merge-status } a \text{ SUCCESS} = \text{SUCCESS} \longleftrightarrow a = \text{SUCCESS} \rangle$
 $\langle \text{merge-status } a \text{ SUCCESS} = \text{FOUND} \longleftrightarrow a = \text{FOUND} \rangle$
 $\langle \text{merge-status } \text{SUCCESS } a = \text{SUCCESS} \longleftrightarrow a = \text{SUCCESS} \rangle$
 $\langle \text{merge-status } \text{SUCCESS } a = \text{FOUND} \longleftrightarrow a = \text{FOUND} \rangle$
 $\langle \text{merge-status } \text{SUCCESS } a = \text{FAILED} \longleftrightarrow a = \text{FAILED} \rangle$
 $\langle \text{merge-status } a \text{ SUCCESS} = \text{FAILED} \longleftrightarrow a = \text{FAILED} \rangle$
 $\langle \text{merge-status } \text{FOUND } a = \text{FAILED} \longleftrightarrow a = \text{FAILED} \rangle$
 $\langle \text{merge-status } a \text{ FOUND} = \text{FAILED} \longleftrightarrow a = \text{FAILED} \rangle$
 $\langle \text{merge-status } a \text{ FOUND} = \text{SUCCESS} \longleftrightarrow \text{False} \rangle$
 $\langle \text{merge-status } a \text{ } b = \text{FOUND} \longleftrightarrow (a = \text{FOUND} \vee b = \text{FOUND}) \wedge (a \neq \text{FAILED} \wedge b \neq \text{FAILED}) \rangle$
 $\langle \text{proof} \rangle$

lemma *fmdrop-irrelevant*: $\langle x11 \notin \# \text{ dom-m } A \implies \text{fmdrop } x11 \text{ } A = A \rangle$

$\langle \text{proof} \rangle$

lemma *PAC-checker-step-PAC-checker-specification2*:

fixes $a :: \langle \text{status} \rangle$

assumes $AB: \langle (\mathcal{V}, A), (\mathcal{V}_B, B) \rangle \in \text{polys-rel-full} \rangle$ **and**

$\langle \neg \text{is-failed } a \rangle$ **and**

[simp,intro]: $\langle a = \text{FOUND} \implies \text{spec} \in \text{pac-ideal } (\text{set-mset } A_0) \rangle$ **and**

$A_0B: \langle \text{PAC-Format}^{**} (\mathcal{V}_0, A_0) (\mathcal{V}, B) \rangle$ **and**

$\text{spec}_0: \langle \text{vars spec} \subseteq \mathcal{V}_0 \rangle$ **and**

$\text{vars-}A_0: \langle \bigcup (\text{vars } \text{' set-mset } A_0) \subseteq \mathcal{V}_0 \rangle$

shows $\langle \text{PAC-checker-step spec } (a, (\mathcal{V}, A)) \text{ } st \leq \Downarrow (\text{status-rel } \times_r \text{ polys-rel-full}) (\text{PAC-checker-specification-step2 } (\mathcal{V}_0, A_0) \text{ spec } (\mathcal{V}, B)) \rangle$

$\langle \text{proof} \rangle$

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*:

$\langle \text{PAC-checker-specification-spec spec } A (st, x2) \implies$
 $\text{PAC-checker-specification-step-spec } A \text{ spec } x2 (st', x1a) \implies$
 $\text{PAC-checker-specification-spec spec } A (st', x1a) \rangle$
 $\langle \text{proof} \rangle$

lemma *RES-SPEC-eq*:

$\langle \text{RES } \Phi = \text{SPEC}(\lambda P. P \in \Phi) \rangle$

$\langle \text{proof} \rangle$

lemma *is-failed-is-success-completeD*:

$\langle \neg \text{is-failed } x \implies \neg \text{is-success } x \implies \text{is-found } x \rangle$
 $\langle \text{proof} \rangle$

lemma *PAC-checker-PAC-checker-specification2*:

$\langle (A, B) \in \text{polys-rel-full} \implies$
 $\neg \text{is-failed } a \implies$
 $(a = \text{FOUND} \implies \text{spec} \in \text{pac-ideal } (\text{set-mset } (\text{snd } B))) \implies$
 $\bigcup (\text{vars } ' \text{ set-mset } (\text{ran-m } (\text{snd } A))) \subseteq \text{fst } B \implies$
 $\text{vars spec} \subseteq \text{fst } B \implies$
 $\text{PAC-checker spec } A \text{ a st} \leq \Downarrow (\text{status-rel} \times_r \text{polys-rel-full}) (\text{PAC-checker-specification2 spec } B) \rangle$
 $\langle \text{proof} \rangle$

definition *remap-polys-polynom-bool* :: $\langle \text{int mpoly} \Rightarrow \text{nat set} \Rightarrow (\text{nat}, \text{int-poly}) \text{ fmap} \Rightarrow (\text{status} \times \text{fpac-step}) \text{ nres} \rangle$ **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* :: $\langle \text{int mpoly} \Rightarrow \text{nat set} \Rightarrow (\text{nat}, \text{int-poly}) \text{ fmap} \Rightarrow (\text{status} \times \text{fpac-step}) \text{ nres} \rangle$ **where**

$\langle \text{remap-polys-change-all spec} = (\lambda \mathcal{V} A. \text{SPEC } (\lambda(st, \mathcal{V}', A').$
 $(\neg \text{is-failed } st \longrightarrow$
 $\text{pac-ideal } (\text{set-mset } (\text{ran-m } A)) = \text{pac-ideal } (\text{set-mset } (\text{ran-m } A')) \wedge$
 $\bigcup (\text{vars } ' \text{ set-mset } (\text{ran-m } A)) \subseteq \mathcal{V}' \wedge$
 $\bigcup (\text{vars } ' \text{ set-mset } (\text{ran-m } A')) \subseteq \mathcal{V}') \wedge$
 $(st = \text{FOUND} \longrightarrow \text{spec} \in \# \text{ran-m } A')) \rangle$

lemma *fmap-eq-dom-iff*:

$\langle A = A' \longleftrightarrow \text{dom-m } A = \text{dom-m } A' \wedge (\forall i \in \# \text{dom-m } A. \text{the } (\text{fmlookup } A \ i) = \text{the } (\text{fmlookup } A' \ i)) \rangle$
 $\langle \text{proof} \rangle$

lemma *ideal-remap-incl*:

$\langle \text{finite } A' \implies (\forall a' \in A'. \exists a \in A. a - a' \in B) \implies \text{ideal } (A' \cup B) \subseteq \text{ideal } (A \cup B) \rangle$
 $\langle \text{proof} \rangle$

lemma *pac-ideal-remap-eq*:

$\langle \text{dom-m } b = \text{dom-m } ba \implies$
 $\forall i \in \# \text{dom-m } ba.$
 $\text{the } (\text{fmlookup } b \ i) - \text{the } (\text{fmlookup } ba \ i)$
 $\in \text{More-Modules.ideal polynom-bool} \implies$
 $\text{pac-ideal } ((\lambda x. \text{the } (\text{fmlookup } b \ x)) ' \text{ set-mset } (\text{dom-m } ba)) = \text{pac-ideal } ((\lambda x. \text{the } (\text{fmlookup } ba \ x)) ' \text{ set-mset } (\text{dom-m } ba)) \rangle$
 $\langle \text{proof} \rangle$

lemma *remap-polys-polynom-bool-remap-polys-change-all*:

$\langle \text{remap-polys-polynom-bool spec } \mathcal{V} A \leq \text{remap-polys-change-all spec } \mathcal{V} A \rangle$
 $\langle \text{proof} \rangle$

definition *remap-polys* :: $\langle \text{int mpoly} \Rightarrow \text{nat set} \Rightarrow (\text{nat}, \text{int-poly}) \text{ fmap} \Rightarrow (\text{status} \times \text{fpac-step}) \text{ nres} \rangle$
where

```

   $\langle \text{remap-polys spec} = (\lambda \mathcal{V} A. \text{do} \{$ 
     $\text{dom} \leftarrow \text{SPEC}(\lambda \text{dom}. \text{set-mset} (\text{dom-m } A) \subseteq \text{dom} \wedge \text{finite dom});$ 

     $\text{failed} \leftarrow \text{SPEC}(\lambda :: \text{bool}. \text{True});$ 
     $\text{if failed}$ 
     $\text{then do} \{$ 
       $\text{RETURN} (\text{FAILED}, \mathcal{V}, \text{fmempty})$ 
     $\}$ 
     $\text{else do} \{$ 
       $(b, N) \leftarrow \text{FOREACH dom}$ 
       $(\lambda i (b, \mathcal{V}, A').$ 
         $\text{if } i \in \# \text{ dom-m } A$ 
         $\text{then do} \{$ 
           $p \leftarrow \text{SPEC}(\lambda p. \text{the } (\text{fmlookup } A \ i) - p \in \text{ideal polynom-bool} \wedge \text{vars } p \subseteq \text{vars } (\text{the } (\text{fmlookup}$ 
 $A \ i)))));$ 
           $\text{eq} \leftarrow \text{SPEC}(\lambda \text{eq}. \text{eq} \longrightarrow p = \text{spec});$ 
           $\mathcal{V} \leftarrow \text{SPEC}(\lambda \mathcal{V}'. \mathcal{V} \cup \text{vars } (\text{the } (\text{fmlookup } A \ i)) \subseteq \mathcal{V}');$ 
           $\text{RETURN}(b \vee \text{eq}, \mathcal{V}, \text{fmupd } i \ p \ A')$ 
         $\}$ 
       $\text{else RETURN } (b, \mathcal{V}, A')$ 
       $(\text{False}, \mathcal{V}, \text{fmempty});$ 
       $\text{RETURN} (\text{if } b \text{ then FOUND else SUCCESS}, N)$ 
     $\}$ 
   $\})$ 

```

lemma *remap-polys-spec*:

```

 $\langle \text{remap-polys spec } \mathcal{V} \ A \leq \text{remap-polys-polynom-bool spec } \mathcal{V} \ A \rangle$ 
 $\langle \text{proof} \rangle$ 

```

6.3 Full Checker

definition *full-checker*

```

::  $\langle \text{int-poly} \Rightarrow (\text{nat}, \text{int-poly}) \text{ fmap} \Rightarrow (\text{int-poly}, \text{nat}, \text{nat}) \text{ pac-step list} \Rightarrow (\text{status} \times -) \text{ nres} \rangle$ 
where
 $\langle \text{full-checker spec0 } A \text{ pac} = \text{do} \{$ 
   $\text{spec} \leftarrow \text{normalize-poly-spec spec0};$ 
   $(st, \mathcal{V}, A) \leftarrow \text{remap-polys-change-all spec } \{ \} \ A;$ 
   $\text{if is-failed } st \text{ then}$ 
   $\text{RETURN } (st, \mathcal{V}, A)$ 
   $\text{else do} \{$ 
     $\mathcal{V} \leftarrow \text{SPEC}(\lambda \mathcal{V}'. \mathcal{V} \cup \text{vars spec0} \subseteq \mathcal{V}');$ 
     $\text{PAC-checker spec } (\mathcal{V}, A) \text{ st pac}$ 
   $\}$ 
 $\rangle$ 

```

lemma *restricted-ideal-to-mono*:

```

 $\langle \text{restricted-ideal-to}_I \mathcal{V} \ I \subseteq \text{restricted-ideal-to}_I \mathcal{V}' \ J \implies$ 
 $\mathcal{U} \subseteq \mathcal{V} \implies$ 
 $\text{restricted-ideal-to}_I \mathcal{U} \ I \subseteq \text{restricted-ideal-to}_I \mathcal{U} \ J \rangle$ 
 $\langle \text{proof} \rangle$ 

```

lemma *full-checker-spec*:

assumes $\langle (A, A') \in \text{polys-rel} \rangle$

shows

$\langle \text{full-checker spec } A \text{ pac} \leq \Downarrow \{((st, G), (st', G')). (st, st') \in \text{status-rel} \wedge$

$$(st \neq \text{FAILED} \longrightarrow (G, G') \in \text{polys-rel-full})\}$$

$$(PAC\text{-checker-specification spec } (A'))\rangle$$

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

lemma *full-checker-spec'*:

shows

$$\langle (\text{uncurry2 full-checker}, \text{uncurry2 } (\lambda \text{spec } A -. PAC\text{-checker-specification spec } A)) \in$$

$$(Id \times_r \text{polys-rel}) \times_r Id \rightarrow_f \langle \{((st, G), (st', G')). (st, st') \in \text{status-rel} \wedge$$

$$(st \neq \text{FAILED} \longrightarrow (G, G') \in \text{polys-rel-full})\} \rangle \text{nres-rel} \rangle$$

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

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*:

$$\langle \exists \varphi. \text{bij } (\varphi :: \text{string} \Rightarrow \text{nat}) \rangle$$

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

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* = $\langle \text{string multiset} \rangle$

type-synonym *mset-polynom* =

$\langle (\text{term-poly} * \text{int}) \text{ multiset} \rangle$

definition *normalized-poly* :: $\langle \text{mset-polynom} \Rightarrow \text{bool} \rangle$ **where**

$$\langle \text{normalized-poly } p \longleftrightarrow$$

$$\text{distinct-mset } (\text{fst } \# p) \wedge$$

$$0 \notin \# \text{snd } \# p \rangle$$

lemma *normalized-poly-simps[simp]*:

$$\langle \text{normalized-poly } \{\# \} \rangle$$

$$\langle \text{normalized-poly } (\text{add-mset } t \ p) \longleftrightarrow \text{snd } t \neq 0 \wedge$$

$$\text{fst } t \notin \# \text{fst } \# p \wedge \text{normalized-poly } p \rangle$$

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

lemma *normalized-poly-mono*:

$$\langle \text{normalized-poly } B \Longrightarrow A \subseteq \# B \Longrightarrow \text{normalized-poly } A \rangle$$

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

definition *mult-poly-by-monom* :: $\langle \text{term-poly} * \text{int} \Rightarrow \text{mset-polynom} \Rightarrow \text{mset-polynom} \rangle$ **where**

$$\langle \text{mult-poly-by-monom} = (\lambda \text{ys } q. \text{image-mset } (\lambda \text{xs}. (\text{fst } \text{xs} + \text{fst } \text{ys}, \text{snd } \text{ys} * \text{snd } \text{xs})) \ q) \rangle$$

definition *mult-poly-raw* :: $\langle \text{mset-polynom} \Rightarrow \text{mset-polynom} \Rightarrow \text{mset-polynom} \rangle$ **where**
 $\langle \text{mult-poly-raw } p \ q =$
 $\quad (\text{sum-mset } ((\lambda y. \text{mult-poly-by-monom } y \ q) \ \# \ p)) \rangle$

definition *remove-powers* :: $\langle \text{mset-polynom} \Rightarrow \text{mset-polynom} \rangle$ **where**
 $\langle \text{remove-powers } xs = \text{image-mset } (\text{apfst remdups-mset}) \ xs \rangle$

definition *all-vars-mset* :: $\langle \text{mset-polynom} \Rightarrow \text{string multiset} \rangle$ **where**
 $\langle \text{all-vars-mset } p = \bigcup \# (\text{fst } \# \ p) \rangle$

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**
 $\langle \text{add-to-coefficient} = (\lambda(a, n) \ b. \ \{ \#(a', -) \in \# \ b. \ a' \neq a \# \} +$
 $\quad (\text{if } n + \text{sum-mset } (\text{snd } \# \{ \#(a', -) \in \# \ b. \ a' = a \# \}) = 0 \text{ then } \{ \# \}$
 $\quad \text{else } \{ \#(a, n + \text{sum-mset } (\text{snd } \# \{ \#(a', -) \in \# \ b. \ a' = a \# \})) \# \}) \rangle$

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*:

$\langle n + \text{sum-mset } (\text{snd } \# \{ \#(a', -) \in \# \ b. \ a' = a \# \}) \neq 0 \implies$
 $\quad \text{add-to-coefficient } (a, n) \ b = \{ \#(a', -) \in \# \ b. \ a' \neq a \# \} +$
 $\quad \{ \#(a, n + \text{sum-mset } (\text{snd } \# \{ \#(a', -) \in \# \ b. \ a' = a \# \})) \# \} \rangle$
 $\langle n + \text{sum-mset } (\text{snd } \# \{ \#(a', -) \in \# \ b. \ a' = a \# \}) = 0 \implies$
 $\quad \text{add-to-coefficient } (a, n) \ b = \{ \#(a', -) \in \# \ b. \ a' \neq a \# \} \rangle$ **and**
add-to-coefficient-simps-If:
 $\langle \text{add-to-coefficient } (a, n) \ b = \{ \#(a', -) \in \# \ b. \ a' \neq a \# \} +$
 $\quad (\text{if } n + \text{sum-mset } (\text{snd } \# \{ \#(a', -) \in \# \ b. \ a' = a \# \}) = 0 \text{ then } \{ \# \}$
 $\quad \text{else } \{ \#(a, n + \text{sum-mset } (\text{snd } \# \{ \#(a', -) \in \# \ b. \ a' = a \# \})) \# \} \rangle$
 $\langle \text{proof} \rangle$

interpretation *comp-fun-commute* $\langle \text{add-to-coefficient} \rangle$
 $\langle \text{proof} \rangle$

lemma *normalized-poly-normalize-poly[simp]*:
 $\langle \text{normalized-poly } (\text{normalize-poly } p) \rangle$
 $\langle \text{proof} \rangle$

7.2 Addition

inductive *add-poly-p* :: $\langle \text{mset-polynom} \times \text{mset-polynom} \times \text{mset-polynom} \Rightarrow \text{mset-polynom} \times \text{mset-polynom} \times \text{mset-polynom} \Rightarrow \text{bool} \rangle$ **where**

add-new-coeff-r:

$\langle \text{add-poly-p } (p, \text{add-mset } x \ q, r) \ (p, q, \text{add-mset } x \ r) \rangle \mid$

add-new-coeff-l:

$\langle \text{add-poly-p } (\text{add-mset } x \ p, q, r) \ (p, q, \text{add-mset } x \ r) \rangle \mid$

add-same-coeff-l:

$\langle \text{add-poly-p } (\text{add-mset } (x, n) \ p, q, \text{add-mset } (x, m) \ r) \ (p, q, \text{add-mset } (x, n + m) \ r) \rangle \mid$

add-same-coeff-r:

$\langle \text{add-poly-p } (p, \text{add-mset } (x, n) \ q, \text{add-mset } (x, m) \ r) \ (p, q, \text{add-mset } (x, n + m) \ r) \rangle \mid$

rem-0-coeff:

$\langle \text{add-poly-p } (p, q, \text{add-mset } (x, 0) \ r) \ (p, q, r) \rangle$

inductive-cases *add-poly-pE*: $\langle \text{add-poly-p } S \ T \rangle$

lemmas *add-poly-p-induct* =
add-poly-p.induct[split-format(complete)]

lemma *add-poly-p-empty-l*:
 $\langle \text{add-poly-p}^{**} (p, q, r) \ (\{\#\}, q, p + r) \rangle$
 $\langle \text{proof} \rangle$

lemma *add-poly-p-empty-r*:
 $\langle \text{add-poly-p}^{**} (p, q, r) \ (p, \{\#\}, q + r) \rangle$
 $\langle \text{proof} \rangle$

lemma *add-poly-p-sym*:
 $\langle \text{add-poly-p } (p, q, r) \ (p', q', r') \longleftrightarrow \text{add-poly-p } (q, p, r) \ (q', p', r') \rangle$
 $\langle \text{proof} \rangle$

lemma *wf-if-measure-in-wf*:
 $\langle \text{wf } R \implies (\bigwedge a \ b. (a, b) \in S \implies (\nu \ a, \nu \ b) \in R) \implies \text{wf } S \rangle$
 $\langle \text{proof} \rangle$

lemma *lexn-n*:
 $\langle n > 0 \implies (x \# xs, y \# ys) \in \text{lexn } r \ n \longleftrightarrow$
 $(\text{length } xs = n-1 \wedge \text{length } ys = n-1) \wedge ((x, y) \in r \vee (x = y \wedge (xs, ys) \in \text{lexn } r \ (n-1))) \rangle$
 $\langle \text{proof} \rangle$

lemma *wf-add-poly-p*:
 $\langle \text{wf } \{(x, y). \text{add-poly-p } y \ x\} \rangle$
 $\langle \text{proof} \rangle$

lemma *mult-poly-by-monom-simps[simp]*:
 $\langle \text{mult-poly-by-monom } t \ \{\#\} = \{\#\} \rangle$
 $\langle \text{mult-poly-by-monom } t \ (ps + qs) = \text{mult-poly-by-monom } t \ ps + \text{mult-poly-by-monom } t \ qs \rangle$
 $\langle \text{mult-poly-by-monom } a \ (\text{add-mset } p \ ps) = \text{add-mset } (fst \ a + fst \ p, snd \ a * snd \ p) \ (\text{mult-poly-by-monom } a \ ps) \rangle$
 $\langle \text{proof} \rangle$

inductive *mult-poly-p* :: $\langle \text{mset-polynom} \Rightarrow \text{mset-polynom} \times \text{mset-polynom} \Rightarrow \text{mset-polynom} \times \text{mset-polynom} \Rightarrow \text{bool} \rangle$

for *q* :: *mset-polynom* **where**

mult-step:

$\langle \text{mult-poly-p } q \ (\text{add-mset } (xs, n) \ p, r) \ (p, (\lambda(y, m). (\text{remdups-mset } (xs + ys), n * m))) \ \{\# \ q + r \} \rangle$

lemmas *mult-poly-p-induct* = *mult-poly-p.induct[split-format(complete)]*

7.3 Normalisation

inductive *normalize-poly-p* :: $\langle \text{mset-polynom} \Rightarrow \text{mset-polynom} \Rightarrow \text{bool} \rangle$ **where**

rem-0-coeff[simp, intro]:

$\langle \text{normalize-poly-p } p \ q \implies \text{normalize-poly-p } (\text{add-mset } (xs, 0) \ p) \ q \mid$

merge-dup-coeff[simp, intro]:

$\langle \text{normalize-poly-p } p \ q \implies \text{normalize-poly-p } (\text{add-mset } (xs, m) \ (\text{add-mset } (xs, n) \ p)) \ (\text{add-mset } (xs, m + n) \ q) \mid$

