

PAC Checker

Mathias Fleury and Daniela Kaufmann

April 18, 2024

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	Overview	2
2	Libraries	4
2.1	More Polynomials	4
2.2	More Ideals	8
3	Finite maps and multisets	9
3.1	Finite sets and multisets	9
3.2	Finite map and multisets	10
3.3	More Theorem about Loops	26
4	Specification of the PAC checker	27
4.1	Ideals	27
4.2	PAC Format	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	45
7.3	Normalisation	46
7.4	Correctness	46

8	Terms	49
8.1	Ordering	49
8.2	Polynomials	50
8.3	Addition	52
8.4	Multiplication	55
8.5	Normalisation	58
8.6	Multiplication and normalisation	61
8.7	Correctness	61
9	Executable Checker	63
9.1	Definitions	63
9.2	Correctness	70
10	Various Refinement Relations	76
11	Hash Map as association list	80
11.1	Conversion from assoc to other map	82
12	Initial Normalisation of Polynomials	82
12.1	Sorting	82
12.2	Sorting applied to monomials	84
12.3	Lifting to polynomials	86
13	Code Synthesis of the Complete Checker	93
14	Correctness theorem	103

```

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

```

1 Overview

One solution to check circuit of multipliers is to use algebraic method, like producing proofs on polynomials. We are here interested in checking PAC proofs on the Boolean ring. The idea is the following: each variable represents an input or the output of a gate and we want to prove the bitwise multiplication of the input bits yields the output, namely the bitwise representation of the multiplication of the input (modulo 2^n where n is the number of bits of the circuit).

Algebraic proof systems typically reason over polynomials in a ring $\mathbb{K}[X]$, where the variables X represent Boolean values. The aim of an algebraic proof is to derive whether a polynomial f can be derived from a given set of polynomials $G = \{g_1, \dots, g_l\} \subseteq \mathbb{K}[X]$ together with the Boolean value constraints $B(X) = \{x_i^2 - x_i \mid x_i \in X\}$. In algebraic terms this means to show that the polynomial $f \in \langle G \cup B(X) \rangle$.

In our setting we set $\mathbb{K} = \mathbb{Z}$ and we treat the Boolean value constraints implicitly, i.e., we consider proofs in the ring $\mathbb{Z}[X]/\langle B(X) \rangle$ to admit shorter proofs

The checker takes as input 3 files:

1. an input file containing all polynomials that are initially present;
2. a target (or specification) polynomial ;

3. a “proof” file to check that contains the proof in PAC format that shows that the specification is in the ideal generated by the polynomials present initially.

Each step of the proof is either an addition of two polynomials previously derived, a multiplication from a previously derived polynomial and an arbitrary polynomial, and the deletion a derived polynomial.

One restriction on the proofs compared to generic PAC proofs is that $x^2 = x$ in the Boolean ring we are considering.

The checker can produce two outputs: valid (meaning that each derived polynomial in the proof has been correctly derived and the specification polynomial was also derived at some point [either in the proof or as input]) or invalid (without proven information what went wrong).

The development is organised as follows:

- `PAC_Specification.thy` (this file) contains the specification as described above on ideals without any peculiarities on the PAC proof format
- `PAC_Checker_Specification.thy` specialises to the PAC format and enters the non-determinism monad to prepare the subsequent refinements.
- `PAC_Checker.thy` contains the refined version where polynomials are represented as lists.
- `PAC_Checker_Synthesis.thy` contains the efficient implementation with imperative data structure like a hash set.
- `PAC_Checker_MLton.thy` contains the code generation and the command to compile the file with the ML compiler MLton.

Here is an example of a proof and an input file (taken from the appendix of our FMCAD paper [1], available at http://fmv.jku.at/paccheck_pasteque):

```

<res.input>          <res.proof>
1 x*y;                3 = fz, -z+1;
2 y*z-y-z+1;         4 * 3, y-1, -fz*y+fz-y*z+y+z-1;
                       5 + 2, 4, -fz*y+fz;
                       2 d;
                       4 d;
<res.target>         6 * 1, fz, fz*x*y;
-x*z+x;              1 d;
                       7 * 5, x, -fz*x*y+fz*x;
                       8 + 6, 7, fz*x;
                       9 * 3, x, -fz*x-x*z+x;
                       10 + 8, 9, -x*z+x;

```

Each line starts with a number that is used to index the (conclusion) polynomial. In the proof, there are four kind of steps:

1. $3 = fz, -z+1$; is an extension that introduces a new variable (in this case `fz`);

2. 4 * 3, $y-1$, $-fz*y+fz-y*z+y+z-1$; is a multiplication of the existing polynomial with index 3 by the arbitrary polynomial $y-1$ and generates the new polynomial $-fz*y+fz-y*z+y+z-1$ with index 4;
3. 5 + 2, 4, $-fz*y+fz$; is an addition of the existing polynomials with index 2 and 4 and generates the new polynomial $-fz*y+fz$ with index 5;
4. 1 d; deletes the polynomial with index 1 and it cannot be reused in subsequent steps.

Remark that unlike DRAT checker, we do forward checking and check every derived polynomial. The target polynomial can also be part of the input file.

2 Libraries

2.1 More Polynomials

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

lemma *Const₀-add*:

$\langle Const_0 (a + b) = Const_0 a + Const_0 b \rangle$
 $\langle proof \rangle$

lemma *Const-mult*:

$\langle Const (a * b) = Const a * Const b \rangle$
 $\langle proof \rangle$

lemma *Const₀-mult*:

$\langle Const_0 (a * b) = Const_0 a * Const_0 b \rangle$
 $\langle proof \rangle$

lemma *Const0[simp]*:

$\langle Const 0 = 0 \rangle$
 $\langle proof \rangle$

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

$\langle Const (-n) = - Const n \rangle$
 $\langle proof \rangle$

lemma [simp]: $\langle Const_0 0 = 0 \rangle$

$\langle MPoly 0 = 0 \rangle$
 $\langle proof \rangle$

lemma *Const-add*:

$\langle Const (a + b) = Const a + Const b \rangle$
 $\langle proof \rangle$

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

$\langle proof \rangle$

lemma *degree-uminus[simp]*:

$\langle degree (-A) x' = degree A x' \rangle$
 $\langle proof \rangle$

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-add'*:

$p \in \text{keys } (f + g) \implies p \in \text{keys } f \cup \text{keys } g$
 $\langle \text{proof} \rangle$

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

lemma *vars-in-right-only*:

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

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 *polynomial-sum-monom*:

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

shows

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

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

$\langle \text{proof} \rangle$

lemma *vars-mult-monom*:

fixes $p :: \langle \text{int mpoly} \rangle$

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

$\langle \text{proof} \rangle$

term $\langle (x', u, \text{lookup } u \ x', A) \rangle$

lemma *in-mapping-mult-single*:

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

\longleftrightarrow

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

$\langle \text{proof} \rangle$

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*:

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

$\langle \text{proof} \rangle$

lemma *degree-mult-Var'*:

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

$\langle \text{proof} \rangle$

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'*:

$\langle \text{MPoly } (\text{monomial } 1 \ x') \wedge^{(n+1)} = \text{MPoly } (\text{monomial } (1) \ (((\lambda x. x + x') \wedge^n) \ x')) \rangle$

$\langle \text{proof} \rangle$

lemma *MPoly-monomial-power*:

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

$\langle \text{proof} \rangle$

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*:: $'a \Rightarrow ('a \Rightarrow_0 \ 'b :: \{\text{monoid-add, minus, one}\}) \Rightarrow ('a \Rightarrow_0 \ 'b) \ \mathbf{where}$

$\text{decrease-key } k0 \ f = \text{Abs-poly-mapping } (\lambda k. \ \text{if } k = k0 \ \wedge \ \text{lookup } f \ k \neq 0 \ \text{then } \text{lookup } f \ k - 1 \ \text{else } \text{lookup } f \ k)$

$f k$)

lemma *remove-key-lookup*:

$lookup (decrease-key k0 f) k = (if k = k0 \wedge lookup f k \neq 0 \text{ then } lookup f k - 1 \text{ else } lookup f k)$
 $\langle proof \rangle$

lemma *polynomial-split-on-var*:

fixes $p :: \langle 'a :: \{comm-monoid-add, cancel-comm-monoid-add, semiring-0, comm-semiring-1\} mpoly \rangle$

obtains $q r$ **where**

$\langle p = monom (monomial (Suc 0) x') 1 * q + r \rangle$ **and**
 $\langle x' \notin vars r \rangle$