$\text{same}[\text{simp}, \text{intro}]$:
 $\langle \text{normalize-poly-p } p \text{ } p \rangle \mid$
 $\text{keep-coeff}[\text{simp}, \text{intro}]$:
 $\langle \text{normalize-poly-p } p \text{ } q \implies \text{normalize-poly-p } (\text{add-mset } x \text{ } p) (\text{add-mset } x \text{ } q) \rangle$

7.4 Correctness

This locale maps string polynoms to real polynoms.

locale $\text{poly-embed} =$
fixes $\varphi :: \langle \text{string} \Rightarrow \text{nat} \rangle$
assumes $\varphi\text{-inj}: \langle \text{inj } \varphi \rangle$
begin

definition $\text{poly-of-vars} :: \text{term-poly} \Rightarrow ('a :: \{\text{comm-semiring-1}\}) \text{ mpoly}$ **where**
 $\langle \text{poly-of-vars } xs = \text{fold-mset } (\lambda a \text{ } b. \text{Var } (\varphi \text{ } a) * b) (1 :: 'a \text{ mpoly}) xs \rangle$

lemma $\text{poly-of-vars-simps}[\text{simp}]$:
shows
 $\langle \text{poly-of-vars } (\text{add-mset } x \text{ } xs) = \text{Var } (\varphi \text{ } x) * (\text{poly-of-vars } xs :: ('a :: \{\text{comm-semiring-1}\}) \text{ mpoly}) \rangle$ **(is**
 $?A)$ **and**
 $\langle \text{poly-of-vars } (xs + ys) = \text{poly-of-vars } xs * (\text{poly-of-vars } ys :: ('a :: \{\text{comm-semiring-1}\}) \text{ mpoly}) \rangle$ **(is**
 $?B)$
 $\langle \text{proof} \rangle$

definition mononom-of-vars **where**
 $\langle \text{mononom-of-vars} \equiv (\lambda(xs, n). (+) (\text{Const } n * \text{poly-of-vars } xs)) \rangle$

interpretation $\text{comp-fun-commute } \langle \text{mononom-of-vars} \rangle$
 $\langle \text{proof} \rangle$

lemma $[\text{simp}]$:
 $\langle \text{poly-of-vars } \{\# \} = 1 \rangle$
 $\langle \text{proof} \rangle$

lemma $\text{mononom-of-vars-add}[\text{simp}]$:
 $\langle \text{NO-MATCH } 0 \text{ } b \implies \text{mononom-of-vars } xs \text{ } b = \text{Const } (\text{snd } xs) * \text{poly-of-vars } (\text{fst } xs) + b \rangle$
 $\langle \text{proof} \rangle$

definition $\text{polynom-of-mset} :: \langle \text{mset-polynom} \Rightarrow \cdot \rangle$ **where**
 $\langle \text{polynom-of-mset } p = \text{sum-mset } (\text{mononom-of-vars } \text{'\# } p) 0 \rangle$

lemma $\text{polynom-of-mset-append}[\text{simp}]$:
 $\langle \text{polynom-of-mset } (xs + ys) = \text{polynom-of-mset } xs + \text{polynom-of-mset } ys \rangle$
 $\langle \text{proof} \rangle$

lemma $\text{polynom-of-mset-Cons}[\text{simp}]$:
 $\langle \text{polynom-of-mset } (\text{add-mset } x \text{ } ys) = \text{Const } (\text{snd } x) * \text{poly-of-vars } (\text{fst } x) + \text{polynom-of-mset } ys \rangle$
 $\langle \text{proof} \rangle$

lemma $\text{polynom-of-mset-empty}[\text{simp}]$:
 $\langle \text{polynom-of-mset } \{\# \} = 0 \rangle$
 $\langle \text{proof} \rangle$

lemma $\text{polynom-of-mset-mult-poly-by-monom}[\text{simp}]$:

$\langle \text{polynom-of-mset } (\text{mult-poly-by-monom } x \text{ } ys) =$
 $(\text{Const } (\text{snd } x) * \text{poly-of-vars } (\text{fst } x) * \text{polynom-of-mset } ys) \rangle$
 $\langle \text{proof} \rangle$

lemma *polynom-of-mset-mult-poly-raw[simp]:*
 $\langle \text{polynom-of-mset } (\text{mult-poly-raw } xs \text{ } ys) = \text{polynom-of-mset } xs * \text{polynom-of-mset } ys \rangle$
 $\langle \text{proof} \rangle$

lemma *polynom-of-mset-uminus:*
 $\langle \text{polynom-of-mset } \{\# \text{case } x \text{ of } (a, b) \Rightarrow (a, - b). x \in \# \text{ } za\#\} =$
 $- \text{polynom-of-mset } za \rangle$
 $\langle \text{proof} \rangle$

lemma *X2-X-polynom-bool-mult-in:*
 $\langle \text{Var } (x1) * (\text{Var } (x1) * p) - \text{Var } (x1) * p \in \text{More-Modules.ideal polynom-bool} \rangle$
 $\langle \text{proof} \rangle$

lemma *polynom-of-list-remove-powers-polynom-bool:*
 $\langle (\text{polynom-of-mset } xs) - \text{polynom-of-mset } (\text{remove-powers } xs) \in \text{ideal polynom-bool} \rangle$
 $\langle \text{proof} \rangle$

lemma *add-poly-p-polynom-of-mset:*
 $\langle \text{add-poly-p } (p, q, r) (p', q', r') \Rightarrow$
 $\text{polynom-of-mset } r + (\text{polynom-of-mset } p + \text{polynom-of-mset } q) =$
 $\text{polynom-of-mset } r' + (\text{polynom-of-mset } p' + \text{polynom-of-mset } q') \rangle$
 $\langle \text{proof} \rangle$

lemma *rtrancp-add-poly-p-polynom-of-mset:*
 $\langle \text{add-poly-p}^{**} (p, q, r) (p', q', r') \Rightarrow$
 $\text{polynom-of-mset } r + (\text{polynom-of-mset } p + \text{polynom-of-mset } q) =$
 $\text{polynom-of-mset } r' + (\text{polynom-of-mset } p' + \text{polynom-of-mset } q') \rangle$
 $\langle \text{proof} \rangle$

lemma *rtrancp-add-poly-p-polynom-of-mset-full:*
 $\langle \text{add-poly-p}^{**} (p, q, \{\#\}) (\{\#\}, \{\#\}, r') \Rightarrow$
 $\text{polynom-of-mset } r' = (\text{polynom-of-mset } p + \text{polynom-of-mset } q) \rangle$
 $\langle \text{proof} \rangle$

lemma *poly-of-vars-remdups-mset:*
 $\langle \text{poly-of-vars } (\text{remdups-mset } (xs)) - (\text{poly-of-vars } xs)$
 $\in \text{More-Modules.ideal polynom-bool} \rangle$
 $\langle \text{proof} \rangle$

lemma *polynom-of-mset-mult-map:*
 $\langle \text{polynom-of-mset}$
 $\{\# \text{case } x \text{ of } (ys, n) \Rightarrow (\text{remdups-mset } (ys + xs), n * m). x \in \# \text{ } q\#\} -$
 $\text{Const } m * (\text{poly-of-vars } xs * \text{polynom-of-mset } q)$
 $\in \text{More-Modules.ideal polynom-bool} \rangle$
 $(\text{is } \langle ?P \text{ } q \in \cdot \rangle)$
 $\langle \text{proof} \rangle$

lemma *mult-poly-p-mult-ideal:*

$\langle \text{mult-poly-p } q \ (p, r) \ (p', r') \implies$
 $(\text{polynom-of-mset } p' * \text{polynom-of-mset } q + \text{polynom-of-mset } r') - (\text{polynom-of-mset } p * \text{polynom-of-mset } q + \text{polynom-of-mset } r)$
 $\in \text{ideal polynom-bool}$
 $\langle \text{proof} \rangle$

lemma *rtrancp-mult-poly-p-mult-ideal*:

$\langle (\text{mult-poly-p } q)^{**} \ (p, r) \ (p', r') \implies$
 $(\text{polynom-of-mset } p' * \text{polynom-of-mset } q + \text{polynom-of-mset } r') - (\text{polynom-of-mset } p * \text{polynom-of-mset } q + \text{polynom-of-mset } r)$
 $\in \text{ideal polynom-bool}$
 $\langle \text{proof} \rangle$

lemma *rtrancp-mult-poly-p-mult-ideal-final*:

$\langle (\text{mult-poly-p } q)^{**} \ (p, \{\#\}) \ (\{\#\}, r) \implies$
 $(\text{polynom-of-mset } r) - (\text{polynom-of-mset } p * \text{polynom-of-mset } q)$
 $\in \text{ideal polynom-bool}$
 $\langle \text{proof} \rangle$

lemma *normalize-poly-p-poly-of-mset*:

$\langle \text{normalize-poly-p } p \ q \implies \text{polynom-of-mset } p = \text{polynom-of-mset } q \rangle$
 $\langle \text{proof} \rangle$

lemma *rtrancp-normalize-poly-p-poly-of-mset*:

$\langle \text{normalize-poly-p}^{**} \ p \ q \implies \text{polynom-of-mset } p = \text{polynom-of-mset } q \rangle$
 $\langle \text{proof} \rangle$

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 $\varphi\text{-bij}$: $\langle \text{bij-betw } \varphi \ V \ N \rangle$

begin

definition $\varphi' :: \langle \text{nat} \Rightarrow \text{string} \rangle$ **where**

$\langle \varphi' = \text{the-inv-into } V \ \varphi \rangle$

lemma $\varphi'\text{-}\varphi[\text{simp}]$:

$\langle x \in V \implies \varphi' (\varphi \ x) = x \rangle$

$\langle \text{proof} \rangle$

lemma $\varphi\text{-}\varphi'[\text{simp}]$:

$\langle x \in N \implies \varphi (\varphi' \ x) = x \rangle$

$\langle \text{proof} \rangle$

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

$\langle (\text{uncurry } f, g) \in [P]_f A \rightarrow \langle B \rangle \text{nres-rel} \Rightarrow$

$(\bigwedge a \ b \ x'. P \ x' \Rightarrow ((a, b), x') \in A \Rightarrow f \ a \ b \leq \Downarrow B \ (g \ x')) \rangle$

$\langle \text{proof} \rangle$

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

$\langle (f, \text{uncurry } g) \in [P]_f A \rightarrow \langle B \rangle \text{nres-rel} \Rightarrow$

$(\bigwedge a \ b \ x'. P \ (a, b) \Rightarrow (x', (a, b)) \in A \Rightarrow f \ x' \leq \Downarrow B \ (g \ a \ b)) \rangle$

$\langle \text{proof} \rangle$

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*

$\langle \text{proof} \rangle$

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*:

$\langle (x, y) \in \text{less-than-char} \longleftrightarrow \text{less-char } x \ y \rangle$

$\langle \text{proof} \rangle$

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**

antisym-less-than-char:

$\langle \text{antisym less-than-char} \rangle$

$\langle \text{proof} \rangle$

8.2 Polynomials

definition *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**
 $\langle \text{term-poly-list-rel} = \{(xs, ys).$
 $\quad ys = \text{mset } xs \wedge$
 $\quad \text{distinct } xs \wedge$
 $\quad \text{sorted-wrt } (\text{rel2p var-order-rel}) \text{ } xs\}$

definition $\text{unsorted-term-poly-list-rel} :: \langle (\text{term-poly-list} \times \text{term-poly}) \text{ set} \rangle$ **where**
 $\langle \text{unsorted-term-poly-list-rel} = \{(xs, ys).$
 $\quad ys = \text{mset } xs \wedge \text{distinct } xs\}$

definition $\text{poly-list-rel} :: \langle \text{'a} \Rightarrow ((\text{'a} \times \text{int}) \text{ list} \times \text{mset-polynom}) \text{ set} \rangle$ **where**
 $\langle \text{poly-list-rel } R = \{(xs, ys).$
 $\quad (xs, ys) \in \langle R \times_r \text{int-rel} \rangle \text{list-rel } O \text{ list-mset-rel} \wedge$
 $\quad 0 \notin \# \text{snd '}\# \text{ } ys\}$

definition $\text{sorted-poly-list-rel-wrt} :: \langle \text{'a} \Rightarrow \text{'a} \Rightarrow \text{bool} \rangle$
 $\Rightarrow \langle \text{'a} \times \text{string multiset} \rangle \text{ set} \Rightarrow \langle (\text{'a} \times \text{int}) \text{ list} \times \text{mset-polynom} \rangle \text{ set} \rangle$ **where**
 $\langle \text{sorted-poly-list-rel-wrt } S \text{ } R = \{(xs, ys).$
 $\quad (xs, ys) \in \langle R \times_r \text{int-rel} \rangle \text{list-rel } O \text{ list-mset-rel} \wedge$
 $\quad \text{sorted-wrt } S \text{ } (\text{map fst } xs) \wedge$
 $\quad \text{distinct } (\text{map fst } xs) \wedge$
 $\quad 0 \notin \# \text{snd '}\# \text{ } ys\}$

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 sorted-poly-rel **where**
 $\langle \text{sorted-poly-rel} \equiv \text{sorted-poly-list-rel term-order} \rangle$

definition $\text{sorted-repeat-poly-list-rel-wrt} :: \langle \text{'a} \Rightarrow \text{'a} \Rightarrow \text{bool} \rangle$
 $\Rightarrow \langle \text{'a} \times \text{string multiset} \rangle \text{ set} \Rightarrow \langle (\text{'a} \times \text{int}) \text{ list} \times \text{mset-polynom} \rangle \text{ set} \rangle$ **where**
 $\langle \text{sorted-repeat-poly-list-rel-wrt } S \text{ } R = \{(xs, ys).$
 $\quad (xs, ys) \in \langle R \times_r \text{int-rel} \rangle \text{list-rel } O \text{ list-mset-rel} \wedge$
 $\quad \text{sorted-wrt } S \text{ } (\text{map fst } xs) \wedge$
 $\quad 0 \notin \# \text{snd '}\# \text{ } ys\}$

abbreviation $\text{sorted-repeat-poly-list-rel}$ **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 $\text{sorted-repeat-poly-rel}$ **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]*:

$\langle ([], s') \in \text{sorted-poly-list-rel-wrt } S \ T \longleftrightarrow s' = \{\#\} \rangle$
 $\langle \text{proof} \rangle$

definition *fully-unsorted-poly-list-rel* :: $\langle - \Rightarrow (('a \times \text{int}) \text{ list} \times \text{mset-polynom}) \text{ set} \rangle$ **where**

$\langle \text{fully-unsorted-poly-list-rel } R = \{(xs, ys). \langle xs, ys \rangle \in \langle R \times_r \text{int-rel} \rangle \text{list-rel } O \text{list-mset-rel} \} \rangle$

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]*:

$\langle (p, \{\#\}) \in \text{fully-unsorted-poly-list-rel } R \longleftrightarrow p = [] \rangle$
 $\langle ([], p') \in \text{fully-unsorted-poly-list-rel } R \longleftrightarrow p' = \{\#\} \rangle$
 $\langle \text{proof} \rangle$

definition *poly-list-rel-with0* :: $\langle - \Rightarrow (('a \times \text{int}) \text{ list} \times \text{mset-polynom}) \text{ set} \rangle$ **where**

$\langle \text{poly-list-rel-with0 } R = \{(xs, ys). \langle xs, ys \rangle \in \langle R \times_r \text{int-rel} \rangle \text{list-rel } O \text{list-mset-rel} \} \rangle$

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]*:

$\langle (p, \{\#\}) \in \text{poly-list-rel-with0 } R \longleftrightarrow p = [] \rangle$
 $\langle ([], p') \in \text{poly-list-rel-with0 } R \longleftrightarrow p' = \{\#\} \rangle$
 $\langle \text{proof} \rangle$

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**
 $\langle \text{sorted-repeat-poly-list-rel-with0-wrt } S \ R = \{(xs, ys). \langle xs, ys \rangle \in \langle R \times_r \text{int-rel} \rangle \text{list-rel } O \text{list-mset-rel} \wedge \text{sorted-wrt } S \ (\text{map fst } xs) \} \rangle$

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*:

$\langle (xs, ys) \in \text{term-poly-list-rel} \implies \text{distinct } xs \rangle$
 $\langle (xs, ys) \in \text{term-poly-list-rel} \implies ys = \text{mset } xs \rangle$
 $\langle (xs, ys) \in \text{term-poly-list-rel} \implies \text{sorted-wrt } (\text{rel2p var-order-rel}) \ xs \rangle$
 $\langle (xs, ys) \in \text{term-poly-list-rel} \implies \text{sorted-wrt } (\text{rel2p } (\text{Id} \cup \text{var-order-rel})) \ xs \rangle$
 $\langle \text{proof} \rangle$

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'* :: $\langle \text{llist-polynom} \times \text{llist-polynom} \Rightarrow \text{llist-polynom} \rangle$ **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* :: $\langle \text{llist-polynom} \times \text{llist-polynom} \Rightarrow \text{llist-polynom nres} \rangle$ **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*]:

$\langle \text{nonzero-coeffs } \{ \# \} \rangle$

$\langle \text{nonzero-coeffs } (\text{add-mset } (xs, n) a) \longleftrightarrow \text{nonzero-coeffs } a \wedge n \neq 0 \rangle$
 $\langle \text{proof} \rangle$

lemma *nonzero-coeffsD*:

$\langle \text{nonzero-coeffs } a \implies (x, n) \in \# a \implies n \neq 0 \rangle$
 $\langle \text{proof} \rangle$

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
 $(\text{mset } ys, n) \in \# a \wedge (\forall x \in \text{set } p. S \text{ } ys \text{ } (\text{fst } x)) \wedge \text{sorted-wrt } (\text{rel2p var-order-rel}) \text{ } ys \wedge$
 $\text{distinct } ys \wedge ys \notin \text{set } (\text{map } \text{fst } p) \wedge n \neq 0 \wedge \text{nonzero-coeffs } a \rangle$
 $\langle \text{proof} \rangle$

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
 $(\text{mset } ys, n) \in \# a \wedge (\forall x \in \text{set } p. S \text{ } ys \text{ } (\text{fst } x)) \wedge \text{sorted-wrt } (\text{rel2p var-order-rel}) \text{ } ys \wedge$
 $\text{distinct } ys \wedge ys \notin \text{set } (\text{map } \text{fst } p) \wedge n \neq 0 \wedge \text{nonzero-coeffs } a \rangle$
 $\langle \text{proof} \rangle$

lemma *sorted-repeat-poly-list-rel-ConsD*:

$\langle ((ys, n) \# p, a) \in \text{sorted-repeat-poly-list-rel } S \implies (p, \text{remove1-mset } (\text{mset } ys, n) a) \in \text{sorted-repeat-poly-list-rel } S$
 \wedge
 $(\text{mset } ys, n) \in \# a \wedge (\forall x \in \text{set } p. S \text{ } ys \text{ } (\text{fst } x)) \wedge \text{sorted-wrt } (\text{rel2p var-order-rel}) \text{ } ys \wedge$
 $\text{distinct } ys \wedge n \neq 0 \wedge \text{nonzero-coeffs } a \rangle$
 $\langle \text{proof} \rangle$

lemma *sorted-repeat-poly-list-rel-Cons-iff*:

$\langle ((ys, n) \# p, a) \in \text{sorted-repeat-poly-list-rel } S \longleftrightarrow (p, \text{remove1-mset } (\text{mset } ys, n) a) \in \text{sorted-repeat-poly-list-rel } S$
 \wedge
 $(\text{mset } ys, n) \in \# a \wedge (\forall x \in \text{set } p. S \text{ } ys \text{ } (\text{fst } x)) \wedge \text{sorted-wrt } (\text{rel2p var-order-rel}) \text{ } ys \wedge$
 $\text{distinct } ys \wedge n \neq 0 \wedge \text{nonzero-coeffs } a \rangle$
 $\langle \text{proof} \rangle$

lemma *add-poly-p-add-mset-sum-0*:

$\langle n + m = 0 \implies \text{add-poly-p}^{**} (A, Aa, \{\#\}) (\{\#\}, \{\#\}, r) \implies$
 add-poly-p^{**}
 $(\text{add-mset } (\text{mset } ys, n) A, \text{add-mset } (\text{mset } ys, m) Aa, \{\#\})$
 $(\{\#\}, \{\#\}, r) \rangle$
 $\langle \text{proof} \rangle$

lemma *monoms-add-poly-l'D*:

$\langle (aa, ba) \in \text{set } (\text{add-poly-l'} x) \implies aa \in \text{fst } ' \text{set } (\text{fst } x) \vee aa \in \text{fst } ' \text{set } (\text{snd } x) \rangle$
 $\langle \text{proof} \rangle$

lemma *add-poly-p-add-to-result*:

$\langle \text{add-poly-p}^{**} (A, B, r) (A', B', r') \implies$
 add-poly-p^{**}
 $(A, B, p + r) (A', B', p + r') \rangle$
 $\langle \text{proof} \rangle$

lemma *add-poly-p-add-mset-comb*:

$\langle \text{add-poly-p}^{**} (A, Aa, \{\#\}) (\{\#\}, \{\#\}, r) \implies$
 add-poly-p^{**}
 $(\text{add-mset } (xs, n) A, Aa, \{\#\})$
 $(\{\#\}, \{\#\}, \text{add-mset } (xs, n) r) \rangle$
 $\langle \text{proof} \rangle$

lemma *add-poly-p-add-mset-comb2*:

$\langle \text{add-poly-p}^{**} (A, Aa, \{\#\}) (\{\#\}, \{\#\}, r) \implies$
 add-poly-p^{**}
 $(\text{add-mset } (ys, n) A, \text{add-mset } (ys, m) Aa, \{\#\})$
 $(\{\#\}, \{\#\}, \text{add-mset } (ys, n + m) r) \rangle$
 $\langle \text{proof} \rangle$

lemma *add-poly-p-add-mset-comb3*:

$\langle \text{add-poly-p}^{**} (A, Aa, \{\#\}) (\{\#\}, \{\#\}, r) \implies$
 add-poly-p^{**}
 $(A, \text{add-mset } (ys, m) Aa, \{\#\})$
 $(\{\#\}, \{\#\}, \text{add-mset } (ys, m) r) \rangle$
 $\langle \text{proof} \rangle$

lemma *total-on-lexord*:

$\langle \text{Relation.total-on UNIV } R \implies \text{Relation.total-on UNIV } (\text{lexord } R) \rangle$
 $\langle \text{proof} \rangle$

lemma *antisym-lexord*:

$\langle \text{antisym } R \implies \text{irrefl } R \implies \text{antisym } (\text{lexord } R) \rangle$
 $\langle \text{proof} \rangle$

lemma *less-than-char-linear*:

$\langle (a, b) \in \text{less-than-char} \vee$
 $a = b \vee (b, a) \in \text{less-than-char} \rangle$
 $\langle \text{proof} \rangle$

lemma *total-on-lexord-less-than-char-linear*:

$\langle xs \neq ys \implies (xs, ys) \notin \text{lexord } (\text{lexord less-than-char}) \longleftrightarrow$
 $(ys, xs) \in \text{lexord } (\text{lexord less-than-char}) \rangle$
 $\langle \text{proof} \rangle$

lemma *sorted-poly-list-rel-nonzeroD*:

$\langle (p, r) \in \text{sorted-poly-list-rel term-order} \implies$
 $\text{nonzero-coeffs } (r) \rangle$
 $\langle (p, r) \in \text{sorted-poly-list-rel } (\text{rel2p } (\text{lexord } (\text{lexord less-than-char}))) \implies$
 $\text{nonzero-coeffs } (r) \rangle$
 $\langle \text{proof} \rangle$

lemma *add-poly-l'-add-poly-p*:

assumes $\langle (pq, pq') \in \text{sorted-poly-rel} \times_r \text{sorted-poly-rel} \rangle$
shows $\langle \exists r. (\text{add-poly-l' } pq, r) \in \text{sorted-poly-rel} \wedge$
 $\text{add-poly-p}^{**} (\text{fst } pq', \text{snd } pq', \{\#\}) (\{\#\}, \{\#\}, r) \rangle$
 $\langle \text{proof} \rangle$

lemma *add-poly-l-add-poly*:

$\langle \text{add-poly-l } x = \text{RETURN } (\text{add-poly-l}' x) \rangle$
 $\langle \text{proof} \rangle$

lemma *add-poly-l-spec*:

$\langle (\text{add-poly-l, uncurry } (\lambda p \ q. \text{SPEC}(\lambda r. \text{add-poly-p}^{**} (p, q, \{\#\}) (\{\#\}, \{\#\}, r)))) \in$
 $\text{sorted-poly-rel} \times_r \text{sorted-poly-rel} \rightarrow_f \langle \text{sorted-poly-rel} \rangle_{\text{nres-rel}} \rangle$
 $\langle \text{proof} \rangle$

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 $\langle \text{sort-poly-spec } p \leq \Downarrow (\text{sorted-repeat-poly-rel}) (\text{RETURN } p') \rangle$
 $\langle \text{proof} \rangle$

9.2 Multiplication

fun *mult-monoms* :: $\langle \text{term-poly-list} \Rightarrow \text{term-poly-list} \Rightarrow \text{term-poly-list} \rangle$ **where**

$\langle \text{mult-monoms } p [] = p \rangle \mid$
 $\langle \text{mult-monoms } [] p = p \rangle \mid$
 $\langle \text{mult-monoms } (x \# p) (y \# q) =$
 $\text{if } x = y \text{ then } x \# \text{mult-monoms } p \ q$
 $\text{else if } (x, y) \in \text{var-order-rel} \text{ then } x \# \text{mult-monoms } p (y \# q)$
 $\text{else } y \# \text{mult-monoms } (x \# p) \ q \rangle$

lemma *term-poly-list-rel-empty-iff[simp]*:

$\langle ([], q') \in \text{term-poly-list-rel} \longleftrightarrow q' = \{\#\} \rangle$
 $\langle \text{proof} \rangle$

lemma *term-poly-list-rel-Cons-iff*:

$\langle (y \# p, p') \in \text{term-poly-list-rel} \longleftrightarrow$
 $(p, \text{remove1-mset } y \ p') \in \text{term-poly-list-rel} \wedge$
 $y \in \# p' \wedge y \notin \text{set } p \wedge y \notin \# \text{remove1-mset } y \ p' \wedge$
 $(\forall x \in \# \text{mset } p. (y, x) \in \text{var-order-rel}) \rangle$
 $\langle \text{proof} \rangle$

lemma *var-order-rel-antisym[simp]*:

$\langle (y, y) \notin \text{var-order-rel} \rangle$
 $\langle \text{proof} \rangle$

lemma *term-poly-list-rel-remdups-mset*:

$\langle (p, p') \in \text{term-poly-list-rel} \implies$
 $(p, \text{remdups-mset } p') \in \text{term-poly-list-rel} \rangle$
 $\langle \text{proof} \rangle$

lemma *var-notin-notin-mult-monomsD*:

$\langle y \in \text{set } (\text{mult-monoms } p \ q) \implies y \in \text{set } p \vee y \in \text{set } q \rangle$
 $\langle \text{proof} \rangle$

lemma *term-poly-list-rel-set-mset*:

$\langle (p, q) \in \text{term-poly-list-rel} \implies \text{set } p = \text{set-mset } q \rangle$
 $\langle \text{proof} \rangle$

lemma *mult-monoms-spec*:

$\langle (mult-monoms, (\lambda a b. remdups-mset (a + b))) \in term-poly-list-rel \rightarrow term-poly-list-rel \rightarrow term-poly-list-rel \rangle$
 $\langle proof \rangle$

definition *mult-monomials* :: $\langle term-poly-list \times int \Rightarrow term-poly-list \times int \Rightarrow term-poly-list \times int \rangle$ **where**

$\langle mult-monomials = (\lambda(x, a) (y, b). (mult-monoms x y, a * b)) \rangle$

definition *mult-poly-raw* :: $\langle llist-polynom \Rightarrow llist-polynom \Rightarrow llist-polynom \rangle$ **where**

$\langle mult-poly-raw p q = foldl (\lambda b x. map (mult-monomials x) q @ b) [] p \rangle$

fun *map-append* **where**

$\langle map-append f b [] = b \rangle$ |
 $\langle map-append f b (x \# xs) = f x \# map-append f b xs \rangle$

lemma *map-append-alt-def*:

$\langle map-append f b xs = map f xs @ b \rangle$
 $\langle proof \rangle$

lemma *foldl-append-empty*:

$\langle NO-MATCH [] xs \Longrightarrow foldl (\lambda b x. f x @ b) xs p = foldl (\lambda b x. f x @ b) [] p @ xs \rangle$
 $\langle proof \rangle$

lemma *poly-list-rel-empty-iff[simp]*:

$\langle ([], r) \in poly-list-rel R \longleftrightarrow r = \{\#\} \rangle$
 $\langle proof \rangle$

lemma *mult-poly-raw-simp[simp]*:

$\langle mult-poly-raw [] q = [] \rangle$
 $\langle mult-poly-raw (x \# p) q = mult-poly-raw p q @ map (mult-monomials x) q \rangle$
 $\langle proof \rangle$

lemma *sorted-poly-list-relD*:

$\langle (q, q') \in sorted-poly-list-rel R \Longrightarrow q' = (\lambda(a, b). (mset a, b)) \text{ ‘\#’ } mset q \rangle$
 $\langle proof \rangle$

lemma *list-all2-in-set-ExD*:

$\langle list-all2 R p q \Longrightarrow x \in set p \Longrightarrow \exists y \in set q. R x y \rangle$
 $\langle proof \rangle$

inductive-cases *mult-poly-p-elim*: $\langle mult-poly-p q (A, r) (B, r') \rangle$

lemma *mult-poly-p-add-mset-same*:

$\langle (mult-poly-p q)^{**} (A, r) (B, r') \Longrightarrow (mult-poly-p q)^{**} (add-mset x A, r) (add-mset x B, r') \rangle$
 $\langle proof \rangle$

lemma *mult-poly-raw-mult-poly-p*:

assumes $\langle (p, p') \in sorted-poly-rel \rangle$ **and** $\langle (q, q') \in sorted-poly-rel \rangle$
shows $\langle \exists r. (mult-poly-raw p q, r) \in unsorted-poly-rel \wedge (mult-poly-p q)^{**} (p', \{\#\}) (\{\#\}, r) \rangle$
 $\langle proof \rangle$

fun *merge-coeffs* :: $\langle llist-polynom \Rightarrow llist-polynom \rangle$ **where**

$\langle merge-coeffs [] = [] \rangle$ |
 $\langle merge-coeffs [(xs, n)] = [(xs, n)] \rangle$ |

$\langle \text{merge-coeffs } ((xs, n) \# (ys, m) \# p) =$
 $\quad (\text{if } xs = ys$
 $\quad \text{then if } n + m \neq 0 \text{ then merge-coeffs } ((xs, n + m) \# p) \text{ else merge-coeffs } p$
 $\quad \text{else } (xs, n) \# \text{merge-coeffs } ((ys, m) \# p)) \rangle$

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*:
 $\langle \text{mononoms } (\text{merge-coeffs } p) \subseteq \text{mononoms } p \rangle$
 $\langle \text{proof} \rangle$

lemma *fst-normalize-polynom-subsetD*:
 $\langle (a, b) \in \text{set } (\text{merge-coeffs } p) \implies a \in \text{mononoms } p \rangle$
 $\langle \text{proof} \rangle$

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 $\langle \text{distinct (map fst (merge-coeffs } xs)) \rangle$
 $\langle \text{proof} \rangle$

lemma *in-set-merge-coeffsD*:
 $\langle (a, b) \in \text{set } (\text{merge-coeffs } p) \implies \exists b. (a, b) \in \text{set } p \rangle$
 $\langle \text{proof} \rangle$

lemma *rtranclp-normalize-poly-add-mset*:
 $\langle \text{normalize-poly-}p^{**} A r \implies \text{normalize-poly-}p^{**} (\text{add-mset } x A) (\text{add-mset } x r) \rangle$
 $\langle \text{proof} \rangle$

lemma *nonzero-coeffs-diff*:
 $\langle \text{nonzero-coeffs } A \implies \text{nonzero-coeffs } (A - B) \rangle$
 $\langle \text{proof} \rangle$

lemma *merge-coeffs-is-normalize-poly-p*:
 $\langle (xs, ys) \in \text{sorted-repeat-poly-rel} \implies \exists r. (\text{merge-coeffs } xs, r) \in \text{sorted-poly-rel} \wedge \text{normalize-poly-}p^{**} ys$
 $r \rangle$
 $\langle \text{proof} \rangle$

9.3 Normalisation

definition *normalize-poly* **where**
 $\langle \text{normalize-poly } p = \text{do } \{$
 $\quad p \leftarrow \text{sort-poly-spec } p;$
 $\quad \text{RETURN } (\text{merge-coeffs } p)$
 $\} \rangle$

definition *sort-coeff* $:: \langle \text{string list} \Rightarrow \text{string list nres} \rangle$ **where**
 $\langle \text{sort-coeff } ys = \text{SPEC}(\lambda xs. \text{mset } xs = \text{mset } ys \wedge \text{sorted-wrt } (\text{rel2p } (\text{Id} \cup \text{var-order-rel})) xs) \rangle$

lemma *distinct-var-order-Id-var-order*:
 $\langle \text{distinct } a \implies \text{sorted-wrt } (\text{rel2p } (\text{Id} \cup \text{var-order-rel})) a \implies$
 $\quad \text{sorted-wrt var-order } a \rangle$
 $\langle \text{proof} \rangle$

definition *sort-all-coeffs* :: $\langle \text{llist-polynom} \Rightarrow \text{llist-polynom nres} \rangle$ **where**
 $\langle \text{sort-all-coeffs } xs = \text{monadic-nfoldli } xs \ (\lambda\cdot. \text{RETURN } \text{True}) \ (\lambda(a, n) \ b. \text{do } \{a \leftarrow \text{sort-coeff } a; \text{RETURN } ((a, n) \# b)\}) \ [] \rangle$

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 $\langle \text{monadic-nfoldli } xs \ (\lambda\cdot. \text{RETURN } \text{True}) \ (\lambda(a, n) \ b. \text{do } \{a \leftarrow \text{sort-coeff } a; \text{RETURN } ((a, n) \# b)\}) \ xs' \leq$
 $\Downarrow \text{Id } (\text{SPEC}(\lambda ys. \text{map } (\lambda(a, b). (\text{mset } a, b)) \ (\text{rev } xs \ @ \ xs') = \text{map } (\lambda(a, b). (\text{mset } a, b)) \ (ys) \wedge$
 $(\forall xs \in \text{mononoms } ys. \text{sorted-wrt } (\text{rel2p } (\text{var-order-rel})) \ xs))) \rangle$
 $\langle \text{proof} \rangle$

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*:

$\langle \forall x \in \text{mononoms } xs. \text{distinct } x \implies$
 $\text{sort-all-coeffs } xs \leq \Downarrow \text{Id } (\text{shuffle-coefficients } xs) \rangle$
 $\langle \text{proof} \rangle$

lemma *unsorted-term-poly-list-rel-mset*:

$\langle (ys, aa) \in \text{unsorted-term-poly-list-rel} \implies \text{mset } ys = aa \rangle$
 $\langle \text{proof} \rangle$

lemma *RETURN-map-alt-def*:

$\langle \text{RETURN } o \ (\text{map } f) =$
 $\text{REC}_T \ (\lambda g \ xs.$
 $\text{case } xs \text{ of}$
 $\quad [] \Rightarrow \text{RETURN } []$
 $\quad | x \# xs \Rightarrow \text{do } \{xs \leftarrow g \ xs; \text{RETURN } (f \ x \ # \ xs)\} \rangle$
 $\langle \text{proof} \rangle$

lemma *fully-unsorted-poly-rel-Cons-iff*:

$\langle ((ys, n) \# p, a) \in \text{fully-unsorted-poly-rel} \longleftrightarrow$
 $(p, \text{remove1-mset } (\text{mset } ys, n) \ a) \in \text{fully-unsorted-poly-rel} \wedge$
 $(\text{mset } ys, n) \in \# \ a \wedge \text{distinct } ys \rangle$
 $\langle \text{proof} \rangle$

lemma *map-mset-unsorted-term-poly-list-rel*:

$\langle (\bigwedge a. a \in \text{mononoms } s \implies \text{distinct } a) \implies \forall x \in \text{mononoms } s. \text{distinct } x \implies$
 $(\forall xs \in \text{mononoms } s. \text{sorted-wrt } (\text{rel2p } (\text{Id} \cup \text{var-order-rel})) \ xs) \implies$
 $(s, \text{map } (\lambda(a, y). (\text{mset } a, y)) \ s)$
 $\in (\text{term-poly-list-rel} \times_r \text{int-rel}) \text{list-rel} \rangle$
 $\langle \text{proof} \rangle$

lemma *list-rel-unsorted-term-poly-list-relD*:

$\langle (p, y) \in (\text{unsorted-term-poly-list-rel} \times_r \text{int-rel}) \text{list-rel} \implies$
 $\text{mset } y = (\lambda(a, y). (\text{mset } a, y)) \ ' \# \ \text{mset } p \wedge (\forall x \in \text{mononoms } p. \text{distinct } x) \rangle$
 $\langle \text{proof} \rangle$

lemma *shuffle-terms-distinct-iff*:

assumes $\langle \text{map } (\lambda(a, y). (\text{mset } a, y)) \text{ } p = \text{map } (\lambda(a, y). (\text{mset } a, y)) \text{ } s \rangle$
shows $\langle (\forall x \in \text{set } p. \text{distinct } (\text{fst } x)) \longleftrightarrow (\forall x \in \text{set } s. \text{distinct } (\text{fst } x)) \rangle$
 $\langle \text{proof} \rangle$

lemma

$\langle (p, y) \in \langle \text{unsorted-term-poly-list-rel} \times_r \text{int-rel} \rangle \text{list-rel} \implies$
 $(a, b) \in \text{set } p \implies \text{distinct } a \rangle$
 $\langle \text{proof} \rangle$

lemma *sort-all-coeffs-unsorted-poly-rel-with0*:

assumes $\langle (p, p') \in \text{fully-unsorted-poly-rel} \rangle$
shows $\langle \text{sort-all-coeffs } p \leq \Downarrow (\text{unsorted-poly-rel-with0}) (\text{RETURN } p') \rangle$
 $\langle \text{proof} \rangle$

lemma *sort-poly-spec-id'*:

assumes $\langle (p, p') \in \text{unsorted-poly-rel-with0} \rangle$
shows $\langle \text{sort-poly-spec } p \leq \Downarrow (\text{sorted-repeat-poly-rel-with0}) (\text{RETURN } p') \rangle$
 $\langle \text{proof} \rangle$

fun *merge-coeffs0* :: $\langle \text{llist-polynom} \Rightarrow \text{llist-polynom} \rangle$ **where**

$\langle \text{merge-coeffs0 } [] = [] \rangle \mid$
 $\langle \text{merge-coeffs0 } [(xs, n)] = (\text{if } n = 0 \text{ then } [] \text{ else } [(xs, n)]) \rangle \mid$
 $\langle \text{merge-coeffs0 } ((xs, n) \# (ys, m) \# p) =$
 $(\text{if } xs = ys$
 $\text{then if } n + m \neq 0 \text{ then merge-coeffs0 } ((xs, n + m) \# p) \text{ else merge-coeffs0 } p$
 $\text{else if } n = 0 \text{ then merge-coeffs0 } ((ys, m) \# p)$
 $\text{else } (xs, n) \# \text{merge-coeffs0 } ((ys, m) \# p)) \rangle$

lemma *sorted-repeat-poly-list-rel-with0-wrt-ConsD*:

$\langle ((ys, n) \# p, a) \in \text{sorted-repeat-poly-list-rel-with0-wrt } S \text{ term-poly-list-rel} \implies$
 $(p, \text{remove1-mset } (\text{mset } ys, n) a) \in \text{sorted-repeat-poly-list-rel-with0-wrt } S \text{ term-poly-list-rel} \wedge$
 $(\text{mset } ys, n) \in \# a \wedge (\forall x \in \text{set } p. S \text{ } ys \text{ } (\text{fst } x)) \wedge \text{sorted-wrt } (\text{rel2p var-order-rel}) \text{ } ys \wedge$
 $\text{distinct } ys \rangle$
 $\langle \text{proof} \rangle$

lemma *sorted-repeat-poly-list-rel-with0-wrtl-Cons-iff*:

$\langle ((ys, n) \# p, a) \in \text{sorted-repeat-poly-list-rel-with0-wrt } S \text{ term-poly-list-rel} \longleftrightarrow$
 $(p, \text{remove1-mset } (\text{mset } ys, n) a) \in \text{sorted-repeat-poly-list-rel-with0-wrt } S \text{ term-poly-list-rel} \wedge$
 $(\text{mset } ys, n) \in \# a \wedge (\forall x \in \text{set } p. S \text{ } ys \text{ } (\text{fst } x)) \wedge \text{sorted-wrt } (\text{rel2p var-order-rel}) \text{ } ys \wedge$
 $\text{distinct } ys \rangle$
 $\langle \text{proof} \rangle$

lemma *fst-normalize0-polynom-subsetD*:

$\langle (a, b) \in \text{set } (\text{merge-coeffs0 } p) \implies a \in \text{mononoms } p \rangle$
 $\langle \text{proof} \rangle$

lemma *in-set-merge-coeffs0D*:

$\langle (a, b) \in \text{set } (\text{merge-coeffs0 } p) \implies \exists b. (a, b) \in \text{set } p \rangle$
 $\langle \text{proof} \rangle$

lemma *merge-coeffs0-is-normalize-poly-p*:

$\langle (xs, ys) \in \text{sorted-repeat-poly-rel-with0} \implies \exists r. (\text{merge-coeffs0 } xs, r) \in \text{sorted-poly-rel} \wedge \text{normalize-poly-p}^{**}$

$ys\ r\rangle$
 $\langle proof \rangle$

definition *full-normalize-poly* **where**

$\langle full-normalize-poly\ p = do \{$
 $\quad p \leftarrow sort-all-coeffs\ p;$
 $\quad p \leftarrow sort-poly-spec\ p;$
 $\quad RETURN\ (merge-coeffs0\ p)$
 $\}\rangle$

fun *sorted-remdups* **where**

$\langle sorted-remdups\ (x \# y \# zs) =$
 $\quad (if\ x = y\ then\ sorted-remdups\ (y \# zs)\ else\ x \# sorted-remdups\ (y \# zs)) \mid$
 $\langle sorted-remdups\ zs = zs \rangle$

lemma *set-sorted-remdups[simp]*:

$\langle set\ (sorted-remdups\ xs) = set\ xs \rangle$
 $\langle proof \rangle$

lemma *distinct-sorted-remdups*:

$\langle sorted-wrt\ R\ xs \implies transp\ R \implies Restricted-Predicates.total-on\ R\ UNIV \implies$
 $\quad antisym\ R \implies distinct\ (sorted-remdups\ xs) \rangle$
 $\langle proof \rangle$

lemma *full-normalize-poly-normalize-poly-p*:

assumes $\langle (p, p') \in fully-unsorted-poly-rel \rangle$
shows $\langle full-normalize-poly\ p \leq \Downarrow (sorted-poly-rel)\ (SPEC\ (\lambda r. normalize-poly-p^{**}\ p'\ r)) \rangle$
(is $\langle ?A \leq \Downarrow\ ?R\ ?B \rangle$
 $\langle proof \rangle$

definition *mult-poly-full* $:: \langle \cdot \rangle$ **where**

$\langle mult-poly-full\ p\ q = do \{$
 $\quad let\ pq = mult-poly-raw\ p\ q;$
 $\quad normalize-poly\ pq$
 $\}\rangle$

lemma *normalize-poly-normalize-poly-p*:

assumes $\langle (p, p') \in unsorted-poly-rel \rangle$
shows $\langle normalize-poly\ p \leq \Downarrow (sorted-poly-rel)\ (SPEC\ (\lambda r. normalize-poly-p^{**}\ p'\ r)) \rangle$
 $\langle proof \rangle$

9.4 Multiplication and normalisation

definition *mult-poly-p'* $:: \langle \cdot \rangle$ **where**

$\langle mult-poly-p'\ p'\ q' = do \{$
 $\quad pq \leftarrow SPEC(\lambda r. (mult-poly-p\ q')^{**}\ (p', \{\#\})\ (\{\#\}, r));$
 $\quad SPEC\ (\lambda r. normalize-poly-p^{**}\ pq\ r)$
 $\}\rangle$

lemma *unsorted-poly-rel-fully-unsorted-poly-rel*:

$\langle unsorted-poly-rel \subseteq fully-unsorted-poly-rel \rangle$
 $\langle proof \rangle$

lemma *mult-poly-full-mult-poly-p'*:

assumes $\langle (p, p') \in sorted-poly-rel \rangle\ \langle (q, q') \in sorted-poly-rel \rangle$
shows $\langle mult-poly-full\ p\ q \leq \Downarrow (sorted-poly-rel)\ (mult-poly-p'\ p'\ q') \rangle$

⟨proof⟩

definition *add-poly-spec* :: ⟨ \rightarrow ⟩ **where**

⟨*add-poly-spec* $p\ q = SPEC\ (\lambda r. p + q - r \in ideal\ polynom\ bool)$ ⟩

definition *add-poly-p'* :: ⟨ \rightarrow ⟩ **where**

⟨*add-poly-p'* $p\ q = SPEC(\lambda r. add-poly-p^{**}\ (p, q, \{\#\})\ (\{\#\}, \{\#\}, r))$ ⟩

lemma *add-poly-l-add-poly-p'*:

assumes ⟨ $(p, p') \in sorted-poly-rel$ ⟩ ⟨ $(q, q') \in sorted-poly-rel$ ⟩

shows ⟨*add-poly-l* $(p, q) \leq \Downarrow (sorted-poly-rel)\ (add-poly-p'\ p'\ q')$ ⟩

⟨proof⟩

9.5 Correctness

context *poly-embed*

begin

definition *mset-poly-rel* **where**

⟨*mset-poly-rel* = $\{(a, b). b = polynom-of-mset\ a\}$ ⟩

definition *var-rel* **where**

⟨*var-rel* = $br\ \varphi\ (\lambda-. True)$ ⟩

lemma *normalize-poly-p-normalize-poly-spec*:

⟨ $(p, p') \in mset-poly-rel \implies$

$SPEC\ (\lambda r. normalize-poly-p^{**}\ p\ r) \leq \Downarrow mset-poly-rel\ (normalize-poly-spec\ p')$ ⟩

⟨proof⟩

lemma *mult-poly-p'-mult-poly-spec*:

⟨ $(p, p') \in mset-poly-rel \implies (q, q') \in mset-poly-rel \implies$

$mult-poly-p'\ p\ q \leq \Downarrow mset-poly-rel\ (mult-poly-spec\ p'\ q')$ ⟩

⟨proof⟩

lemma *add-poly-p'-add-poly-spec*:

⟨ $(p, p') \in mset-poly-rel \implies (q, q') \in mset-poly-rel \implies$

$add-poly-p'\ p\ q \leq \Downarrow mset-poly-rel\ (add-poly-spec\ p'\ q')$ ⟩

⟨proof⟩

end

definition *weak-equality-l* :: ⟨ $l\ list-polynom \Rightarrow l\ list-polynom \Rightarrow bool\ nres$ ⟩ **where**

⟨*weak-equality-l* $p\ q = RETURN\ (p = q)$ ⟩

definition *weak-equality* :: ⟨ $int\ mpoly \Rightarrow int\ mpoly \Rightarrow bool\ nres$ ⟩ **where**

⟨*weak-equality* $p\ q = SPEC\ (\lambda r. r \longrightarrow p = q)$ ⟩

definition *weak-equality-spec* :: ⟨ $mset-polynom \Rightarrow mset-polynom \Rightarrow bool\ nres$ ⟩ **where**

⟨*weak-equality-spec* $p\ q = SPEC\ (\lambda r. r \longrightarrow p = q)$ ⟩

lemma *term-poly-list-rel-same-rightD*:

⟨ $(a, aa) \in term-poly-list-rel \implies (a, ab) \in term-poly-list-rel \implies aa = ab$ ⟩

⟨proof⟩

lemma *list-rel-term-poly-list-rel-same-rightD*:
 $\langle (xa, y) \in \langle \text{term-poly-list-rel} \times_r \text{int-rel} \rangle \text{list-rel} \implies$
 $(xa, ya) \in \langle \text{term-poly-list-rel} \times_r \text{int-rel} \rangle \text{list-rel} \implies$
 $y = ya \rangle$
 $\langle \text{proof} \rangle$

lemma *weak-equality-l-weak-equality-spec*:
 $\langle (\text{uncurry weak-equality-l}, \text{uncurry weak-equality-spec}) \in$
 $\text{sorted-poly-rel} \times_r \text{sorted-poly-rel} \rightarrow_f \langle \text{bool-rel} \rangle \text{nres-rel} \rangle$
 $\langle \text{proof} \rangle$

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**
 $\langle \text{merge-cstatus } (\text{CFAILED } a) - = \text{CFAILED } a \rangle |$
 $\langle \text{merge-cstatus } - (\text{CFAILED } a) = \text{CFAILED } a \rangle |$
 $\langle \text{merge-cstatus } \text{CFOUND} - = \text{CFOUND} \rangle |$
 $\langle \text{merge-cstatus } - \text{CFOUND} = \text{CFOUND} \rangle |$
 $\langle \text{merge-cstatus } - - = \text{CSUCCESS} \rangle$

definition *code-status-status-rel* :: $\langle ('a \text{ code-status} \times \text{status}) \text{ set} \rangle$ **where**
 $\langle \text{code-status-status-rel} =$
 $\{(\text{CFOUND}, \text{FOUND}), (\text{CSUCCESS}, \text{SUCCESS})\} \cup$
 $\{(\text{CFAILED } a, \text{FAILED}) \mid a. \text{True}\} \rangle$

lemma *in-code-status-status-rel-iff[simp]*:
 $\langle (\text{CFOUND}, b) \in \text{code-status-status-rel} \longleftrightarrow b = \text{FOUND} \rangle$
 $\langle (a, \text{FOUND}) \in \text{code-status-status-rel} \longleftrightarrow a = \text{CFOUND} \rangle$

$\langle (CSUCCESS, b) \in \text{code-status-status-rel} \longleftrightarrow b = SUCCESS \rangle$
 $\langle (a, SUCCESS) \in \text{code-status-status-rel} \longleftrightarrow a = CSUCCESS \rangle$
 $\langle (a, FAILED) \in \text{code-status-status-rel} \longleftrightarrow \text{is-cfailed } a \rangle$
 $\langle (CFAILED\ C, b) \in \text{code-status-status-rel} \longleftrightarrow b = FAILED \rangle$
 $\langle \text{proof} \rangle$

Refinement relation **fun** *pac-step-rel-raw* :: $\langle ('olbl \times 'lbl) \text{ set} \Rightarrow ('a \times 'b) \text{ set} \Rightarrow ('c \times 'd) \text{ set} \Rightarrow$
 $('a, 'c, 'olbl) \text{ pac-step} \Rightarrow ('b, 'd, 'lbl) \text{ pac-step} \Rightarrow \text{bool} \rangle$ **where**
 $\langle \text{pac-step-rel-raw } R1\ R2\ R3\ (\text{Add } p1\ p2\ i\ r)\ (\text{Add } p1'\ p2'\ i'\ r') \longleftrightarrow$
 $(p1, p1') \in R1 \wedge (p2, p2') \in R2 \wedge (i, i') \in R1 \wedge$
 $(r, r') \in R2 \rangle \mid$
 $\langle \text{pac-step-rel-raw } R1\ R2\ R3\ (\text{Mult } p1\ p2\ i\ r)\ (\text{Mult } p1'\ p2'\ i'\ r') \longleftrightarrow$
 $(p1, p1') \in R1 \wedge (p2, p2') \in R2 \wedge (i, i') \in R1 \wedge$
 $(r, r') \in R2 \rangle \mid$
 $\langle \text{pac-step-rel-raw } R1\ R2\ R3\ (\text{Del } p1)\ (\text{Del } p1') \longleftrightarrow$
 $(p1, p1') \in R1 \rangle \mid$
 $\langle \text{pac-step-rel-raw } R1\ R2\ R3\ (\text{Extension } i\ x\ p1)\ (\text{Extension } j\ x'\ p1') \longleftrightarrow$
 $(i, j) \in R1 \wedge (x, x') \in R3 \wedge (p1, p1') \in R2 \rangle \mid$
 $\langle \text{pac-step-rel-raw } R1\ R2\ R3\ - \longleftrightarrow \text{False} \rangle$

fun *pac-step-rel-assn* :: $\langle ('olbl \Rightarrow 'lbl \Rightarrow \text{assn}) \Rightarrow ('a \Rightarrow 'b \Rightarrow \text{assn}) \Rightarrow ('c \Rightarrow 'd \Rightarrow \text{assn}) \Rightarrow ('a, 'c, 'olbl)$
 $\text{pac-step} \Rightarrow ('b, 'd, 'lbl) \text{ pac-step} \Rightarrow \text{assn} \rangle$ **where**
 $\langle \text{pac-step-rel-assn } R1\ R2\ R3\ (\text{Add } p1\ p2\ i\ r)\ (\text{Add } p1'\ p2'\ i'\ r') =$
 $R1\ p1\ p1' * R1\ p2\ p2' * R1\ i\ i' * R2\ r\ r' \rangle \mid$
 $\langle \text{pac-step-rel-assn } R1\ R2\ R3\ (\text{Mult } p1\ p2\ i\ r)\ (\text{Mult } p1'\ p2'\ i'\ r') =$
 $R1\ p1\ p1' * R2\ p2\ p2' * R1\ i\ i' * R2\ r\ r' \rangle \mid$
 $\langle \text{pac-step-rel-assn } R1\ R2\ R3\ (\text{Del } p1)\ (\text{Del } p1') =$
 $R1\ p1\ p1' \rangle \mid$
 $\langle \text{pac-step-rel-assn } R1\ R2\ R3\ (\text{Extension } i\ x\ p1)\ (\text{Extension } i'\ x'\ p1') =$
 $R1\ i\ i' * R3\ x\ x' * R2\ p1\ p1' \rangle \mid$
 $\langle \text{pac-step-rel-assn } R1\ R2\ - - = \text{false} \rangle$

lemma *pac-step-rel-assn-alt-def*:

$\langle \text{pac-step-rel-assn } R1\ R2\ R3\ x\ y = ($
 $\text{case } (x, y) \text{ of}$
 $(\text{Add } p1\ p2\ i\ r, \text{Add } p1'\ p2'\ i'\ r') \Rightarrow$
 $R1\ p1\ p1' * R1\ p2\ p2' * R1\ i\ i' * R2\ r\ r'$
 $\mid (\text{Mult } p1\ p2\ i\ r, \text{Mult } p1'\ p2'\ i'\ r') \Rightarrow$
 $R1\ p1\ p1' * R2\ p2\ p2' * R1\ i\ i' * R2\ r\ r'$
 $\mid (\text{Del } p1, \text{Del } p1') \Rightarrow R1\ p1\ p1'$
 $\mid (\text{Extension } i\ x\ p1, \text{Extension } i'\ x'\ p1') \Rightarrow R1\ i\ i' * R3\ x\ x' * R2\ p1\ p1'$
 $\mid - \Rightarrow \text{false}$
 \rangle
 $\langle \text{proof} \rangle$

Addition checking **definition** *error-msg* **where**

$\langle \text{error-msg } i\ msg = CFAILED\ ("s\ CHECKING\ failed\ at\ line\ " @ show\ i\ @ " with\ error\ " @ msg) \rangle$

definition *error-msg-notin-dom-err* **where**

$\langle \text{error-msg-notin-dom-err} = "notin\ domain" \rangle$

definition *error-msg-notin-dom* :: $\langle \text{nat} \Rightarrow \text{string} \rangle$ **where**

$\langle \text{error-msg-notin-dom } i = show\ i\ @\ \text{error-msg-notin-dom-err} \rangle$

definition *error-msg-reused-dom* **where**

$\langle \text{error-msg-reused-dom } i = \text{show } i @ \text{ " already in domain" } \rangle$

definition *error-msg-not-equal-dom* **where**

$\langle \text{error-msg-not-equal-dom } p \ q \ pq \ r = \text{show } p @ \text{ " } + \text{ " } @ \text{show } q @ \text{ " } = \text{ " } @ \text{show } pq @ \text{ " not equal" } @ \text{show } r \rangle$

definition *check-not-equal-dom-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**

$\langle \text{check-not-equal-dom-err } p \ q \ pq \ r = \text{SPEC } (\lambda -. \text{True}) \rangle$

definition *vars-l-list* :: $\langle \text{l-list-polynom} \Rightarrow \text{string set} \rangle$ **where**

$\langle \text{vars-l-list } xs = \bigcup (\text{set 'fst 'set } xs) \rangle$

definition *check-addition-l* :: $\langle - \Rightarrow - \Rightarrow \text{string set} \Rightarrow \text{nat} \Rightarrow \text{nat} \Rightarrow \text{nat} \Rightarrow \text{l-list-polynom} \Rightarrow \text{string code-status nres} \rangle$ **where**

$\langle \text{check-addition-l spec } A \ \forall \ p \ q \ i \ r = \text{do } \{$
 $\quad \text{let } b = p \in \# \text{ dom-m } A \wedge q \in \# \text{ dom-m } A \wedge i \notin \# \text{ dom-m } A \wedge \text{vars-l-list } r \subseteq \mathcal{V};$
 $\quad \text{if } \neg b$
 $\quad \text{then RETURN } (\text{error-msg } i \ ((\text{if } p \notin \# \text{ dom-m } A \text{ then error-msg-notin-dom } p \text{ else } []) @ (\text{if } q \notin \#$
 $\text{dom-m } A \text{ then error-msg-notin-dom } p \text{ else } [])) @$
 $\quad (\text{if } i \in \# \text{ dom-m } A \text{ then error-msg-reused-dom } p \text{ else } []))$
 $\quad \text{else do } \{$
 $\quad \quad \text{ASSERT } (p \in \# \text{ dom-m } A);$
 $\quad \quad \text{let } p = \text{the } (\text{fmlookup } A \ p);$
 $\quad \quad \text{ASSERT } (q \in \# \text{ dom-m } A);$
 $\quad \quad \text{let } q = \text{the } (\text{fmlookup } A \ q);$
 $\quad \quad pq \leftarrow \text{add-poly-l } (p, q);$
 $\quad \quad b \leftarrow \text{weak-equality-l } pq \ r;$
 $\quad \quad b' \leftarrow \text{weak-equality-l } r \ \text{spec};$
 $\quad \quad \text{if } b \text{ then } (\text{if } b' \text{ then RETURN CFOUND else RETURN CSUCCESS})$
 $\quad \quad \text{else do } \{$
 $\quad \quad \quad c \leftarrow \text{check-not-equal-dom-err } p \ q \ pq \ r;$
 $\quad \quad \quad \text{RETURN } (\text{error-msg } i \ c)$
 $\quad \quad \}$
 $\quad \}$
 \rangle

Multiplication checking **definition** *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 *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 *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**

$\langle \text{check-mult-l spec } A \ \forall \ p \ q \ i \ r = \text{do } \{$


```

let b = p ∈# dom-m A ∧ i ∉# dom-m A ∧ vars-llist q ⊆ V ∧ vars-llist r ⊆ V;
if ¬b
then do {
  c ← check-mult-l-dom-err (p ∉# dom-m A) p (i ∈# dom-m A) i;
  RETURN (error-msg i c)}
else do {
  ASSERT (p ∈# dom-m A);
  let p = the (fmlookup A p);
  pq ← mult-poly-full p q;
  b ← weak-equality-l pq r;
  b' ← weak-equality-l r spec;
  if b then (if b' then RETURN CFOUND else RETURN CSUCCESS) else do {
    c ← check-mult-l-mult-err p q pq r;
    RETURN (error-msg i c)}
  }
}
}
}

```

Deletion checking definition *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 *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 *check-extension-l-no-new-var-err* :: $\langle \text{l-list-polynom} \Rightarrow \text{string nres} \rangle$ where
 $\langle \text{check-extension-l-no-new-var-err } p = \text{SPEC } (\lambda -. \text{True}) \rangle$

definition *check-extension-l-new-var-multiple-err* :: $\langle \text{string} \Rightarrow \text{l-list-polynom} \Rightarrow \text{string nres} \rangle$ where
 $\langle \text{check-extension-l-new-var-multiple-err } v \ p = \text{SPEC } (\lambda -. \text{True}) \rangle$

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
 $\langle \text{check-extension-l-side-cond-err } v \ p \ p' \ q = \text{SPEC } (\lambda -. \text{True}) \rangle$

definition *check-extension-l*
 :: $\langle - \Rightarrow - \Rightarrow \text{string set} \Rightarrow \text{nat} \Rightarrow \text{string} \Rightarrow \text{l-list-polynom} \Rightarrow (\text{string code-status}) \text{ nres} \rangle$
where

```

⟨check-extension-l spec A V i v p = do {
  let b = i ∉# dom-m A ∧ v ∉ V ∧ ([v], -1) ∈ set p;
  if ¬b
  then do {
    c ← check-extension-l-dom-err i;
    RETURN (error-msg i c)}
  } else do {
    let p' = remove1 ([v], -1) p;
    let b = vars-llist p' ⊆ V;
    if ¬b
    then do {
      c ← check-extension-l-new-var-multiple-err v p';
      RETURN (error-msg i c)}
    }
  } else do {
    p2 ← mult-poly-full p' p';

```



```

    }
  } (is (· ≥ ?A))
⟨proof⟩

```

lemma *check-mult-alt-def*:

```

⟨check-mult A V p q i r ≥
  do {
    b ← SPEC(λb. b → p ∈# dom-m A ∧ i ∉# dom-m A ∧ vars q ⊆ V ∧ vars r ⊆ V);
    if ¬b
    then RETURN False
    else do {
      ASSERT (p ∈# dom-m A);
      let p = the (fmlookup A p);
      pq ← mult-poly-spec p q;
      p ← weak-equality pq r;
      RETURN p
    }
  }
⟩
⟨proof⟩

```

primrec *insort-key-rel* :: ('b ⇒ 'b ⇒ bool) ⇒ 'b ⇒ 'b list ⇒ '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))

⟨proof⟩

lemma *sorted-wrt-insort-key-rel*:

```

(total-on R (insert x (set xs)) ⇒ transp R ⇒ reflp R ⇒
  sorted-wrt R xs ⇒ sorted-wrt R (insort-key-rel R x xs))
⟨proof⟩

```

lemma *sorted-wrt-insort-key-rel2*:

```

(total-on R (insert x (set xs)) ⇒ transp R ⇒ x ∉ set xs ⇒
  sorted-wrt R xs ⇒ sorted-wrt R (insort-key-rel R x xs))
⟨proof⟩

```

Step checking definition *PAC-checker-l-step* :: (· ⇒ string code-status × string set × · ⇒ (l-list-polynom, string, nat) pac-step ⇒ ·) **where**

```

⟨PAC-checker-l-step = (λspec (st', V, A) st. case st of
  Add - - - ⇒
    do {
      r ← full-normalize-poly (pac-res st);
      eq ← check-addition-l spec A V (pac-src1 st) (pac-src2 st) (new-id st) r;
      let - = eq;
      if ¬is-cfailed eq
      then RETURN (merge-cstatus st' eq,
        V, fmupd (new-id st) r A)
      else RETURN (eq, V, A)
    }
  | Del - ⇒
    do {
      eq ← 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,
       $\mathcal{V}$ , fmupd (new-id st) r A)
    else RETURN (eq,  $\mathcal{V}$ , A)
  }
| Extension - - -  $\Rightarrow$ 
  do {
    r  $\leftarrow$  full-normalize-poly (([new-var st], -1) # (pac-res st));
    (eq)  $\leftarrow$  check-extension-l spec A  $\mathcal{V}$  (new-id st) (new-var st) r;
    if  $\neg$ is-cfailed eq
    then do {
      RETURN (st',
        insert (new-var st)  $\mathcal{V}$ , fmupd (new-id st) r A)
    }
    else RETURN (eq,  $\mathcal{V}$ , A)
  }
)

```

lemma pac-step-rel-raw-def:

$\langle \langle K, V, R \rangle \text{ pac-step-rel-raw} = \text{pac-step-rel-raw } K \ V \ R \rangle$
 $\langle \text{proof} \rangle$

definition mononoms-equal-up-to-reorder **where**

$\langle \text{mononoms-equal-up-to-reorder } xs \ ys \longleftrightarrow$
 $\text{map } (\lambda(a, b). \text{ (mset } a, b)) \ xs = \text{map } (\lambda(a, b). \text{ (mset } a, b)) \ ys \rangle$

definition normalize-poly-l **where**

$\langle \text{normalize-poly-l } p = \text{SPEC } (\lambda p'.$
 $\text{normalize-poly-p}^* ((\lambda(a, b). \text{ (mset } a, b)) \text{ ' \# mset } p) ((\lambda(a, b). \text{ (mset } a, b)) \text{ ' \# mset } p') \wedge$
 $0 \notin \text{snd ' \# mset } p' \wedge$
 $\text{sorted-wrt } (\text{rel2p } (\text{term-order-rel} \times_r \text{int-rel})) \ p' \wedge$
 $(\forall x \in \text{mononoms } p'. \text{ sorted-wrt } (\text{rel2p } \text{var-order-rel}) \ x) \rangle$