$\langle proof \rangle$

lemma *polynomial-split-on-var2*:

fixes $p :: \langle int mpoly \rangle$

assumes $\langle x' \notin vars s \rangle$

obtains $q r$ **where**

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

$\langle proof \rangle$

lemma *finit-whenI[intro]*:

$\langle finite \{x. (0 :: nat) < (y x)\} \implies finite \{x. 0 < (y x \text{ when } x \neq x')\} \rangle$

$\langle proof \rangle$

lemma *polynomial-split-on-var-diff-sq2*:

fixes $p :: \langle int mpoly \rangle$

obtains $q r s$ **where**

$\langle p = monom (monomial (Suc 0) x') 1 * q + r + s * (monom (monomial (Suc 0) x') 1^2 - monom (monomial (Suc 0) x') 1) \rangle$ **and**

$\langle x' \notin vars r \rangle$ **and**

$\langle x' \notin vars q \rangle$

$\langle proof \rangle$

lemma *polynomial-decomp-alien-var*:

fixes $q A b :: \langle int mpoly \rangle$

assumes

$q: \langle q = A * (monom (monomial (Suc 0) x') 1) + b \rangle$ **and**

$x: \langle x' \notin vars q \rangle \langle x' \notin vars b \rangle$

shows

$\langle A = 0 \rangle$ **and**

$\langle q = b \rangle$

$\langle proof \rangle$

lemma *polynomial-decomp-alien-var2*:

fixes $q A b :: \langle int mpoly \rangle$

assumes

$q: \langle q = A * (monom (monomial (Suc 0) x') 1 + p) + b \rangle$ **and**

$x: \langle x' \notin vars q \rangle \langle x' \notin vars b \rangle \langle x' \notin vars p \rangle$

shows

$\langle A = 0 \rangle$ **and**

$\langle q = b \rangle$

$\langle proof \rangle$

lemma *vars-unE*: $\langle x \in \text{vars } (a * b) \implies (x \in \text{vars } a \implies \text{thesis}) \implies (x \in \text{vars } b \implies \text{thesis}) \implies \text{thesis} \rangle$
 $\langle \text{proof} \rangle$

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

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* ($\text{insert } ?a \ ?S) = \{x. \exists k. x - k *$

$?a \in \text{More-Modules.ideal } ?S\}$, except that it is more useful if we need to take an element of $\text{More-Modules.ideal } (\text{insert } a \ S)$.

lemma *ideal-insert'*:

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

lemma *ideal-mult-right-in*:

$\langle a \in \text{ideal } A \implies a * b \in \text{More-Modules.ideal } A \rangle$
 $\langle \text{proof} \rangle$

lemma *ideal-mult-right-in2*:

$\langle a \in \text{ideal } A \implies b * a \in \text{More-Modules.ideal } A \rangle$
 $\langle \text{proof} \rangle$

lemma [*simp*]: $\langle \text{vars } (\text{Var } x :: 'a :: \{\text{zero-neq-one}\} \text{mpoly}) = \{x\} \rangle$

$\langle \text{proof} \rangle$

lemma *vars-minus-Var-subset*:

$\langle \text{vars } (p' - \text{Var } x :: 'a :: \{\text{ab-group-add,one,zero-neq-one}\} \text{mpoly}) \subseteq \mathcal{V} \implies \text{vars } p' \subseteq \text{insert } x \ \mathcal{V} \rangle$
 $\langle \text{proof} \rangle$

lemma *vars-add-Var-subset*:

$\langle \text{vars } (p' + \text{Var } x :: 'a :: \{\text{ab-group-add,one,zero-neq-one}\} \text{mpoly}) \subseteq \mathcal{V} \implies \text{vars } p' \subseteq \text{insert } x \ \mathcal{V} \rangle$
 $\langle \text{proof} \rangle$

lemma *coeff-monomila-in-varsD*:

$\langle \text{coeff } p \ (\text{monomial } (\text{Suc } 0) \ x) \neq 0 \implies x \in \text{vars } (p :: \text{int } \text{mpoly}) \rangle$
 $\langle \text{proof} \rangle$

lemma *coeff-MPoly-monomial*[*simp*]:

$\langle (\text{MPoly-Type.coeff } (\text{MPoly } (\text{monomial } a \ m)) \ m) = a \rangle$
 $\langle \text{proof} \rangle$

end

theory *Finite-Map-Multiset*

imports

HOL-Library.Finite-Map

Nested-Multisets-Ordinals.Duplicate-Free-Multiset

begin

notation *image-mset* (**infixr** $\#$ 90)