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**
 $\langle \text{remap-polys-l spec} = (\lambda \mathcal{V} \ A. \text{ do} \{$
 $\text{dom} \leftarrow \text{SPEC}(\lambda \text{dom}. \text{ set-mset } (\text{dom-m } A) \subseteq \text{dom} \wedge \text{finite dom});$
 $\text{failed} \leftarrow \text{SPEC}(\lambda-::\text{bool}. \text{ True});$
 if failed
 $\text{then do } \{$

```

  c ← remap-polys-l-dom-err;
  RETURN (error-msg (0 :: nat) c, V, fmempty)
}
else do {
  (b, V, A) ← FOREACH dom
  (λi (b, V, A').
    if i ∈# dom-m A
    then do {
      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, fmupe i p A')
    } else RETURN (b, V, A'))
  (False, V, fmempty);
  RETURN (if b then CFOUND else CSUCCESS, 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**

⟨*pac-step-rel* ≡ *p2rel* ((*Id*, *fully-unsorted-poly-rel* *O* *mset-poly-rel*, *var-rel*) *pac-step-rel-raw*)⟩

abbreviation *fmap-polys-rel* **where**

⟨*fmap-polys-rel* ≡ ⟨*nat-rel*, *sorted-poly-rel* *O* *mset-poly-rel*⟩*fmap-rel*⟩

lemma

⟨*normalize-poly-p* *s0* *s* ⇒
 (*s0*, *p*) ∈ *mset-poly-rel* ⇒
 (*s*, *p*) ∈ *mset-poly-rel*⟩
 ⟨*proof*⟩

lemma *vars-poly-of-vars*:

⟨*vars* (*poly-of-vars* *a* :: *int* *mpoly*) ⊆ (*φ* ‘ *set-mset* *a*)⟩
 ⟨*proof*⟩

lemma *vars-polynom-of-mset*:

⟨*vars* (*polynom-of-mset* *za*) ⊆ ⋃ (*image* *φ* ‘ (*set-mset* *o* *fst*) ‘ *set-mset* *za*)⟩
 ⟨*proof*⟩

lemma *fully-unsorted-poly-rel-vars-subset-vars-llist*:

$\langle (A, B) \in \text{fully-unsorted-poly-rel } O \text{ mset-poly-rel} \implies \text{vars } B \subseteq \varphi \text{ ' vars-llist } A \rangle$
 $\langle \text{proof} \rangle$

lemma *fully-unsorted-poly-rel-extend-vars*:

$\langle (A, B) \in \text{fully-unsorted-poly-rel } O \text{ mset-poly-rel} \implies$
 $(x1c, x1a) \in \langle \text{var-rel} \rangle \text{set-rel} \implies$
 $\text{RETURN } (x1c \cup \text{vars-llist } A)$
 $\leq \Downarrow (\langle \text{var-rel} \rangle \text{set-rel})$
 $(\text{SPEC } ((\subseteq) (x1a \cup \text{vars } (B)))) \rangle$
 $\langle \text{proof} \rangle$

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 $\langle \text{remap-polys-l spec } \mathcal{V} A \leq$

$\Downarrow (\text{code-status-status-rel} \times_r \langle \text{var-rel} \rangle \text{set-rel} \times_r \text{fmap-polys-rel}) (\text{remap-polys spec}' \mathcal{V}' B) \rangle$

$(\text{is } \langle - \leq \Downarrow ?R - \rangle)$

$\langle \text{proof} \rangle$

lemma *fref-to-Down-curry*:

$\langle (\text{uncurry } f, \text{uncurry } g) \in [P]_f A \rightarrow \langle B \rangle \text{nres-rel} \implies$
 $(\bigwedge x x' y y'. P (x', y') \implies ((x, y), (x', y')) \in A \implies f x y \leq \Downarrow B (g x' y')) \rangle$
 $\langle \text{proof} \rangle$

lemma *weak-equality-spec-weak-equality*:

$\langle (p, p') \in \text{mset-poly-rel} \implies$
 $(r, r') \in \text{mset-poly-rel} \implies$
 $\text{weak-equality-spec } p r \leq \text{weak-equality } p' r' \rangle$
 $\langle \text{proof} \rangle$

lemma *weak-equality-l-weak-equality-l'[refine]*:

$\langle \text{weak-equality-l } p q \leq \Downarrow \text{bool-rel } (\text{weak-equality } p' q') \rangle$
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 $p p' q q'$
 $\langle \text{proof} \rangle$

lemma *error-msg-ne-SUCCESS[iff]*:

$\langle \text{error-msg } i m \neq \text{CSUCCESS} \rangle$
 $\langle \text{error-msg } i m \neq \text{CFOUND} \rangle$
 $\langle \text{is-cfailed } (\text{error-msg } i m) \rangle$
 $\langle \neg \text{is-cfound } (\text{error-msg } i m) \rangle$
 $\langle \text{proof} \rangle$

lemma *sorted-poly-rel-vars-llist*:

$\langle (r, r') \in \text{sorted-poly-rel } O \text{ mset-poly-rel} \implies$
 $\text{vars } r' \subseteq \varphi \text{ ' vars-llist } r \rangle$
 $\langle \text{proof} \rangle$

lemma *check-addition-l-check-add:*

assumes $\langle (A, B) \in \text{fmap-polys-rel} \rangle$ **and** $\langle (r, r') \in \text{sorted-poly-rel } O \text{ mset-poly-rel} \rangle$
 $\langle (p, p') \in \text{Id} \rangle \langle (q, q') \in \text{Id} \rangle \langle (i, i') \in \text{nat-rel} \rangle$
 $\langle (\mathcal{V}', \mathcal{V}) \in \langle \text{var-rel} \rangle \text{set-rel} \rangle$

shows

$\langle \text{check-addition-l spec } A \mathcal{V}' p q i r \leq \Downarrow \{ (st, b). (\neg \text{is-cfailed } st \longleftrightarrow b) \wedge$
 $(\text{is-cfound } st \longrightarrow \text{spec} = r) \} (\text{check-add } B \mathcal{V} p' q' i' r') \rangle$

$\langle \text{proof} \rangle$

lemma *check-del-l-check-del:*

$\langle (A, B) \in \text{fmap-polys-rel} \implies (x3, x3a) \in \text{Id} \implies \text{check-del-l spec } A (\text{pac-src1 } (\text{Del } x3))$
 $\leq \Downarrow \{ (st, b). (\neg \text{is-cfailed } st \longleftrightarrow b) \wedge (b \longrightarrow st = \text{CSUCCESS}) \} (\text{check-del } B (\text{pac-src1 } (\text{Del } x3a))) \rangle$

$\langle \text{proof} \rangle$

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

$\langle \text{check-mult-l spec } A \mathcal{V} p q i r \leq \Downarrow \{ (st, b). (\neg \text{is-cfailed } st \longleftrightarrow b) \wedge$
 $(\text{is-cfound } st \longrightarrow \text{spec} = r) \} (\text{check-mult } B \mathcal{V}' p' q' i' r') \rangle$

$\langle \text{proof} \rangle$

lemma *normalize-poly-normalize-poly-spec:*

assumes $\langle (r, t) \in \text{unsorted-poly-rel } O \text{ mset-poly-rel} \rangle$

shows

$\langle \text{normalize-poly } r \leq \Downarrow (\text{sorted-poly-rel } O \text{ mset-poly-rel}) (\text{normalize-poly-spec } t) \rangle$

$\langle \text{proof} \rangle$

lemma *remove1-list-rel:*

$\langle (xs, ys) \in \langle R \rangle \text{list-rel} \implies$
 $(a, b) \in R \implies$
 $\text{IS-RIGHT-UNIQUE } R \implies$
 $\text{IS-LEFT-UNIQUE } R \implies$
 $(\text{remove1 } a \text{ } xs, \text{remove1 } b \text{ } ys) \in \langle R \rangle \text{list-rel} \rangle$
 $\langle \text{proof} \rangle$

lemma *remove1-list-rel2:*

$\langle (xs, ys) \in \langle R \rangle \text{list-rel} \implies$
 $(a, b) \in R \implies$
 $(\bigwedge c. (a, c) \in R \implies c = b) \implies$
 $(\bigwedge c. (c, b) \in R \implies c = a) \implies$
 $(\text{remove1 } a \text{ } xs, \text{remove1 } b \text{ } ys) \in \langle R \rangle \text{list-rel} \rangle$
 $\langle \text{proof} \rangle$

lemma *remove1-sorted-poly-rel-mset-poly-rel:*

assumes

$\langle (r, r') \in \text{sorted-poly-rel } O \text{ mset-poly-rel} \rangle$ **and**
 $\langle ([a], 1) \in \text{set } r \rangle$

shows

$\langle (\text{remove1 } ([a], 1) \text{ } r, r' - \text{Var } (\varphi \text{ } a))$
 $\in \text{sorted-poly-rel } O \text{ mset-poly-rel} \rangle$

$\langle \text{proof} \rangle$

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

$\langle (\text{remove1 } ([a], -1) \ r, r' + \text{Var } (\varphi \ a)) \in \text{sorted-poly-rel } O \text{ mset-poly-rel} \rangle$

$\langle \text{proof} \rangle$

lemma *insert-var-rel-set-rel:*

$\langle (\mathcal{V}, \mathcal{V}') \in \langle \text{var-rel} \rangle \text{set-rel} \implies$

$(yb, x2) \in \text{var-rel} \implies$

$\langle \text{insert } yb \ \mathcal{V}, \text{insert } x2 \ \mathcal{V}' \rangle \in \langle \text{var-rel} \rangle \text{set-rel} \rangle$

$\langle \text{proof} \rangle$

lemma *var-rel-set-rel-iff:*

$\langle (\mathcal{V}, \mathcal{V}') \in \langle \text{var-rel} \rangle \text{set-rel} \implies$

$(yb, x2) \in \text{var-rel} \implies$

$yb \in \mathcal{V} \longleftrightarrow x2 \in \mathcal{V}' \rangle$

$\langle \text{proof} \rangle$

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

$\langle \text{check-extension-l spec } A \ \mathcal{V} \ i \ x \ r \leq$

$\Downarrow \{((st), (b)).$

$(\neg \text{is-cfailed } st \longleftrightarrow b) \wedge$

$(\text{is-cfound } st \longrightarrow \text{spec} = r) \rangle \langle \text{check-extension } B \ \mathcal{V}' \ i' \ x' \ r' \rangle$

$\langle \text{proof} \rangle$

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

$\langle \text{full-normalize-poly } p$

$\leq \Downarrow (\text{sorted-poly-rel } O \text{ mset-poly-rel})$

$\langle \text{normalize-poly-spec } p' \rangle$

$\langle \text{proof} \rangle$

lemma *insort-key-rel-decomp:*

$\langle \exists \text{ys zs. } xs = \text{ys} @ \text{zs} \wedge \text{insort-key-rel } R \ x \ xs = \text{ys} @ x \ \# \ \text{zs} \rangle$

$\langle \text{proof} \rangle$

lemma *list-rel-append-same-length:*

$\langle \text{length } xs = \text{length } xs' \implies (xs @ \text{ys}, xs' @ \text{ys}') \in \langle R \rangle \text{list-rel} \longleftrightarrow (xs, xs') \in \langle R \rangle \text{list-rel} \wedge (\text{ys}, \text{ys}') \in \langle R \rangle \text{list-rel} \rangle$

$\langle \text{proof} \rangle$

lemma *term-poly-list-rel-list-relD:* $\langle (\text{ys}, \text{cs}) \in \langle \text{term-poly-list-rel} \times_r \text{int-rel} \rangle \text{list-rel} \implies$

$\text{cs} = \text{map } (\lambda(a, y). (\text{mset } a, y)) \ \text{ys} \rangle$

$\langle \text{proof} \rangle$

lemma *term-poly-list-rel-single*: $\langle [x32], \{\#x32\# \} \rangle \in \text{term-poly-list-rel}$
 $\langle \text{proof} \rangle$

lemma *unsorted-poly-rel-list-rel-list-rel-uminus*:

$\langle \text{map } (\lambda(a, b). (a, - b)) r, yc \rangle$
 $\in \langle \text{unsorted-term-poly-list-rel} \times_r \text{int-rel} \rangle \text{list-rel} \implies$
 $\langle r, \text{map } (\lambda(a, b). (a, - b)) yc \rangle$
 $\in \langle \text{unsorted-term-poly-list-rel} \times_r \text{int-rel} \rangle \text{list-rel}$
 $\langle \text{proof} \rangle$

lemma *mset-poly-rel-minus*: $\langle \{\#(a, b)\# \}, v' \rangle \in \text{mset-poly-rel} \implies$
 $\langle \text{mset } yc, r' \rangle \in \text{mset-poly-rel} \implies$
 $\langle r, yc \rangle$
 $\in \langle \text{unsorted-term-poly-list-rel} \times_r \text{int-rel} \rangle \text{list-rel} \implies$
 $\langle \text{add-mset } (a, b) (\text{mset } yc),$
 $v' + r' \rangle$
 $\in \text{mset-poly-rel}$
 $\langle \text{proof} \rangle$

lemma *fully-unsorted-poly-rel-diff*:

$\langle [v], v' \rangle \in \text{fully-unsorted-poly-rel} \ O \ \text{mset-poly-rel} \implies$
 $\langle r, r' \rangle \in \text{fully-unsorted-poly-rel} \ O \ \text{mset-poly-rel} \implies$
 $\langle v \# r,$
 $v' + r' \rangle$
 $\in \text{fully-unsorted-poly-rel} \ O \ \text{mset-poly-rel}$
 $\langle \text{proof} \rangle$

lemma *PAC-checker-l-step-PAC-checker-step*:

assumes

$\langle \text{Ast}, \text{Bst} \rangle \in \text{code-status-status-rel} \times_r \langle \text{var-rel} \rangle \text{set-rel} \times_r \text{fmap-polys-rel}$ **and**
 $\langle \text{st}, \text{st}' \rangle \in \text{pac-step-rel}$ **and**
 $\text{spec}: \langle \text{spec}, \text{spec}' \rangle \in \text{sorted-poly-rel} \ O \ \text{mset-poly-rel}$

shows

$\langle \text{PAC-checker-l-step spec Ast st} \leq \Downarrow (\text{code-status-status-rel} \times_r \langle \text{var-rel} \rangle \text{set-rel} \times_r \text{fmap-polys-rel})$
 $(\text{PAC-checker-step spec}' \text{Bst st}') \rangle$
 $\langle \text{proof} \rangle$

lemma *code-status-status-rel-discrim-iff*:

$\langle x1a, x1c \rangle \in \text{code-status-status-rel} \implies \text{is-cfailed } x1a \longleftrightarrow \text{is-failed } x1c$
 $\langle x1a, x1c \rangle \in \text{code-status-status-rel} \implies \text{is-cfound } x1a \longleftrightarrow \text{is-found } x1c$
 $\langle \text{proof} \rangle$

lemma *PAC-checker-l-PAC-checker*:

assumes

$\langle A, B \rangle \in \langle \text{var-rel} \rangle \text{set-rel} \times_r \text{fmap-polys-rel}$ **and**
 $\langle \text{st}, \text{st}' \rangle \in \langle \text{pac-step-rel} \rangle \text{list-rel}$ **and**
 $\langle \text{spec}, \text{spec}' \rangle \in \text{sorted-poly-rel} \ O \ \text{mset-poly-rel}$ **and**
 $\langle b, b' \rangle \in \text{code-status-status-rel}$

shows

$\langle \text{PAC-checker-l spec A b st} \leq \Downarrow (\text{code-status-status-rel} \times_r \langle \text{var-rel} \rangle \text{set-rel} \times_r \text{fmap-polys-rel})$
 $(\text{PAC-checker spec}' B b' \text{st}') \rangle$
 $\langle \text{proof} \rangle$

end

lemma *less-than-char-of-char*[code-unfold]:
 $\langle (x, y) \in \text{less-than-char} \longleftrightarrow (\text{of-char } x :: \text{nat}) < \text{of-char } y \rangle$
 $\langle \text{proof} \rangle$

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*
 $:: \langle \text{l-list-polynom} \Rightarrow (\text{nat}, \text{l-list-polynom}) \text{ fmap} \Rightarrow (-, \text{string}, \text{nat}) \text{ pac-step list} \Rightarrow$
 $(\text{string code-status} \times -) \text{ nres} \rangle$

where

$\langle \text{full-checker-l spec } A \text{ st} = \text{do} \{$
 $\text{spec}' \leftarrow \text{full-normalize-poly spec};$
 $(b, \mathcal{V}, A) \leftarrow \text{remap-polys-l spec}' \{ \} A;$
 $\text{if is-cfailed } b$
 $\text{then RETURN } (b, \mathcal{V}, A)$
 $\text{else do} \{$
 $\text{let } \mathcal{V} = \mathcal{V} \cup \text{vars-l-list spec};$
 $\text{PAC-checker-l spec}' (\mathcal{V}, A) b \text{ st}$
 $\}$
 $\}$

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**

$\langle \text{unsorted-fmap-polys-rel} \equiv \langle \text{nat-rel}, \text{fully-unsorted-poly-rel } O \text{ mset-poly-rel} \rangle \text{ fmap-rel} \rangle$

lemma *full-checker-l-full-checker*:

assumes

$\langle (A, B) \in \text{unsorted-fmap-polys-rel} \rangle$ **and**
 $\langle (st, st') \in \langle \text{pac-step-rel} \rangle \text{list-rel} \rangle$ **and**
 $\langle (spec, spec') \in \text{fully-unsorted-poly-rel } O \text{ mset-poly-rel} \rangle$

shows

$\langle \text{full-checker-l spec } A \text{ st} \leq \Downarrow (\text{code-status-status-rel} \times_r \langle \text{var-rel} \rangle \text{set-rel} \times_r \text{fmap-polys-rel}) (\text{full-checker}$
 $\text{spec}' B \text{ st}') \rangle$
 $\langle \text{proof} \rangle$

lemma *full-checker-l-full-checker'*:

$\langle (\text{uncurry2 full-checker-l}, \text{uncurry2 full-checker}) \in$
 $((\text{fully-unsorted-poly-rel } O \text{ mset-poly-rel}) \times_r \text{unsorted-fmap-polys-rel}) \times_r \langle \text{pac-step-rel} \rangle \text{list-rel} \rightarrow_f$
 $\langle (\text{code-status-status-rel} \times_r \langle \text{var-rel} \rangle \text{set-rel} \times_r \text{fmap-polys-rel}) \rangle \text{nres-rel} \rangle$
 $\langle \text{proof} \rangle$

end

definition *remap-polys-l2* :: $\langle \text{l-list-polynom} \Rightarrow \text{string set} \Rightarrow (\text{nat}, \text{l-list-polynom}) \text{ fmap} \Rightarrow - \text{ nres} \rangle$ **where**
 $\langle \text{remap-polys-l2 spec} = (\lambda \mathcal{V} A. \text{do}\{$
 $n \leftarrow \text{upper-bound-on-dom } A;$
 $b \leftarrow \text{RETURN } (n \geq 2^{64});$
 $\text{if } b$
 $\text{then do } \{$
 $c \leftarrow \text{remap-polys-l-dom-err};$
 $\text{RETURN } (\text{error-msg } (0 :: \text{nat}) \ c, \ \mathcal{V}, \ \text{fmempty})$
 $\}$
 $\text{else do } \{$
 $(b, \ \mathcal{V}, \ A) \leftarrow \text{nfoldli } ([0..<n]) \ (\lambda -. \ \text{True})$
 $(\lambda i \ (b, \ \mathcal{V}, \ A').$
 $\text{if } i \in \# \ \text{dom-m } A$
 $\text{then do } \{$
 $\text{ASSERT}(\text{fmlookup } A \ i \neq \text{None});$
 $p \leftarrow \text{full-normalize-poly } (\text{the } (\text{fmlookup } A \ i));$
 $eq \leftarrow \text{weak-equality-l } p \ \text{spec};$
 $\mathcal{V} \leftarrow \text{RETURN } (\mathcal{V} \cup \text{vars-l-list } (\text{the } (\text{fmlookup } A \ i)));$
 $\text{RETURN}(b \vee eq, \ \mathcal{V}, \ \text{fmupd } i \ p \ A')$
 $\}$ $\text{else RETURN } (b, \ \mathcal{V}, \ A')$
 $\}$
 $\text{RETURN } (\text{if } b \text{ then } \text{CFOUND} \text{ else } \text{CSUCCESS}, \ \mathcal{V}, \ A)$
 $\}$
 $\rangle\rangle$

lemma *remap-polys-l2-remap-polys-l*:
 $\langle \text{remap-polys-l2 spec } \mathcal{V} \ A \leq \Downarrow \text{Id } (\text{remap-polys-l spec } \mathcal{V} \ A) \rangle$
 $\langle \text{proof} \rangle$

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**
 $\langle \text{uint64-nat-rel} = \text{br nat-of-uint64 } (\lambda -. \ \text{True}) \rangle$

abbreviation *uint64-nat-assn* **where**
 $\langle \text{uint64-nat-assn} \equiv \text{pure uint64-nat-rel} \rangle$

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