3 Finite maps and multisets

3.1 Finite sets and multisets

abbreviation *mset-fset* :: $\langle 'a \ \text{fset} \Rightarrow 'a \ \text{multiset} \rangle$ **where**

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

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

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

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

⟨proof⟩

lemma *mset-fset-fset-mset[simp]*:
⟨mset-fset (fset-mset N) = remdups-mset N⟩
⟨proof⟩

lemma *in-mset-fset-fmmember[simp]*: ⟨x ∈# mset-fset N ⟷ x |∈| N⟩
⟨proof⟩

lemma *in-fset-mset-mset[simp]*: ⟨x |∈| fset-mset N ⟷ x ∈# N⟩
⟨proof⟩

3.2 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*
⟨dom-m N = mset-fset (fmdom N)⟩

definition *ran-m where*
⟨ran-m N = the '# fmlookup N '# dom-m N⟩

lemma *dom-m-fmdrop[simp]*: ⟨dom-m (fmdrop C N) = remove1-mset C (dom-m N)⟩
⟨proof⟩

lemma *dom-m-fmdrop-All*: ⟨dom-m (fmdrop C N) = removeAll-mset C (dom-m N)⟩
⟨proof⟩

lemma *dom-m-fmupd[simp]*: ⟨dom-m (fmupd k C N) = add-mset k (remove1-mset k (dom-m N))⟩
⟨proof⟩

lemma *distinct-mset-dom*: ⟨distinct-mset (dom-m N)⟩
⟨proof⟩

lemma *in-dom-m-lookup-iff*: ⟨C ∈# dom-m N' ⟷ fmlookup N' C ≠ None⟩
⟨proof⟩

lemma *in-dom-in-ran-m[simp]*: ⟨i ∈# dom-m N ⟹ the (fmlookup N i) ∈# ran-m N⟩
⟨proof⟩

lemma *fmupd-same[simp]*:
⟨x1 ∈# dom-m x1aa ⟹ fmupd x1 (the (fmlookup x1aa x1)) x1aa = x1aa⟩
⟨proof⟩

lemma *ran-m-fmempty[simp]*: ⟨ran-m fmempty = {#}⟩ **and**
dom-m-fmempty[simp]: ⟨dom-m fmempty = {#}⟩
⟨proof⟩

lemma *fmrestrict-set-fmupd*:
⟨a ∈ xs ⟹ fmrestrict-set xs (fmupd a C N) = fmupd a C (fmrestrict-set xs N)⟩
⟨a ∉ xs ⟹ fmrestrict-set xs (fmupd a C N) = fmrestrict-set xs N⟩
⟨proof⟩

lemma *fset-fmdom-fmrestrict-set*:

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

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

$\langle \text{proof} \rangle$

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

$\langle \text{proof} \rangle$

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 $\langle \text{fmdrop } k \ (\text{fmupd } k \ v \ N'') = \text{fmdrop } k \ N'' \rangle$

$\langle \text{proof} \rangle$

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

⟨proof⟩

lemma *ran-m-mapsto-upd*:

assumes

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

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

⟨proof⟩

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

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

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

⟨proof⟩

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

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

⟨proof⟩

lemma *filter-mset-cong2*:

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

⟨proof⟩

lemma *ran-m-fmdrop*:

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

⟨proof⟩

lemma *ran-m-fmdrop-notin*:

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

⟨proof⟩

lemma *ran-m-fmdrop-If*:

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

⟨proof⟩

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

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

⟨proof⟩

end

theory *WB-Sort*

imports

Refine-Imperative-HOL.IICF

HOL-Library.Rewrite

Nested-Multisets-Ordinals.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**
 $\langle \text{choose-pivot} \ _ \ _ \ _ \ lo \ hi = \text{SPEC}(\lambda k. k \geq lo \wedge k \leq hi) \rangle$

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**

$\langle \text{isPartition-wrt } R \text{ } xs \text{ } lo \text{ } hi \text{ } p \equiv (\forall i. i \geq lo \wedge i < p \longrightarrow R (xs!i) (xs!p)) \wedge (\forall j. j > p \wedge j \leq hi \longrightarrow R (xs!p) (xs!j)) \rangle$

lemma *isPartition-wrtI*:

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

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

$\langle \text{isPartition } xs \text{ } lo \text{ } hi \text{ } p \equiv \text{isPartition-wrt } (\leq) \text{ } xs \text{ } lo \text{ } hi \text{ } p \rangle$