```

  ⟨proof⟩
end

instantiation uint64 :: hashable
begin
definition hashcode-uint64 :: ⟨uint64 ⇒ uint32⟩ where
  ⟨hashcode-uint64 n = (uint32-of-nat (nat-of-uint64 ((n) AND ((2 :: uint64) ^ 32 - 1))))⟩

definition def-hashmap-size-uint64 :: ⟨uint64 itself ⇒ nat⟩ where
  ⟨def-hashmap-size-uint64 = (λ-. 16)⟩
  — same as nat
instance
  ⟨proof⟩
end

lemma word-nat-of-uint64-Rep-inject[simp]: ⟨nat-of-uint64 ai = nat-of-uint64 bi ⟷ ai = bi⟩
  ⟨proof⟩

instance uint64 :: heap
  ⟨proof⟩

instance uint64 :: semiring-numeral
  ⟨proof⟩

lemma nat-of-uint64-012[simp]: ⟨nat-of-uint64 0 = 0⟩ ⟨nat-of-uint64 2 = 2⟩ ⟨nat-of-uint64 1 = 1⟩
  ⟨proof⟩

definition uint64-of-nat-conv where
  [simp]: ⟨uint64-of-nat-conv (x :: nat) = x⟩
lemma less-upper-bintrunc-id: ⟨n < 2 ^ 64 ⟹ n ≥ 0 ⟹ bintrunc b n = n⟩
  ⟨proof⟩

lemma nat-of-uint64-uint64-of-nat-id: ⟨n < 2 ^ 64 ⟹ nat-of-uint64 (uint64-of-nat n) = n⟩
  ⟨proof⟩

lemma [sepref-fr-rules]:
  ⟨(return o uint64-of-nat, RETURN o uint64-of-nat-conv) ∈ [λa. a < 2 ^ 64]_a nat-assnk → uint64-nat-assn⟩
  ⟨proof⟩

definition string-rel :: ⟨(String.literal × string) set⟩ where
  ⟨string-rel = {(x, y). y = String.explode x}⟩

abbreviation string-assn :: ⟨string ⇒ String.literal ⇒ assn⟩ where
  ⟨string-assn ≡ pure string-rel⟩

lemma eq-string-eq:
  ⟨((=), (=)) ∈ string-rel → string-rel → bool-rel⟩
  ⟨proof⟩

lemmas eq-string-eq-hnr =
  eq-string-eq[sepref-import-param]

definition string2-rel :: ⟨(string × string) set⟩ where
  ⟨string2-rel ≡ ⟨Id⟩list-rel⟩

```

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*:
 $\langle \text{poly-assn} = \text{pure poly-rel} \rangle$
 $\langle \text{proof} \rangle$

abbreviation *polys-assn* **where**
 $\langle \text{polys-assn} \equiv \text{hm-fmap-assn uint64-nat-assn poly-assn} \rangle$

lemma *string-rel-string-assn*:
 $\langle (\uparrow ((c, a) \in \text{string-rel})) = \text{string-assn } a \ c \rangle$
 $\langle \text{proof} \rangle$

lemma *single-valued-string-rel*:
 $\langle \text{single-valued string-rel} \rangle$
 $\langle \text{proof} \rangle$

lemma *IS-LEFT-UNIQUE-string-rel*:
 $\langle \text{IS-LEFT-UNIQUE string-rel} \rangle$
 $\langle \text{proof} \rangle$

lemma *IS-RIGHT-UNIQUE-string-rel*:
 $\langle \text{IS-RIGHT-UNIQUE string-rel} \rangle$
 $\langle \text{proof} \rangle$

lemma *single-valued-monom-rel*: $\langle \text{single-valued monom-rel} \rangle$
 $\langle \text{proof} \rangle$

lemma *single-valued-monomial-rel*:
 $\langle \text{single-valued monomial-rel} \rangle$
 $\langle \text{proof} \rangle$

lemma *single-valued-monom-rel'*: $\langle \text{IS-LEFT-UNIQUE monom-rel} \rangle$
 $\langle \text{proof} \rangle$

lemma *single-valued-monomial-rel'*:
 ⟨IS-LEFT-UNIQUE monomial-rel⟩
 ⟨proof⟩

lemma [safe-constraint-rules]:
 ⟨Sepref-Constraints.CONSTRAINT single-valued string-rel⟩
 ⟨Sepref-Constraints.CONSTRAINT IS-LEFT-UNIQUE string-rel⟩
 ⟨proof⟩

lemma *eq-string-monom-hnr*[sepref-fr-rules]:
 ⟨(uncurry (return oo (=)), uncurry (RETURN oo (=))) ∈ monom-assn^k *_a monom-assn^k →_a bool-assn⟩
 ⟨proof⟩

definition *term-order-rel'* **where**
 [simp]: ⟨term-order-rel' x y = ((x, y) ∈ term-order-rel)⟩

lemma *term-order-rel*[def-pat-rules]:
 ⟨(∈)\$(x,y)\$term-order-rel ≡ term-order-rel'\$x\$y⟩
 ⟨proof⟩

lemma *term-order-rel-alt-def*:
 ⟨term-order-rel = lexord (p2rel char.lexordp)⟩
 ⟨proof⟩

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

⟨proof⟩

end

instantiation *list* :: (linorder) linorder

begin

definition *less-list* **where** *less-list* = lexordp (<)

definition *less-eq-list* **where** *less-eq-list* = lexordp-eq

instance

⟨proof⟩

end

lemma *term-order-rel'-alt-def-lexord*:
 ⟨term-order-rel' x y = ord-class.lexordp x y⟩ **and**
term-order-rel'-alt-def:
 ⟨term-order-rel' x y ↔ x < y⟩
 ⟨proof⟩

lemma *list-rel-list-rel-order-iff*:

assumes ⟨(a, b) ∈ ⟨string-rel⟩list-rel⟩ ⟨(a', b') ∈ ⟨string-rel⟩list-rel⟩

shows $\langle a < a' \longleftrightarrow b < b' \rangle$
 $\langle proof \rangle$

lemma *string-rel-le*[*sepref-import-param*]:
shows $\langle ((<), (<)) \in \langle string-rel \rangle list-rel \rightarrow \langle string-rel \rangle list-rel \rightarrow bool-rel \rangle$
 $\langle proof \rangle$

lemma [*sepref-import-param*]:
assumes $\langle CONSTRAINT\ IS\ LEFT\ UNIQUE\ R \rangle \langle CONSTRAINT\ IS\ RIGHT\ UNIQUE\ R \rangle$
shows $\langle (remove1, remove1) \in R \rightarrow \langle R \rangle list-rel \rightarrow \langle R \rangle list-rel \rangle$
 $\langle proof \rangle$

instantiation *pac-step* :: (*heap*, *heap*, *heap*) *heap*
begin

instance
 $\langle proof \rangle$

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* :: $- \Rightarrow 'a\ list \Rightarrow 'a\ list \Rightarrow 'a\ list$
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*]:
 $mset\ (merge\ f\ xs\ ys) = mset\ xs + mset\ ys$
 $\langle proof \rangle$

lemma *set-merge* [*simp*]:
 $set\ (merge\ f\ xs\ ys) = set\ xs \cup set\ ys$
 $\langle proof \rangle$

lemma *sorted-merge*:
 $transp\ f \implies (\bigwedge x\ y. f\ x\ y \vee f\ y\ x) \implies$
 $sorted-wrt\ f\ (merge\ f\ xs\ ys) \longleftrightarrow sorted-wrt\ f\ xs \wedge sorted-wrt\ f\ ys$
 $\langle proof \rangle$

fun *msort* :: $- \Rightarrow 'a\ list \Rightarrow 'a\ list$

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* :: $\langle \Rightarrow \text{nat} \Rightarrow \text{nat} \Rightarrow ('a \times 'a \times 'a) \Rightarrow ('a \times 'a \times 'a) \rangle$ **where**

```

  swap-ternary f m n =
    (if (m = 0 ∧ n = 1)
      then (λ(a, b, c). if f a b then (a, b, c)
        else (b,a,c))
      else if (m = 0 ∧ n = 2)
        then (λ(a, b, c). if f a c then (a, b, c)
          else (c,b,a))
      else if (m = 1 ∧ n = 2)
        then (λ(a, b, c). if f b c then (a, b, c)
          else (a,c,b))
      else (λ(a, b, c). (a,b,c)))

```

fun *msort2* :: $- \Rightarrow 'a \text{ list} \Rightarrow 'a \text{ list}$

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 $\langle \text{msort } (\leq) \text{ xs} = \text{msort2 } (\leq) \text{ xs} \rangle$

$\langle \text{proof} \rangle$

lemma *sorted-msort*:

$\text{transp } f \implies (\bigwedge x y. f x y \vee f y x) \implies$

$\text{sorted-wrt } f (\text{msort } f \text{ xs})$

$\langle \text{proof} \rangle$

lemma *mset-msort*[simp]:

$\text{mset } (\text{msort } f \text{ xs}) = \text{mset } \text{xs}$

$\langle \text{proof} \rangle$

12.2 Sorting applied to monomials

lemma *merge-coeffs-alt-def*:

$\langle (\text{RETURN } o \text{ merge-coeffs}) p =$

$REC_T(\lambda f p.$
 $\quad (case\ p\ of$
 $\quad \quad [] \Rightarrow RETURN\ []$
 $\quad | [-] \Rightarrow RETURN\ p$
 $\quad | ((xs, n) \# (ys, m) \# p) \Rightarrow$
 $\quad \quad (if\ xs = ys$
 $\quad \quad \quad then\ if\ n + m \neq 0\ then\ f\ ((xs, n + m) \# p)\ else\ f\ p$
 $\quad \quad \quad else\ do\ \{p \leftarrow f\ ((ys, m) \# p);\ RETURN\ ((xs, n) \# p)\})$
 $\quad \quad p)$
 $\langle proof \rangle$

lemma *hn-invalid-recover*:

$\langle is-pure\ R \implies hn-invalid\ R = (\lambda x\ y. R\ x\ y * true) \rangle$
 $\langle is-pure\ R \implies invalid-assn\ R = (\lambda x\ y. R\ x\ y * true) \rangle$
 $\langle proof \rangle$

lemma *safe-poly-vars*:

shows

$[safe-constraint-rules]:$
 $\quad is-pure\ (poly-assn)\ \mathbf{and}$
 $[safe-constraint-rules]:$
 $\quad is-pure\ (monom-assn)\ \mathbf{and}$
 $[safe-constraint-rules]:$
 $\quad is-pure\ (monomial-assn)\ \mathbf{and}$
 $[safe-constraint-rules]:$
 $\quad is-pure\ string-assn$
 $\langle proof \rangle$

lemma *invalid-assn-distrib*:

$\langle invalid-assn\ monom-assn \times_a invalid-assn\ int-assn = invalid-assn\ (monom-assn \times_a int-assn) \rangle$
 $\langle proof \rangle$

lemma *WTF-RF-recover*:

$\langle hn-ctxt\ (invalid-assn\ monom-assn \times_a invalid-assn\ int-assn)\ xb$
 $\quad x'a \vee_A$
 $\quad hn-ctxt\ monomial-assn\ xb\ x'a \implies_t$
 $\quad hn-ctxt\ (monomial-assn)\ xb\ x'a \rangle$
 $\langle proof \rangle$

lemma *WTF-RF*:

$\langle hn-ctxt\ (invalid-assn\ monom-assn \times_a invalid-assn\ int-assn)\ xb\ x'a *$
 $\quad (hn-invalid\ poly-assn\ la\ l' a * hn-invalid\ int-assn\ a2' a2 *$
 $\quad \quad hn-invalid\ monom-assn\ a1' a1 *$
 $\quad \quad hn-invalid\ poly-assn\ l\ l' *$
 $\quad \quad hn-invalid\ monomial-assn\ xa\ x' *$
 $\quad \quad hn-invalid\ poly-assn\ ax\ px) \implies_t$
 $\quad hn-ctxt\ (monomial-assn)\ xb\ x'a *$
 $\quad hn-ctxt\ poly-assn$
 $\quad \quad la\ l' a *$
 $\quad \quad hn-ctxt\ poly-assn\ l\ l' *$
 $\quad \quad (hn-invalid\ int-assn\ a2' a2 *$
 $\quad \quad \quad hn-invalid\ monom-assn\ a1' a1 *$
 $\quad \quad \quad hn-invalid\ monomial-assn\ xa\ x' *$
 $\quad \quad \quad hn-invalid\ poly-assn\ ax\ px) \rangle$
 $\langle hn-ctxt\ (invalid-assn\ monom-assn \times_a invalid-assn\ int-assn)\ xa\ x' *$

$(hn\text{-}ctxt\ poly\text{-}assn\ l\ l' * hn\text{-}invalid\ poly\text{-}assn\ ax\ px) \Rightarrow_t$
 $hn\text{-}ctxt\ (monomial\text{-}assn)\ xa\ x' *$
 $hn\text{-}ctxt\ poly\text{-}assn\ l\ l' *$
 $hn\text{-}ctxt\ poly\text{-}assn\ ax\ px *$
 emp
 $\langle proof \rangle$

The refinement framework is completely lost here when synthesizing the constants – it does not understand what is pure (actually everything) and what must be destroyed.

sempref-definition *merge-coeffs-impl*
is $\langle RETURN\ o\ merge\text{-}coeffs \rangle$
 $:: \langle poly\text{-}assn^d \rightarrow_a poly\text{-}assn \rangle$
 $\langle proof \rangle$

definition *full-quicksort-poly* **where**
 $\langle full\text{-}quicksort\text{-}poly = full\text{-}quicksort\text{-}ref\ (\lambda x\ y.\ x = y \vee (x, y) \in term\text{-}order\text{-}rel)\ fst \rangle$

lemma *down-eq-id-list-rel*: $\langle \Downarrow (\langle Id \rangle list\text{-}rel)\ x = x \rangle$
 $\langle proof \rangle$

definition *quicksort-poly*: $\langle nat \Rightarrow nat \Rightarrow llist\text{-}polynom \Rightarrow (llist\text{-}polynom)\ nres \rangle$ **where**
 $\langle quicksort\text{-}poly\ x\ y\ z = quicksort\text{-}ref\ (\leq)\ fst\ (x, y, z) \rangle$

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**
 $\langle partition\text{-}between\text{-}poly = partition\text{-}between\text{-}ref\ (\leq)\ fst \rangle$

definition *partition-main-poly* :: $\langle nat \Rightarrow nat \Rightarrow llist\text{-}polynom \Rightarrow (llist\text{-}polynom \times nat)\ nres \rangle$ **where**
 $\langle partition\text{-}main\text{-}poly = partition\text{-}main\ (\leq)\ fst \rangle$

lemma *string-list-trans*:
 $\langle (xa :: char\ list\ list, ya) \in lexord\ (lexord\ \{(x, y). x < y\}) \Rightarrow$
 $(ya, z) \in lexord\ (lexord\ \{(x, y). x < y\}) \Rightarrow$
 $(xa, z) \in lexord\ (lexord\ \{(x, y). x < y\}) \rangle$
 $\langle proof \rangle$

lemma *full-quicksort-sort-poly-spec*:
 $\langle (full\text{-}quicksort\text{-}poly, sort\text{-}poly\text{-}spec) \in \langle Id \rangle list\text{-}rel \rightarrow_f \langle \langle Id \rangle list\text{-}rel \rangle nres\text{-}rel \rangle$
 $\langle proof \rangle$

12.3 Lifting to polynomials

definition *merge-sort-poly* :: $\langle \cdot \rangle$ **where**
 $\langle merge\text{-}sort\text{-}poly = msort\ (\lambda a\ b.\ fst\ a \leq fst\ b) \rangle$

definition *merge-monoms-poly* :: $\langle \cdot \rangle$ **where**
 $\langle merge\text{-}monoms\text{-}poly = msort\ (\leq) \rangle$

definition *merge-poly* :: $\langle \cdot \rangle$ **where**
 $\langle merge\text{-}poly = merge\ (\lambda a\ b.\ fst\ a \leq fst\ b) \rangle$

definition *merge-monoms* :: $\langle \cdot \rangle$ **where**
 $\langle merge\text{-}monoms = merge\ (\leq) \rangle$

definition *msort-poly-impl* :: $\langle (String.literal\ list \times int)\ list \Rightarrow \rightarrow \rangle$ **where**
 $\langle msort\text{-}poly\text{-}impl = msort\ (\lambda a\ b.\ fst\ a \leq fst\ b) \rangle$

definition *msort-monoms-impl* :: $\langle (String.literal\ list) \Rightarrow \rightarrow \rangle$ **where**
 $\langle msort\text{-}monoms\text{-}impl = msort\ (\leq) \rangle$

lemma *msort-poly-impl-alt-def*:

$\langle msort\text{-}poly\text{-}impl\ xs =$
 $\quad (case\ xs\ of$
 $\quad \quad [] \Rightarrow []$
 $\quad \quad | [a] \Rightarrow [a]$
 $\quad \quad | [a,b] \Rightarrow if\ fst\ a \leq fst\ b\ then\ [a,b]\ else\ [b,a]$
 $\quad \quad | xs \Rightarrow merge\text{-}poly$
 $\quad \quad \quad (msort\text{-}poly\text{-}impl\ (take\ ((length\ xs)\ div\ 2)\ xs))$
 $\quad \quad \quad (msort\text{-}poly\text{-}impl\ (drop\ ((length\ xs)\ div\ 2)\ xs))) \rangle$
 $\langle proof \rangle$

lemma *le-term-order-rel'*:

$\langle (\leq) = (\lambda x\ y.\ x = y \vee term\text{-}order\text{-}rel'\ x\ y) \rangle$
 $\langle proof \rangle$

fun *lexord-eq* **where**

$\langle lexord\text{-}eq\ []\ - = True \rangle \mid$
 $\langle lexord\text{-}eq\ (x\ \#\ xs)\ (y\ \#\ ys) = (x < y \vee (x = y \wedge lexord\text{-}eq\ xs\ ys)) \rangle \mid$
 $\langle lexord\text{-}eq\ -\ - = False \rangle$

lemma [*simp*]:

$\langle lexord\text{-}eq\ []\ [] = True \rangle$
 $\langle lexord\text{-}eq\ (a\ \#\ b)\ [] = False \rangle$
 $\langle lexord\text{-}eq\ []\ (a\ \#\ b) = True \rangle$
 $\langle proof \rangle$

lemma *var-order-rel'*:

$\langle (\leq) = (\lambda x\ y.\ x = y \vee (x,y) \in var\text{-}order\text{-}rel) \rangle$
 $\langle proof \rangle$

lemma *var-order-rel''*:

$\langle (x,y) \in var\text{-}order\text{-}rel \longleftrightarrow x < y \rangle$
 $\langle proof \rangle$

lemma *lexord-eq-alt-def1*:

$\langle a \leq b = lexord\text{-}eq\ a\ b \rangle$ **for** $a\ b :: \langle String.literal\ list \rangle$
 $\langle proof \rangle$

lemma *lexord-eq-alt-def2*:

$\langle (RETURN\ oo\ lexord\text{-}eq)\ xs\ ys =$
 $\quad REC_T\ (\lambda f\ (xs,\ ys).$
 $\quad \quad case\ (xs,\ ys)\ of$
 $\quad \quad \quad ([],\ -) \Rightarrow RETURN\ True$
 $\quad \quad | (x\ \#\ xs,\ y\ \#\ ys) \Rightarrow$
 $\quad \quad \quad if\ x < y\ then\ RETURN\ True$
 $\quad \quad \quad else\ if\ x = y\ then\ f\ (xs,\ ys)\ else\ RETURN\ False$
 $\quad \quad | - \Rightarrow RETURN\ False)$
 $\quad \quad (xs,\ ys) \rangle$

$\langle proof \rangle$

definition *var-order'* **where**

[simp]: $\langle var-order' = var-order \rangle$

lemma *var-order-rel*[*def-pat-rules*]:

$\langle (\in) \$ (x, y) \$ var-order-rel \equiv var-order' \$ x \$ y \rangle$

$\langle proof \rangle$

lemma *var-order-rel-alt-def*:

$\langle var-order-rel = p2rel \ char.lexordp \rangle$

$\langle proof \rangle$

lemma *var-order-rel-var-order*:

$\langle (x, y) \in var-order-rel \longleftrightarrow var-order \ x \ y \rangle$

$\langle proof \rangle$

lemma *var-order-string-le*[*sepref-import-param*]:

$\langle ((<), var-order') \in string-rel \rightarrow string-rel \rightarrow bool-rel \rangle$

$\langle proof \rangle$

lemma [*sepref-import-param*]:

$\langle ((\leq), (\leq)) \in monom-rel \rightarrow monom-rel \rightarrow bool-rel \rangle$

$\langle proof \rangle$

lemma [*sepref-import-param*]:

$\langle ((<), (<)) \in string-rel \rightarrow string-rel \rightarrow bool-rel \rangle$

$\langle proof \rangle$

lemma [*sepref-import-param*]:

$\langle ((\leq), (\leq)) \in string-rel \rightarrow string-rel \rightarrow bool-rel \rangle$

$\langle proof \rangle$

sepref-register *lexord-eq*

sepref-definition *lexord-eq-term*

is $\langle uncurry \ (RETURN \ oo \ lexord-eq) \rangle$

$\langle :: \langle monom-assn^k *_a monom-assn^k \rightarrow_a bool-assn \rangle \rangle$

$\langle proof \rangle$

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*:

$\langle (a, aa) \in term-order-rel \implies$

$(aa, ab) \in term-order-rel \implies (a, ab) \in term-order-rel \rangle$

$\langle proof \rangle$

lemma *merge-sort-poly-sort-poly-spec*:

$\langle (RETURN \ o \ merge-sort-poly, sort-poly-spec) \in \langle Id \rangle list-rel \rightarrow_f \langle \langle Id \rangle list-rel \rangle nres-rel \rangle$

⟨proof⟩

lemma *msort-alt-def*:

⟨RETURN o (msort f) =
 REC_T (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)}⟩
 ⟨proof⟩

lemma *monomial-rel-order-map*:

⟨(x, a, b) ∈ monomial-rel ⇒
 (y, aa, bb) ∈ monomial-rel ⇒
 fst x ≤ fst y ⟷ a ≤ aa⟩
 ⟨proof⟩

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 ×_r int-rel)⟩ **and**
poly-assn-list:
 ⟨poly-assn = pure (⟨monom-rel ×_r int-rel⟩list-rel)⟩
 ⟨proof⟩

lemma *safe-pac-step-rel-assn[safe-constraint-rules]*:

is-pure K ⇒ is-pure V ⇒ is-pure R ⇒ is-pure (pac-step-rel-assn K V R)
 ⟨proof⟩

lemma *merge-poly-merge-poly*:

⟨(merge-poly, merge-poly)
 ∈ poly-rel → poly-rel → poly-rel⟩
 ⟨proof⟩

lemmas [*fcomp-norm-unfold*] =

poly-assn-list[symmetric]
step-rewrite-pure(1)

lemma *merge-poly-merge-poly2*:

⟨(a, b) ∈ poly-rel ⇒ (a', b') ∈ poly-rel ⇒
 (merge-poly a a', merge-poly b b') ∈ poly-rel⟩
 ⟨proof⟩

lemma *list-rel-takeD*:

⟨(a, b) ∈ ⟨R⟩list-rel ⇒ (n, n') ∈ Id ⇒ (take n a, take n' b) ∈ ⟨R⟩list-rel⟩
 ⟨proof⟩

lemma *list-rel-dropD*:

⟨(a, b) ∈ ⟨R⟩list-rel ⇒ (n, n') ∈ Id ⇒ (drop n a, drop n' b) ∈ ⟨R⟩list-rel⟩

$\langle \text{proof} \rangle$

lemma *merge-sort-poly*[*sepref-import-param*]:
 $\langle (\text{msort-poly-impl}, \text{merge-sort-poly})$
 $\in \text{poly-rel} \rightarrow \text{poly-rel}$
 $\langle \text{proof} \rangle$

lemmas [*sepref-fr-rules*] = *merge-sort-poly*[*FCOMP merge-sort-poly-sort-poly-spec*]

sepref-definition *partition-main-poly-impl*
 is $\langle \text{uncurry2 } \text{partition-main-poly} \rangle$
 $\text{:: } \langle \text{nat-assn}^k *_a \text{ nat-assn}^k *_a \text{ poly-assn}^k \rightarrow_a \text{ prod-assn poly-assn nat-assn} \rangle$
 $\langle \text{proof} \rangle$

declare *partition-main-poly-impl.refine*[*sepref-fr-rules*]

sepref-definition *partition-between-poly-impl*
 is $\langle \text{uncurry2 } \text{partition-between-poly} \rangle$
 $\text{:: } \langle \text{nat-assn}^k *_a \text{ nat-assn}^k *_a \text{ poly-assn}^k \rightarrow_a \text{ prod-assn poly-assn nat-assn} \rangle$
 $\langle \text{proof} \rangle$

declare *partition-between-poly-impl.refine*[*sepref-fr-rules*]

sepref-definition *quicksort-poly-impl*
 is $\langle \text{uncurry2 } \text{quicksort-poly} \rangle$
 $\text{:: } \langle \text{nat-assn}^k *_a \text{ nat-assn}^k *_a \text{ poly-assn}^k \rightarrow_a \text{ poly-assn} \rangle$
 $\langle \text{proof} \rangle$

lemmas [*sepref-fr-rules*] = *quicksort-poly-impl.refine*

sepref-register *quicksort-poly*
sepref-definition *full-quicksort-poly-impl*
 is $\langle \text{full-quicksort-poly} \rangle$
 $\text{:: } \langle \text{poly-assn}^k \rightarrow_a \text{ poly-assn} \rangle$
 $\langle \text{proof} \rangle$

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 $\langle \text{normalize-poly} \rangle$
 $\text{:: } \langle \text{poly-assn}^k \rightarrow_a \text{ poly-assn} \rangle$
 $\langle \text{proof} \rangle$

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*:
 $\langle xs \neq ys \implies (xs, ys) \notin \text{lexord } (\text{less-than-char}) \longleftrightarrow$
 $(ys, xs) \in \text{lexord } \text{less-than-char} \rangle$
 $\langle \text{proof} \rangle$

lemma *string-trans*:
 $\langle (xa, ya) \in \text{lexord } \{(x::\text{char}, y::\text{char}). x < y\} \implies$
 $(ya, z) \in \text{lexord } \{(x::\text{char}, y::\text{char}). x < y\} \implies$
 $(xa, z) \in \text{lexord } \{(x::\text{char}, y::\text{char}). x < y\} \rangle$
 $\langle \text{proof} \rangle$

lemma *full-quicksort-sort-vars-spec*:
 $\langle (\text{full-quicksort-vars}, \text{sort-coeff}) \in \langle \text{Id} \rangle \text{list-rel} \rightarrow_f \langle \langle \text{Id} \rangle \text{list-rel} \rangle \text{nres-rel} \rangle$
 $\langle \text{proof} \rangle$

sempref-definition *partition-main-vars-impl*
is $\langle \text{uncurry2 } \text{partition-main-vars} \rangle$
 $\langle :: \langle \text{nat-assn}^k *_a \text{nat-assn}^k *_a (\text{monom-assn})^k \rightarrow_a \text{prod-assn } (\text{monom-assn}) \text{ nat-assn} \rangle$
 $\langle \text{proof} \rangle$

declare *partition-main-vars-impl.refine*[sempref-fr-rules]

sempref-definition *partition-between-vars-impl*
is $\langle \text{uncurry2 } \text{partition-between-vars} \rangle$
 $\langle :: \langle \text{nat-assn}^k *_a \text{nat-assn}^k *_a \text{monom-assn}^k \rightarrow_a \text{prod-assn } \text{monom-assn } \text{nat-assn} \rangle$
 $\langle \text{proof} \rangle$

declare *partition-between-vars-impl.refine*[sempref-fr-rules]

sempref-definition *quicksort-vars-impl*
is $\langle \text{uncurry2 } \text{quicksort-vars} \rangle$
 $\langle :: \langle \text{nat-assn}^k *_a \text{nat-assn}^k *_a \text{monom-assn}^k \rightarrow_a \text{monom-assn} \rangle$
 $\langle \text{proof} \rangle$

lemmas [sempref-fr-rules] = *quicksort-vars-impl.refine*

sempref-register *quicksort-vars*

lemma *le-var-order-rel*:
 $\langle (\leq) = (\lambda x \ y. x = y \vee (x, y) \in \text{var-order-rel}) \rangle$
 $\langle \text{proof} \rangle$

sepref-definition *full-quicksort-vars-impl*

is $\langle \text{full-quicksort-vars} \rangle$
 $:: \langle \text{monom-assn}^k \rightarrow_a \text{monom-assn} \rangle$
 $\langle \text{proof} \rangle$

lemmas *sort-vars-spec-hnr* =

full-quicksort-vars-impl.refine[*FCOMP full-quicksort-sort-vars-spec*]

lemma *string-rel-order-map*:

$\langle (x, a) \in \text{string-rel} \implies$
 $(y, aa) \in \text{string-rel} \implies$
 $x \leq y \longleftrightarrow a \leq aa \rangle$
 $\langle \text{proof} \rangle$

lemma *merge-monoms-merge-monoms*:

$\langle (\text{merge-monoms}, \text{merge-monoms}) \in \text{monom-rel} \rightarrow \text{monom-rel} \rightarrow \text{monom-rel} \rangle$
 $\langle \text{proof} \rangle$

lemma *merge-monoms-merge-monoms2*:

$\langle (a, b) \in \text{monom-rel} \implies (a', b') \in \text{monom-rel} \implies$
 $(\text{merge-monoms } a \ a', \text{ merge-monoms } b \ b') \in \text{monom-rel} \rangle$
 $\langle \text{proof} \rangle$

lemma *msort-monoms-impl*:

$\langle (\text{msort-monoms-impl}, \text{merge-monoms-poly})$
 $\in \text{monom-rel} \rightarrow \text{monom-rel} \rangle$
 $\langle \text{proof} \rangle$

lemma *merge-sort-monoms-sort-monoms-spec*:

$\langle (\text{RETURN } o \ \text{merge-monoms-poly}, \text{sort-coeff}) \in \langle \text{Id} \rangle \text{list-rel} \rightarrow_f \langle \langle \text{Id} \rangle \text{list-rel} \rangle \text{nres-rel} \rangle$
 $\langle \text{proof} \rangle$

sepref-register *sort-coeff*

lemma [*sepref-fr-rules*]:

$\langle (\text{return } o \ \text{msort-monoms-impl}, \text{sort-coeff}) \in \text{monom-assn}^k \rightarrow_a \text{monom-assn} \rangle$
 $\langle \text{proof} \rangle$

sepref-definition *sort-all-coeffs-impl*

is $\langle \text{sort-all-coeffs} \rangle$
 $:: \langle \text{poly-assn}^k \rightarrow_a \text{poly-assn} \rangle$
 $\langle \text{proof} \rangle$

declare *sort-all-coeffs-impl.refine*[*sepref-fr-rules*]

lemma *merge-coeffs0-alt-def*:

$\langle (\text{RETURN } o \ \text{merge-coeffs0}) \ p =$
 $\text{REC}_T(\lambda f \ p.$
 $(\text{case } p \text{ of}$
 $\quad [] \Rightarrow \text{RETURN } []$
 $\quad | [p] \Rightarrow \text{if } \text{snd } p = 0 \text{ then } \text{RETURN } [] \text{ else } \text{RETURN } [p]$
 $\quad | ((xs, n) \# (ys, m) \# p) \Rightarrow$
 $\quad (\text{if } xs = ys$
 $\quad \text{then if } n + m \neq 0 \text{ then } f((xs, n + m) \# p) \text{ else } f \ p$