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

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

lemma *isPartition-map-def'*:

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

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

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

$\langle \text{proof} \rangle$

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

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

lemma *take-Suc0*:

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

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

$\langle \text{proof} \rangle$

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

$\langle \text{proof} \rangle$

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

$\langle \text{proof} \rangle$

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

$\langle \text{proof} \rangle$

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

$\langle \text{proof} \rangle$

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

$\langle \text{proof} \rangle$

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

definition *sorted-sublist* :: $\langle 'a :: \text{linorder list} \Rightarrow \text{nat} \Rightarrow \text{nat} \Rightarrow \text{bool} \rangle$ **where**
 $\langle \text{sorted-sublist } xs \text{ } lo \text{ } hi = \text{sorted-sublist-wrt } (\leq) \text{ } xs \text{ } lo \text{ } hi \rangle$

abbreviation *sorted-sublist-map* :: $\langle 'b \Rightarrow 'b \Rightarrow \text{bool} \rangle \Rightarrow ('a \Rightarrow 'b) \Rightarrow 'a \text{ list} \Rightarrow \text{nat} \Rightarrow \text{nat} \Rightarrow \text{bool} \rangle$
where
 $\langle \text{sorted-sublist-map } R \text{ } h \text{ } xs \text{ } lo \text{ } hi \equiv \text{sorted-sublist-wrt } (\lambda a \text{ } b. R \text{ } (h \text{ } a) \text{ } (h \text{ } b)) \text{ } xs \text{ } lo \text{ } hi \rangle$

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

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

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

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

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

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

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

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

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

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

Elements in a sorted sublists are actually sorted

lemma *sorted-sublist-wrt-nth-le*:
assumes $\langle \text{sorted-sublist-wrt } R \text{ } xs \text{ } lo \text{ } hi \rangle$ **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 $\langle R \text{ } (xs!i) \text{ } (xs!j) \rangle$
 $\langle \text{proof} \rangle$

We can make the assumption $i < j$ weaker if we have a reflexivie 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**

$\langle \text{sorted-sublist-wrt } R\ xs\ (lo+1)\ hi \rangle$ **and**

$\langle lo \leq hi \rangle$ **and** $\langle hi < \text{length } xs \rangle$ **and** $\langle R\ (xs!lo)\ (xs!(lo+1)) \rangle$

shows $\langle \text{sorted-sublist-wrt } R\ xs\ lo\ hi \rangle$

$\langle \text{proof} \rangle$

lemma *sorted-sublist-map-cons*:

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

$\langle \text{proof} \rangle$

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

$\langle \text{proof} \rangle$

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

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

$\langle \text{proof} \rangle$

lemma *sorted-sublist-wrt-snoc*:

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

$\langle \text{sorted-sublist-wrt } R\ xs\ lo\ (hi-1) \rangle$ **and**

$\langle lo \leq hi \rangle$ **and** $\langle hi < \text{length } xs \rangle$ **and** $\langle R\ (xs!(hi-1))\ (xs!hi) \rangle$

shows $\langle \text{sorted-sublist-wrt } R\ xs\ lo\ hi \rangle$

⟨proof⟩

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

⟨proof⟩

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

⟨proof⟩

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

lemma *isPartition-wrt-trans*:

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

lemma *isPartition-map-trans*:

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

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

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

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

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

The main theorem to merge sorted lists

lemma *merge-sorted-wrt-partitions*:

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

theorem *merge-sorted-map-partitions*:

$\langle (\bigwedge x y z. \llbracket R (h x) (h y); R (h y) (h z) \rrbracket \implies R (h x) (h z) \implies$
 $\text{isPartition-map } R \text{ h xs lo hi } p \implies$
 $\text{sorted-sublist-map } R \text{ h xs lo } (p - \text{Suc } 0) \implies \text{sorted-sublist-map } R \text{ h xs } (\text{Suc } p) \text{ hi} \implies$
 $lo \leq hi \implies lo \leq p \implies p \leq hi \implies hi < \text{length } xs \implies$

sorted-sublist-map R h xs lo hi
 ⟨proof⟩

lemma *partition-wrt-extend*:

⟨*isPartition-wrt R xs lo' hi' p* \implies
 $hi < \text{length } xs \implies$
 $lo \leq lo' \implies lo' \leq hi \implies hi' \leq hi \implies$
 $lo' \leq p \implies p \leq hi' \implies$
 $(\bigwedge i. lo \leq i \implies i < lo' \implies R (xs!i) (xs!p)) \implies$
 $(\bigwedge j. hi' < j \implies j \leq hi \implies R (xs!p) (xs!j)) \implies$
isPartition-wrt R xs lo hi p⟩
 ⟨proof⟩

lemma *partition-map-extend*:

⟨*isPartition-map R h xs lo' hi' p* \implies
 $hi < \text{length } xs \implies$
 $lo \leq lo' \implies lo' \leq hi \implies hi' \leq hi \implies$
 $lo' \leq p \implies p \leq hi' \implies$
 $(\bigwedge i. lo \leq i \implies i < lo' \implies R (h (xs!i)) (h (xs!p))) \implies$
 $(\bigwedge j. hi' < j \implies j \leq hi \implies R (h (xs!p)) (h (xs!j))) \implies$
isPartition-map R h xs lo hi p⟩
 ⟨proof⟩

lemma *isPartition-empty*:

⟨ $(\bigwedge j. \llbracket lo < j; j \leq hi \rrbracket \implies R (xs ! lo) (xs ! j)) \implies$
isPartition-wrt R xs lo hi lo⟩
 ⟨proof⟩

lemma *take-ext*:

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

lemma *drop-ext'*:

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

lemma *drop-ext*:

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

lemma *sublist-ext'*:

⟨ $(\forall i. lo \leq i \wedge i \leq hi \longrightarrow xs^!i = xs!i) \implies$
 $\text{length } xs' = \text{length } xs \implies$ ⟩

$lo \leq hi \implies Suc\ hi < length\ xs \implies$
 $sublist\ xs'\ lo\ hi = sublist\ xs\ lo\ hi$
 ⟨proof⟩

lemma *lt-Suc*: $\langle (a < b) = (Suc\ a = b \vee Suc\ a < b) \rangle$
 ⟨proof⟩

lemma *sublist-until-end-eq-drop*: $\langle Suc\ hi = length\ xs \implies sublist\ xs\ lo\ hi = drop\ lo\ xs \rangle$
 ⟨proof⟩

lemma *sublist-ext*:
 $\langle (\forall i. lo \leq i \wedge i \leq hi \longrightarrow xs!\ i = xs!\ i) \implies$
 $length\ xs' = length\ xs \implies$
 $lo \leq hi \implies hi < length\ xs \implies$
 $sublist\ xs'\ lo\ hi = sublist\ xs\ lo\ hi \rangle$
 ⟨proof⟩

lemma *sorted-wrt-lower-sublist-still-sorted*:
assumes $\langle sorted\ sublist\ wrt\ R\ xs\ lo\ (lo' - Suc\ 0) \rangle$ **and**
 $\langle lo \leq lo' \rangle$ **and** $\langle lo' < length\ xs \rangle$ **and**
 $\langle (\forall i. lo \leq i \wedge i < lo' \longrightarrow xs!\ i = xs!\ i) \rangle$ **and** $\langle length\ xs' = length\ xs \rangle$
shows $\langle sorted\ sublist\ wrt\ R\ xs'\ lo\ (lo' - Suc\ 0) \rangle$
 ⟨proof⟩

lemma *sorted-map-lower-sublist-still-sorted*:
assumes $\langle sorted\ sublist\ map\ R\ h\ xs\ lo\ (lo' - Suc\ 0) \rangle$ **and**
 $\langle lo \leq lo' \rangle$ **and** $\langle lo' < length\ xs \rangle$ **and**
 $\langle (\forall i. lo \leq i \wedge i < lo' \longrightarrow xs!\ i = xs!\ i) \rangle$ **and** $\langle length\ xs' = length\ xs \rangle$
shows $\langle sorted\ sublist\ map\ R\ h\ xs'\ lo\ (lo' - Suc\ 0) \rangle$
 ⟨proof⟩

lemma *sorted-wrt-upper-sublist-still-sorted*:
assumes $\langle sorted\ sublist\ wrt\ R\ xs\ (hi' + 1)\ hi \rangle$ **and**
 $\langle lo \leq lo' \rangle$ **and** $\langle hi < length\ xs \rangle$ **and**
 $\langle \forall j. hi' < j \wedge j \leq hi \longrightarrow xs!\ j = xs!\ j \rangle$ **and** $\langle length\ xs' = length\ xs \rangle$
shows $\langle sorted\ sublist\ wrt\ R\ xs'\ (hi' + 1)\ hi \rangle$
 ⟨proof⟩

lemma *sorted-map-upper-sublist-still-sorted*:
assumes $\langle sorted\ sublist\ map\ R\ h\ xs\ (hi' + 1)\ hi \rangle$ **and**
 $\langle lo \leq lo' \rangle$ **and** $\langle hi < length\ xs \rangle$ **and**
 $\langle \forall j. hi' < j \wedge j \leq hi \longrightarrow xs!\ j = xs!\ j \rangle$ **and** $\langle length\ xs' = length\ xs \rangle$
shows $\langle sorted\ sublist\ map\ R\ h\ xs'\ (hi' + 1)\ hi \rangle$
 ⟨proof⟩

The specification of the partition function

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

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

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

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

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

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 $\langle \text{set } (\text{sublist } xs' \text{ } lo \text{ } hi) \subseteq \text{set } (\text{sublist } xs \text{ } lo \text{ } hi) \rangle$

$\langle \text{proof} \rangle$

lemma *mset-sublist-eq*:

assumes $\langle \text{mset } xs' = \text{mset } xs \rangle$

and $\langle \bigwedge i. i < lo \implies xs ! i = xs ! i \rangle$

and $\langle \bigwedge j. \llbracket hi < j; j < \text{length } xs \rrbracket \implies xs ! j = xs ! j \rangle$

and $\text{bounds}: \langle lo \leq hi \rangle \langle hi < \text{length } xs \rangle$

shows $\langle \text{set } (\text{sublist } xs' \text{ } lo \text{ } hi) = \text{set } (\text{sublist } xs \text{ } lo \text{ } hi) \rangle$

$\langle \text{proof} \rangle$

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

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

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

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

$\text{ASSERT}(lo \leq hi \wedge hi < \text{length } xs \wedge \text{mset } xs = \text{mset } xs0);$ — Premise for a partition function

$(xs, p) \leftarrow \text{SPEC}(\text{uncurry } (\text{partition-spec } R \text{ } h \text{ } xs \text{ } lo \text{ } hi));$ — Abstract partition function

$\text{ASSERT}(\text{mset } xs = \text{mset } xs0);$

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

$\text{ASSERT}(\text{mset } xs = \text{mset } xs0);$

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

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

$\}) \rangle$

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

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

$\langle \text{quicksort-pre } R \text{ } h \text{ } xs0 \text{ } lo \text{ } hi \text{ } xs \equiv lo \leq hi \wedge hi < \text{length } xs \wedge \text{mset } xs = \text{mset } xs0 \rangle$

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

$\langle \text{quicksort-post } R \ h \ lo \ hi \ xs \ xs' \equiv$
 $\text{mset } xs' = \text{mset } xs \wedge$
 $\text{sorted-sublist-map } R \ h \ xs' \ lo \ hi \wedge$
 $(\forall i. i < lo \longrightarrow xs!i = xs!i) \wedge$
 $(\forall j. hi < j \wedge j < \text{length } xs \longrightarrow xs!j = xs!j) \rangle$

Convert Pure to HOL

lemma *quicksort-postI*:

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

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

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 $\langle \text{quicksort-pre } R \ h \ xs0 \ lo \ (p - Suc \ 0) \ xs' \rangle$

<proof>

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 $\langle \text{quicksort-post } R \ h \ lo \ hi \ xs \ xs'' \rangle$

<proof>

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 $\langle \text{quicksort-pre } R \ h \ xs0 \ (Suc \ p) \ hi \ xs'' \rangle$

<proof>

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 $\langle \text{quicksort-post } R \ h \ lo \ hi \ xs \ xs''' \rangle$