```

      else if  $n = 0$  then
        do  $\{p \leftarrow f((ys, m) \# p);$ 
          RETURN  $p\}$ 
      else do  $\{p \leftarrow f((ys, m) \# p);$ 
        RETURN  $((xs, n) \# p)\}$ 
     $p\}$ 
   $\langle proof \rangle$ 

```

Again, Sepref does not understand what is going here.

```

sepref-definition merge-coeffs0-impl
  is  $\langle RETURN \ o \ merge-coeffs0 \rangle$ 
   $:: \langle poly-assn^k \rightarrow_a poly-assn \rangle$ 
   $\langle proof \rangle$ 

```

```

declare merge-coeffs0-impl.refine[sepref-fr-rules]

```

```

sepref-definition fully-normalize-poly-impl
  is  $\langle full-normalize-poly \rangle$ 
   $:: \langle poly-assn^k \rightarrow_a poly-assn \rangle$ 
   $\langle proof \rangle$ 

```

```

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  $\langle version \rangle$ , NoSyn),
       ((binding  $\langle version-def \rangle$ , []), 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*:
 $\langle \text{vars-of-monom-in } xs \mathcal{V} \longleftrightarrow \text{set } xs \subseteq \mathcal{V} \rangle$
 $\langle \text{proof} \rangle$

lemma *vars-llist-alt-def*:
 $\langle \text{vars-llist } xs \subseteq \mathcal{V} \longleftrightarrow \text{vars-of-poly-in } xs \mathcal{V} \rangle$
 $\langle \text{proof} \rangle$

lemma *vars-of-monom-in-alt-def2*:
 $\langle \text{vars-of-monom-in } xs \mathcal{V} \longleftrightarrow \text{fold } (\lambda x b. b \wedge x \in \mathcal{V}) \text{ } xs \text{ True} \rangle$
 $\langle \text{proof} \rangle$

sempref-definition *vars-of-monom-in-impl*
is $\langle \text{uncurry } (\text{RETURN } \text{oo vars-of-monom-in}) \rangle$
 $\langle (list-assn \text{ string-assn})^k *_a \text{ vars-assn}^k \rightarrow_a \text{ bool-assn} \rangle$
 $\langle \text{proof} \rangle$

declare *vars-of-monom-in-impl.refine*[sempref-fr-rules]

lemma *vars-of-poly-in-alt-def2*:
 $\langle \text{vars-of-poly-in } xs \mathcal{V} \longleftrightarrow \text{fold } (\lambda(x, -) b. b \wedge \text{vars-of-monom-in } x \mathcal{V}) \text{ } xs \text{ True} \rangle$
 $\langle \text{proof} \rangle$

sempref-definition *vars-of-poly-in-impl*
is $\langle \text{uncurry } (\text{RETURN } \text{oo vars-of-poly-in}) \rangle$
 $\langle (poly-assn)^k *_a \text{ vars-assn}^k \rightarrow_a \text{ bool-assn} \rangle$
 $\langle \text{proof} \rangle$

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**
 $\langle \text{union-vars-poly } xs \mathcal{V} = \text{fold } (\lambda(xs, -) \mathcal{V}. \text{union-vars-monom } xs \mathcal{V}) \text{ } xs \mathcal{V} \rangle$

lemma *union-vars-monom-alt-def*:
 $\langle \text{union-vars-monom } xs \mathcal{V} = \mathcal{V} \cup \text{set } xs \rangle$
 $\langle \text{proof} \rangle$

lemma *union-vars-poly-alt-def*:
 $\langle \text{union-vars-poly } xs \mathcal{V} = \mathcal{V} \cup \text{vars-llist } xs \rangle$

$\langle \text{proof} \rangle$

sempref-definition *union-vars-monom-impl*
is $\langle \text{uncurry } (\text{RETURN } \text{oo } \text{union-vars-monom}) \rangle$
 $:: \langle \text{monom-assn}^k *_a \text{vars-assn}^d \rightarrow_a \text{vars-assn} \rangle$
 $\langle \text{proof} \rangle$

declare *union-vars-monom-impl.refine*[sempref-fr-rules]

sempref-definition *union-vars-poly-impl*
is $\langle \text{uncurry } (\text{RETURN } \text{oo } \text{union-vars-poly}) \rangle$
 $:: \langle \text{poly-assn}^k *_a \text{vars-assn}^d \rightarrow_a \text{vars-assn} \rangle$
 $\langle \text{proof} \rangle$

declare *union-vars-poly-impl.refine*[sempref-fr-rules]

hide-const (**open**) *Autoref-Fix-Rel.CONSTRAINT*

fun *status-assn where*
 $\langle \text{status-assn} - \text{CSUCCESS } \text{CSUCCESS} = \text{emp} \rangle \mid$
 $\langle \text{status-assn} - \text{CFOUND } \text{CFOUND} = \text{emp} \rangle \mid$
 $\langle \text{status-assn } R \text{ (CFAILED } a \text{) (CFAILED } b \text{) = } R \text{ } a \text{ } b \rangle \mid$
 $\langle \text{status-assn} - - = \text{false} \rangle$

lemma *SUCCESS-hnr*[sempref-fr-rules]:
 $\langle (\text{uncurry0 } (\text{return } \text{CSUCCESS}), \text{uncurry0 } (\text{RETURN } \text{CSUCCESS})) \in \text{unit-assn}^k \rightarrow_a \text{status-assn } R \rangle$
 $\langle \text{proof} \rangle$

lemma *FOUND-hnr*[sempref-fr-rules]:
 $\langle (\text{uncurry0 } (\text{return } \text{CFOUND}), \text{uncurry0 } (\text{RETURN } \text{CFOUND})) \in \text{unit-assn}^k \rightarrow_a \text{status-assn } R \rangle$
 $\langle \text{proof} \rangle$

lemma *is-success-hnr*[sempref-fr-rules]:
 $\langle \text{CONSTRAINT } \text{is-pure } R \implies$
 $((\text{return } o \text{ is-cfound}), (\text{RETURN } o \text{ is-cfound})) \in (\text{status-assn } R)^k \rightarrow_a \text{bool-assn} \rangle$
 $\langle \text{proof} \rangle$

lemma *is-cfailed-hnr*[sempref-fr-rules]:
 $\langle \text{CONSTRAINT } \text{is-pure } R \implies$
 $((\text{return } o \text{ is-cfailed}), (\text{RETURN } o \text{ is-cfailed})) \in (\text{status-assn } R)^k \rightarrow_a \text{bool-assn} \rangle$
 $\langle \text{proof} \rangle$

lemma *merge-cstatus-hnr*[sempref-fr-rules]:
 $\langle \text{CONSTRAINT } \text{is-pure } R \implies$
 $(\text{uncurry } (\text{return } \text{oo } \text{merge-cstatus}), \text{uncurry } (\text{RETURN } \text{oo } \text{merge-cstatus})) \in$
 $(\text{status-assn } R)^k *_a (\text{status-assn } R)^k \rightarrow_a \text{status-assn } R \rangle$
 $\langle \text{proof} \rangle$

sempref-definition *add-poly-impl*
is $\langle \text{add-poly-l} \rangle$
 $:: \langle (\text{poly-assn } \times_a \text{poly-assn})^k \rightarrow_a \text{poly-assn} \rangle$
 $\langle \text{proof} \rangle$

declare *add-poly-impl.refine*[*sepref-fr-rules*]

sepref-register *mult-monomials*

lemma *mult-monoms-alt-def*:

$\langle (RETURN \text{ oo } \text{mult-monoms}) \ x \ y = REC_T$
 $(\lambda f \ (p, q).$
 $\text{case } (p, q) \text{ of}$
 $\quad ([], -) \Rightarrow RETURN \ q$
 $\quad | \ (-, []) \Rightarrow RETURN \ p$
 $\quad | \ (x \# p, y \# q) \Rightarrow$
 $\quad \quad (\text{if } x = y \text{ then do } \{$
 $\quad \quad \quad pq \leftarrow f \ (p, q);$
 $\quad \quad \quad RETURN \ (x \# pq)\}$
 $\quad \text{else if } (x, y) \in \text{var-order-rel}$
 $\quad \text{then do } \{$
 $\quad \quad pq \leftarrow f \ (p, y \# q);$
 $\quad \quad RETURN \ (x \# pq)\}$
 $\quad \text{else do } \{$
 $\quad \quad pq \leftarrow f \ (x \# p, q);$
 $\quad \quad RETURN \ (y \# pq)\})$
 $\quad (x, y)\rangle$
 $\langle \text{proof} \rangle$

sepref-definition *mult-monoms-impl*

is $\langle \text{uncurry } (RETURN \text{ oo } \text{mult-monoms}) \rangle$
 $:: \langle (\text{monom-assn})^k *_{\alpha} (\text{monom-assn})^k \rightarrow_{\alpha} (\text{monom-assn}) \rangle$
 $\langle \text{proof} \rangle$

declare *mult-monoms-impl.refine*[*sepref-fr-rules*]

sepref-definition *mult-monomials-impl*

is $\langle \text{uncurry } (RETURN \text{ oo } \text{mult-monomials}) \rangle$
 $:: \langle (\text{monomial-assn})^k *_{\alpha} (\text{monomial-assn})^k \rightarrow_{\alpha} (\text{monomial-assn}) \rangle$
 $\langle \text{proof} \rangle$

lemma *map-append-alt-def2*:

$\langle (RETURN \text{ o } (\text{map-append } f \ b)) \ xs = REC_T$
 $(\lambda g \ xs. \text{case } xs \text{ of } [] \Rightarrow RETURN \ b$
 $\quad | \ x \# xs \Rightarrow \text{do } \{$
 $\quad \quad y \leftarrow g \ xs;$
 $\quad \quad RETURN \ (f \ x \ \# \ y)$
 $\quad \}) \ xs \rangle$
 $\langle \text{proof} \rangle$

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 $\langle \text{uncurry2 } (RETURN \text{ ooo } \text{map-append-poly-mult}) \rangle$

$:: \langle \text{monomial-assn}^k *_{\alpha} \text{poly-assn}^k *_{\alpha} \text{poly-assn}^k \rightarrow_{\alpha} \text{poly-assn} \rangle$
 $\langle \text{proof} \rangle$

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 $\langle \text{uncurry } (\text{RETURN } \text{oo } \text{mult-poly-raw}) \rangle$
 $:: \langle \text{poly-assn}^k *_{\alpha} \text{poly-assn}^k \rightarrow_{\alpha} \text{poly-assn} \rangle$
 $\langle \text{proof} \rangle$

declare *mult-poly-raw-impl.refine*[*sepref-fr-rules*]

sepref-definition *mult-poly-impl*
is $\langle \text{uncurry } \text{mult-poly-full} \rangle$
 $:: \langle \text{poly-assn}^k *_{\alpha} \text{poly-assn}^k \rightarrow_{\alpha} \text{poly-assn} \rangle$
 $\langle \text{proof} \rangle$

declare *mult-poly-impl.refine*[*sepref-fr-rules*]

lemma *inverse-monomial*:
 $\langle \text{monom-rel}^{-1} \times_r \text{int-rel} = (\text{monom-rel} \times_r \text{int-rel})^{-1} \rangle$
 $\langle \text{proof} \rangle$

lemma *eq-poly-rel-eq*[*sepref-import-param*]:
 $\langle ((=), (=)) \in \text{poly-rel} \rightarrow \text{poly-rel} \rightarrow \text{bool-rel} \rangle$
 $\langle \text{proof} \rangle$

sepref-definition *weak-equality-l-impl*
is $\langle \text{uncurry } \text{weak-equality-l} \rangle$
 $:: \langle \text{poly-assn}^k *_{\alpha} \text{poly-assn}^k \rightarrow_{\alpha} \text{bool-assn} \rangle$
 $\langle \text{proof} \rangle$

declare *weak-equality-l-impl.refine*[*sepref-fr-rules*]

sepref-register *add-poly-l mult-poly-full*

abbreviation *raw-string-assn* $:: \langle \text{string} \Rightarrow \text{string} \Rightarrow \text{assn} \rangle$ **where**
 $\langle \text{raw-string-assn} \equiv \text{list-assn id-assn} \rangle$

definition *show-nat* $:: \langle \text{nat} \Rightarrow \text{string} \rangle$ **where**
 $\langle \text{show-nat } i = \text{show } i \rangle$

lemma [*sepref-import-param*]:
 $\langle (\text{show-nat}, \text{show-nat}) \in \text{nat-rel} \rightarrow \langle \text{Id} \rangle \text{list-rel} \rangle$
 $\langle \text{proof} \rangle$

lemma *status-assn-pure-conv*:
 $\langle \text{status-assn } (\text{id-assn}) a b = \text{id-assn } a b \rangle$
 $\langle \text{proof} \rangle$

lemma [*sepref-fr-rules*]:
 $\langle (\text{uncurry3 } (\lambda x y. \text{return } \text{oo } (\text{error-msg-not-equal-dom } x y)), \text{uncurry3 } \text{check-not-equal-dom-err}) \in$
 $\text{poly-assn}^k *_{\alpha} \text{poly-assn}^k *_{\alpha} \text{poly-assn}^k *_{\alpha} \text{poly-assn}^k \rightarrow_{\alpha} \text{raw-string-assn} \rangle$

$\langle \text{proof} \rangle$

lemma [sepref-fr-rules]:

$\langle (\text{return } o \text{ (error-msg-notin-dom } o \text{ nat-of-uint64)}, \text{RETURN } o \text{ error-msg-notin-dom})$
 $\in \text{uint64-nat-assn}^k \rightarrow_a \text{raw-string-assn} \rangle$
 $\langle (\text{return } o \text{ (error-msg-reused-dom } o \text{ nat-of-uint64)}, \text{RETURN } o \text{ error-msg-reused-dom})$
 $\in \text{uint64-nat-assn}^k \rightarrow_a \text{raw-string-assn} \rangle$
 $\langle (\text{uncurry} (\text{return } oo \text{ } (\lambda i. \text{error-msg } (\text{nat-of-uint64 } i))), \text{uncurry} (\text{RETURN } oo \text{ error-msg}))$
 $\in \text{uint64-nat-assn}^k *_a \text{raw-string-assn}^k \rightarrow_a \text{status-assn raw-string-assn} \rangle$
 $\langle (\text{uncurry} (\text{return } oo \text{ error-msg}), \text{uncurry} (\text{RETURN } oo \text{ error-msg}))$
 $\in \text{nat-assn}^k *_a \text{raw-string-assn}^k \rightarrow_a \text{status-assn raw-string-assn} \rangle$
 $\langle \text{proof} \rangle$

sepref-definition *check-addition-l-impl*

is $\langle \text{uncurry6 } \text{check-addition-l} \rangle$
 $:: \langle \text{poly-assn}^k *_a \text{polys-assn}^k *_a \text{vars-assn}^k *_a \text{uint64-nat-assn}^k *_a \text{uint64-nat-assn}^k *_a$
 $\text{uint64-nat-assn}^k *_a \text{poly-assn}^k \rightarrow_a \text{status-assn raw-string-assn} \rangle$
 $\langle \text{proof} \rangle$

declare *check-addition-l-impl.refine*[sepref-fr-rules]

sepref-register *check-mult-l-dom-err*

definition *check-mult-l-dom-err-impl* **where**

$\langle \text{check-mult-l-dom-err-impl } pd \text{ } p \text{ } ia \text{ } i =$
 $(\text{if } pd \text{ then "The polynom with id " } @ \text{ show (nat-of-uint64 } p) @ \text{ " was not found" else ""}) @$
 $(\text{if } ia \text{ then "The id of the resulting id " } @ \text{ show (nat-of-uint64 } i) @ \text{ " was already given" else ""}) \rangle$

definition *check-mult-l-mult-err-impl* **where**

$\langle \text{check-mult-l-mult-err-impl } p \text{ } q \text{ } pq \text{ } r =$
 $\text{"Multiplying " } @ \text{ show } p @ \text{ " by " } @ \text{ show } q @ \text{ " gives " } @ \text{ show } pq @ \text{ " and not " } @ \text{ show } r \rangle$

lemma [sepref-fr-rules]:

$\langle (\text{uncurry3 } ((\lambda x \text{ } y. \text{return } oo \text{ (check-mult-l-dom-err-impl } x \text{ } y))),$
 $\text{uncurry3 } (\text{check-mult-l-dom-err})) \in \text{bool-assn}^k *_a \text{uint64-nat-assn}^k *_a \text{bool-assn}^k *_a \text{uint64-nat-assn}^k$
 $\rightarrow_a \text{raw-string-assn} \rangle$
 $\langle \text{proof} \rangle$

lemma [sepref-fr-rules]:

$\langle (\text{uncurry3 } ((\lambda x \text{ } y. \text{return } oo \text{ (check-mult-l-mult-err-impl } x \text{ } y))),$
 $\text{uncurry3 } (\text{check-mult-l-mult-err})) \in \text{poly-assn}^k *_a \text{poly-assn}^k *_a \text{poly-assn}^k *_a \text{poly-assn}^k \rightarrow_a \text{raw-string-assn} \rangle$
 $\langle \text{proof} \rangle$

sepref-definition *check-mult-l-impl*

is $\langle \text{uncurry6 } \text{check-mult-l} \rangle$
 $:: \langle \text{poly-assn}^k *_a \text{polys-assn}^k *_a \text{vars-assn}^k *_a \text{uint64-nat-assn}^k *_a \text{poly-assn}^k *_a \text{uint64-nat-assn}^k *_a$
 $\text{poly-assn}^k \rightarrow_a \text{status-assn raw-string-assn} \rangle$
 $\langle \text{proof} \rangle$

declare *check-mult-l-impl.refine*[sepref-fr-rules]

definition *check-ext-l-dom-err-impl* $:: \langle \text{uint64} \Rightarrow \rightarrow \rangle$ **where**

$\langle \text{check-ext-l-dom-err-impl } p =$

"There is already a polynom with index " @ show (nat-of-wint64 p)

lemma [sepref-fr-rules]:
 $\langle (((\text{return } o \text{ (check-ext-l-dom-err-impl)})),$
 $\text{(check-extension-l-dom-err)}) \in \text{uint64-nat-assn}^k \rightarrow_a \text{raw-string-assn} \rangle$
 $\langle \text{proof} \rangle$

definition check-extension-l-no-new-var-err-impl :: $\langle - \Rightarrow - \rangle$ **where**
 $\langle \text{check-extension-l-no-new-var-err-impl } p =$
 $\text{"No new variable could be found in polynom " @ show } p \rangle$

lemma [sepref-fr-rules]:
 $\langle (((\text{return } o \text{ (check-extension-l-no-new-var-err-impl)})),$
 $\text{(check-extension-l-no-new-var-err)}) \in \text{poly-assn}^k \rightarrow_a \text{raw-string-assn} \rangle$
 $\langle \text{proof} \rangle$

definition check-extension-l-side-cond-err-impl :: $\langle - \Rightarrow - \rangle$ **where**
 $\langle \text{check-extension-l-side-cond-err-impl } v \text{ } p \text{ } r \text{ } s =$
 $\text{"Error while checking side conditions of extensions polynow, var is " @ show } v \text{ @}$
 $\text{" polynom is " @ show } p \text{ @ "side condition } p * p - p = \text{" @ show } s \text{ @ " and should be 0"} \rangle$

lemma [sepref-fr-rules]:
 $\langle (((\text{uncurry3 } (\lambda x y. \text{return } oo \text{ (check-extension-l-side-cond-err-impl } x \text{ } y))),$
 $\text{uncurry3 } (\text{check-extension-l-side-cond-err})) \in \text{string-assn}^k *_a \text{poly-assn}^k *_a \text{poly-assn}^k *_a \text{poly-assn}^k$
 $\rightarrow_a \text{raw-string-assn} \rangle$
 $\langle \text{proof} \rangle$

definition check-extension-l-new-var-multiple-err-impl :: $\langle - \Rightarrow - \rangle$ **where**
 $\langle \text{check-extension-l-new-var-multiple-err-impl } v \text{ } p =$
 $\text{"Error while checking side conditions of extensions polynow, var is " @ show } v \text{ @}$
 $\text{" but it either appears at least once in the polynom or another new variable is created " @}$
 $\text{show } p \text{ @ " but should not."} \rangle$

lemma [sepref-fr-rules]:
 $\langle (((\text{uncurry } (\text{return } oo \text{ (check-extension-l-new-var-multiple-err-impl)})),$
 $\text{uncurry } (\text{check-extension-l-new-var-multiple-err})) \in \text{string-assn}^k *_a \text{poly-assn}^k \rightarrow_a \text{raw-string-assn} \rangle$
 $\langle \text{proof} \rangle$

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 :: $\langle \text{l-list-polynom} \Rightarrow \text{l-list-polynom} \rangle$ **where**
 $\langle \text{uminus-poly } p' = \text{map } (\lambda(a, b). (a, - b)) \text{ } p' \rangle$

sepref-register uminus-poly

lemma [sepref-import-param]:
 $\langle (\text{map } (\lambda(a, b). (a, - b)), \text{uminus-poly}) \in \text{poly-rel} \rightarrow \text{poly-rel} \rangle$
 $\langle \text{proof} \rangle$

sepref-register vars-of-poly-in
weak-equality-l

lemma [safe-constraint-rules]:

⟨Sepref-Constraints.CONSTRAINT single-valued (the-pure monomial-assn)⟩ and
 single-valued-the-monomial-assn:
 ⟨single-valued (the-pure monomial-assn)⟩
 ⟨single-valued ((the-pure monomial-assn)⁻¹)⟩
 ⟨proof⟩

sepref-definition check-extension-l-impl

is ⟨uncurry5 check-extension-l⟩
 :: ⟨poly-assn^k *_a polys-assn^k *_a vars-assn^k *_a uint64-nat-assn^k *_a string-assn^k *_a poly-assn^k →_a
 status-assn raw-string-assn⟩
 ⟨proof⟩

declare check-extension-l-impl.refine[sepref-fr-rules]

sepref-definition check-del-l-impl

is ⟨uncurry2 check-del-l⟩
 :: ⟨poly-assn^k *_a polys-assn^k *_a uint64-nat-assn^k →_a status-assn raw-string-assn⟩
 ⟨proof⟩

lemmas [sepref-fr-rules] = check-del-l-impl.refine

abbreviation pac-step-rel **where**

⟨pac-step-rel ≡ p2rel ((Id, ⟨monomial-rel⟩list-rel, Id) pac-step-rel-raw)⟩

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:

⟨hn-ctxt (pac-step-rel-assn nat-assn poly-assn id-assn) b bi =
 hn-val
 (p2rel
 ((nat-rel, poly-rel, Id :: (string × -) set) pac-step-rel-raw)) b bi⟩
 ⟨proof⟩

lemma is-AddD-import[sepref-fr-rules]:

assumes ⟨CONSTRAINT is-pure K⟩ ⟨CONSTRAINT is-pure V⟩

shows

⟨(return o pac-res, RETURN o pac-res) ∈ [λx. is-Add x ∨ is-Mult x ∨ is-Extension x]_a
 (pac-step-rel-assn K V R)^k → V⟩
 ⟨(return o pac-src1, RETURN o pac-src1) ∈ [λx. is-Add x ∨ is-Mult x ∨ is-Del x]_a (pac-step-rel-assn
 K V R)^k → K⟩
 ⟨(return o new-id, RETURN o new-id) ∈ [λx. is-Add x ∨ is-Mult x ∨ is-Extension x]_a (pac-step-rel-assn
 K V R)^k → K⟩
 ⟨(return o is-Add, RETURN o is-Add) ∈ (pac-step-rel-assn K V R)^k →_a bool-assn⟩
 ⟨(return o is-Mult, RETURN o is-Mult) ∈ (pac-step-rel-assn K V R)^k →_a bool-assn⟩
 ⟨(return o is-Del, RETURN o is-Del) ∈ (pac-step-rel-assn K V R)^k →_a bool-assn⟩
 ⟨(return o is-Extension, RETURN o is-Extension) ∈ (pac-step-rel-assn K V R)^k →_a bool-assn⟩
 ⟨proof⟩

lemma [sepref-fr-rules]:

⟨CONSTRAINT is-pure K ⇒

$\langle \text{return } o \text{ pac-src2}, \text{RETURN } o \text{ pac-src2} \rangle \in [\lambda x. \text{is-Add } x]_a (\text{pac-step-rel-assn } K \ V \ R)^k \rightarrow K$
 $\langle \text{CONSTRAINT is-pure } V \Rightarrow$
 $\langle \text{return } o \text{ pac-mult}, \text{RETURN } o \text{ pac-mult} \rangle \in [\lambda x. \text{is-Mult } x]_a (\text{pac-step-rel-assn } K \ V \ R)^k \rightarrow V$
 $\langle \text{CONSTRAINT is-pure } R \Rightarrow$
 $\langle \text{return } o \text{ new-var}, \text{RETURN } o \text{ new-var} \rangle \in [\lambda x. \text{is-Extension } x]_a (\text{pac-step-rel-assn } K \ V \ R)^k \rightarrow R$
 $\langle \text{proof} \rangle$

lemma *is-Mult-lastI*:

$\langle \neg \text{is-Add } b \Rightarrow \neg \text{is-Mult } b \Rightarrow \neg \text{is-Extension } b \Rightarrow \text{is-Del } b \rangle$
 $\langle \text{proof} \rangle$

sempref-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*:

$\langle \text{PAC-checker-l-step } a \ bcd \ e = (\text{let } (b,c,d) = bcd \text{ in } \text{PAC-checker-l-step}' \ a \ b \ c \ d \ e) \rangle$
 $\langle \text{proof} \rangle$

sempref-decl-intf ('k) *acode-status is* ('k) *code-status*

sempref-decl-intf ('k, 'b, 'lbl) *apac-step is* ('k, 'b, 'lbl) *pac-step*

sempref-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**
 nat-rel-the-pure :
 $\langle \text{nat-rel} = \text{the-pure nat-assn} \rangle$ **and**
 WTF-RF : $\langle \text{pure } (\text{the-pure nat-assn}) = \text{nat-assn} \rangle$
 $\langle \text{proof} \rangle$

lemma [*safe-constraint-rules*]:

$\langle \text{CONSTRAINT IS-LEFT-UNIQUE uint64-nat-rel} \rangle$ **and**
 $\text{single-valued-uint64-nat-rel}[\text{safe-constraint-rules}]$:
 $\langle \text{CONSTRAINT single-valued uint64-nat-rel} \rangle$
 $\langle \text{proof} \rangle$

sempref-definition *check-step-impl*

is $\langle \text{uncurry4 } \text{PAC-checker-l-step}' \rangle$
 $:: \langle \text{poly-assn}^k *_a (\text{status-assn raw-string-assn})^d *_a \text{vars-assn}^d *_a \text{polys-assn}^d *_a (\text{pac-step-rel-assn}$
 $(\text{uint64-nat-assn}) \text{poly-assn } (\text{string-assn} :: \text{string} \Rightarrow -))^d \rightarrow_a$
 $\text{status-assn raw-string-assn} \times_a \text{vars-assn} \times_a \text{polys-assn} \rangle$
 $\langle \text{proof} \rangle$

declare *check-step-impl.refine*[*sempref-fr-rules*]

sempref-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*:

$\langle \text{PAC-checker-l } p \ \mathcal{V} \ A \ \text{status steps} =$

$\langle \text{let } (\mathcal{V}, A) = \mathcal{V}A \text{ in PAC-checker-l' } p \mathcal{V} A \text{ status steps} \rangle$
 $\langle \text{proof} \rangle$

sempref-definition *PAC-checker-l-impl*

is $\langle \text{uncurry4 PAC-checker-l'} \rangle$
 $:: \langle \text{poly-assn}^k *_a \text{vars-assn}^d *_a \text{polys-assn}^d *_a (\text{status-assn raw-string-assn})^d *_a$
 $(\text{list-assn } (\text{pac-step-rel-assn } (\text{uint64-nat-assn}) \text{poly-assn string-assn}))^k \rightarrow_a$
 $\text{status-assn raw-string-assn} \times_a \text{vars-assn} \times_a \text{polys-assn} \rangle$
 $\langle \text{proof} \rangle$

declare *PAC-checker-l-impl.refine*[sempref-fr-rules]

abbreviation *polys-assn-input* **where**

$\langle \text{polys-assn-input} \equiv \text{iam-fmap-assn nat-assn poly-assn} \rangle$

definition *remap-polys-l-dom-err-impl* :: $\langle - \rangle$ **where**

$\langle \text{remap-polys-l-dom-err-impl} =$
 $"\text{Error during initialisation. Too many polynoms where provided. If this happens,}" @$
 $"\text{please report the example to the authors, because something went wrong during}" @$
 $"\text{code generation (code generation to arrays is likely to be broken).}" \rangle$

lemma [sempref-fr-rules]:

$\langle ((\text{uncurry0 } (\text{return } (\text{remap-polys-l-dom-err-impl}))),$
 $\text{uncurry0 } (\text{remap-polys-l-dom-err})) \in \text{unit-assn}^k \rightarrow_a \text{raw-string-assn} \rangle$
 $\langle \text{proof} \rangle$

MLton is not able to optimise the calls to pow.

lemma *pow-2-64*: $\langle (2::\text{nat}) \wedge 64 = 18446744073709551616 \rangle$

$\langle \text{proof} \rangle$

sempref-register *upper-bound-on-dom op-fmap-empty*

sempref-definition *remap-polys-l-impl*

is $\langle \text{uncurry2 remap-polys-l2} \rangle$
 $:: \langle \text{poly-assn}^k *_a \text{vars-assn}^d *_a \text{polys-assn-input}^d \rightarrow_a$
 $\text{status-assn raw-string-assn} \times_a \text{vars-assn} \times_a \text{polys-assn} \rangle$
 $\langle \text{proof} \rangle$

lemma *remap-polys-l2-remap-polys-l*:

$\langle (\text{uncurry2 remap-polys-l2}, \text{uncurry2 remap-polys-l}) \in (\text{Id} \times_r \langle \text{Id} \rangle \text{set-rel}) \times_r \text{Id} \rightarrow_f \langle \text{Id} \rangle \text{nres-rel} \rangle$
 $\langle \text{proof} \rangle$

lemma [sempref-fr-rules]:

$\langle (\text{uncurry2 remap-polys-l-impl},$
 $\text{uncurry2 remap-polys-l}) \in \text{poly-assn}^k *_a \text{vars-assn}^d *_a \text{polys-assn-input}^d \rightarrow_a$
 $\text{status-assn raw-string-assn} \times_a \text{vars-assn} \times_a \text{polys-assn} \rangle$
 $\langle \text{proof} \rangle$

sempref-register *remap-polys-l*

sempref-definition *full-checker-l-impl*

is $\langle \text{uncurry2 full-checker-l} \rangle$
 $:: \langle \text{poly-assn}^k *_a \text{polys-assn-input}^d *_a (\text{list-assn } (\text{pac-step-rel-assn } (\text{uint64-nat-assn}) \text{poly-assn string-assn}))^k$
 \rightarrow_a
 $\text{status-assn raw-string-assn} \times_a \text{vars-assn} \times_a \text{polys-assn} \rangle$

⟨proof⟩

sempref-definition *PAC-update-impl*

is ⟨uncurry2 (RETURN ooo fmupd)⟩
 :: ⟨nat-assn^k *_a poly-assn^k *_a (polys-assn-input)^d →_a polys-assn-input⟩
 ⟨proof⟩

sempref-definition *PAC-empty-impl*

is ⟨uncurry0 (RETURN fmempty)⟩
 :: ⟨unit-assn^k →_a polys-assn-input⟩
 ⟨proof⟩

sempref-definition *empty-vars-impl*

is ⟨uncurry0 (RETURN {})⟩
 :: ⟨unit-assn^k →_a vars-assn⟩
 ⟨proof⟩

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 performance difference is really large.

definition *unsafe-ascii-of-literal* :: ⟨-⟩ **where**

⟨unsafe-ascii-of-literal xs = String.ascii-of-literal xs⟩

definition *unsafe-ascii-of-literal'* :: ⟨-⟩ **where**

[simp, symmetric, code]: ⟨unsafe-ascii-of-literal' = unsafe-ascii-of-literal⟩

code-printing

constant *unsafe-ascii-of-literal'* ↪
 (SML) !(List.map (fn c => let val k = Char.ord c in IntInf.fromInt k end) /o 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]: ⟨raw-explode = String.explode⟩

code-printing

constant *raw-explode* ↪
 (SML) String.explode

definition ⟨hashcode-literal' s ≡

foldl (λh x. h * 33 + uint32-of-int (of-char x)) 5381
 (raw-explode s)⟩

lemmas [code] =

hashcode-literal-def[unfolded String.explode-code
 unsafe-ascii-of-literal-def[symmetric]]

definition *uint32-of-char* **where**

[symmetric, code-unfold]: ⟨uint32-of-char x = uint32-of-int (int-of-char x)⟩

code-printing

```
constant uint32-of-char  $\rightarrow$ 
  (SML) !(Word32.fromInt /o (Char.ord))
```

```
lemma [code]: (hashcode s = hashcode-literal' s)
  (proof)
```

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

```
(code/parser.sml)
(code/pasteque.sml)
(code/pasteque.mlb)
```

where (fn dir =>

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 pasteque ^
```

```
  -codegen native -inline 700 -cc-opt -O3 pasteque.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**

$\langle \text{code-status-assn} = \text{hr-comp } (\text{status-assn } \text{raw-string-assn})$
 $\text{code-status-status-rel} \rangle$

definition *full-vars-assn* **where**

$\langle \text{full-vars-assn} = \text{hr-comp } (\text{hs.assn } \text{string-assn})$
 $(\langle \text{var-rel} \rangle \text{set-rel}) \rangle$

lemma *polys-rel-full-polys-rel*:

$\langle \text{polys-rel-full} = \text{Id} \times_r \text{polys-rel} \rangle$
 $\langle \text{proof} \rangle$

definition *full-polys-assn* :: $\langle \cdot \rangle$ **where**

$\langle \text{full-polys-assn} = \text{hr-comp } (\text{hr-comp } \text{polys-assn}$
 $(\langle \text{nat-rel},$
 $\text{sorted-poly-rel } O \text{ mset-poly-rel} \rangle \text{fmap-rel}))$
 $\text{polys-rel} \rangle$

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*:

$\langle (\text{uncurry2 } \text{full-checker-l-impl},$
 $\text{uncurry2 } (\lambda \text{spec } A \rightarrow \text{PAC-checker-specification } \text{spec } A))$
 $\in (\text{full-poly-assn})^k *_a (\text{full-poly-input-assn})^d *_a (\text{fully-pac-assn})^k \rightarrow_a \text{hr-comp}$
 $(\text{code-status-assn} \times_a \text{full-vars-assn} \times_a \text{hr-comp } \text{polys-assn}$
 $(\langle \text{nat-rel}, \text{sorted-poly-rel } O \text{ mset-poly-rel} \rangle \text{fmap-rel}))$
 $\{((\text{st}, G), \text{st}', G').$
 $\text{st} = \text{st}' \wedge (\text{st} \neq \text{FAILED} \rightarrow (G, G') \in \text{Id} \times_r \text{polys-rel})\}$
 $\langle \text{proof} \rangle$

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 *bij- φ* : $\langle \text{bij } \varphi \rangle$

$\langle \text{proof} \rangle$

global-interpretation *PAC*: *poly-embed* **where**

$\varphi = \varphi$

$\langle \text{proof} \rangle$

The full correctness theorem is $(\text{uncurry2 } \text{full-checker-l-impl}, \text{uncurry2 } (\lambda \text{spec } A. \text{PAC-checker-specification spec } A)) \in \text{PAC.full-poly-assn}^k *_a \text{PAC.full-poly-input-assn}^d *_a \text{PAC.fully-pac-assn}^k \rightarrow_a \text{hr-comp} (\text{PAC.code-status-assn} \times_a \text{PAC.full-vars-assn} \times_a \text{hr-comp polys-assn } (\langle \text{nat-rel}, \text{sorted-poly-rel} \text{ } O \text{ PAC.mset-poly-rel} \rangle \text{fmap-rel})) \{((st, G), st', G'). st = st' \wedge (st \neq \text{FAILED} \longrightarrow (G, G') \in \text{Id} \times_r \text{polys-rel})\}.$

end

References

- [1] D. Kaufmann, M. Fleury, and A. Biere. The proof checkers pacheck and pasteqe 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.