<proof>

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

<proof>

definition *partition-main-inv* :: $\langle ('b \Rightarrow 'b \Rightarrow \text{bool}) \Rightarrow ('a \Rightarrow 'b) \Rightarrow \text{nat} \Rightarrow \text{nat} \Rightarrow 'a \text{ list} \Rightarrow (\text{nat} \times \text{nat} \times 'a \text{ list}) \Rightarrow \text{bool} \rangle$ **where**
 $\langle \text{partition-main-inv } R \text{ h lo hi xs0 } p \equiv$
 case p of $(i, j, xs) \Rightarrow$
 $j < \text{length } xs \wedge j \leq \text{hi} \wedge i < \text{length } xs \wedge \text{lo} \leq i \wedge i \leq j \wedge \text{mset } xs = \text{mset } xs0 \wedge$
 $(\forall k. k \geq \text{lo} \wedge k < i \longrightarrow R (h (xs!k)) (h (xs!hi))) \wedge$ — 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
 $(\forall k. k < \text{lo} \longrightarrow xs!k = xs0!k) \wedge$ — Everything below lo is unchanged
 $(\forall k. k \geq j \wedge k < \text{length } xs \longrightarrow xs!k = xs0!k)$ — All elements from j are unchanged (including everything above hi)
 \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* :: $\langle ('b \Rightarrow 'b \Rightarrow \text{bool}) \Rightarrow ('a \Rightarrow 'b) \Rightarrow \text{nat} \Rightarrow \text{nat} \Rightarrow 'a \text{ list} \Rightarrow ('a \text{ list} \times \text{nat}) \text{ nres} \rangle$ **where**
 $\langle \text{partition-main } R \text{ h lo hi xs0} = \text{do } \{$
 $\text{ASSERT}(\text{hi} < \text{length } xs0);$
 $\text{pivot} \leftarrow \text{RETURN } (h (xs0 ! \text{hi}));$
 $(i, j, xs) \leftarrow \text{WHILE}_T^{\text{partition-main-inv } R \text{ h lo hi xs0}}$ — We loop from $j = \text{lo}$ to $j = \text{hi} - (1::'c)$.
 $(\lambda(i, j, xs). j < \text{hi})$
 $(\lambda(i, j, xs). \text{do } \{$
 $\text{ASSERT}(i < \text{length } xs \wedge j < \text{length } xs);$
 $\text{if } R (h (xs!j)) \text{ pivot}$
 $\text{then } \text{RETURN } (i+1, j+1, \text{swap } xs \ i \ j)$
 $\text{else } \text{RETURN } (i, j+1, xs)$
 $\})$
 $(\text{lo}, \text{lo}, xs0);$ — i and j are both initialized to lo
 $\text{ASSERT}(i < \text{length } xs \wedge j = \text{hi} \wedge \text{lo} \leq i \wedge \text{hi} < \text{length } xs \wedge \text{mset } xs = \text{mset } xs0);$
 $\text{RETURN } (\text{swap } xs \ i \ \text{hi}, i)$
 $\}$
 \rangle

lemma *partition-main-correct*:

assumes *bounds*: $\langle \text{hi} < \text{length } xs \rangle \langle \text{lo} \leq \text{hi} \rangle$ **and**
trans: $\langle \bigwedge x \ y \ z. \llbracket R (h \ x) (h \ y); R (h \ y) (h \ z) \rrbracket \implies R (h \ x) (h \ z) \rangle$ **and** *lin*: $\langle \bigwedge x \ y. R (h \ x) (h \ y) \vee R (h \ y) (h \ x) \rangle$
shows $\langle \text{partition-main } R \text{ h lo hi xs} \leq \text{SPEC}(\lambda(xs', p). \text{mset } xs = \text{mset } xs' \wedge$
 $\text{lo} \leq p \wedge p \leq \text{hi} \wedge \text{isPartition-map } R \text{ h } xs' \ \text{lo hi } p \wedge (\forall i. i < \text{lo} \longrightarrow xs!i = xs!i) \wedge (\forall i. \text{hi} < i \wedge i < \text{length } xs' \longrightarrow xs!i = xs!i) \rangle$
 $\langle \text{proof} \rangle$

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

$\langle \text{partition-between } R \text{ h lo hi xs0} = \text{do } \{$
 $\text{ASSERT}(\text{hi} < \text{length } xs0 \wedge \text{lo} \leq \text{hi});$
 $k \leftarrow \text{choose-pivot } R \text{ h } xs0 \ \text{lo hi};$ — choice of pivot

```

  ASSERT(k < length xs0);
  xs ← RETURN (swap xs0 k hi); — move the pivot to the last position, before we start the actual
loop
  ASSERT(length xs = length xs0);
  partition-main R h lo hi xs
}⟩

```

lemma *partition-between-correct*:

```

assumes ⟨hi < length xs⟩ and ⟨lo ≤ hi⟩ and
⟨∧ x y z. [R (h x) (h y); R (h y) (h z)] ⇒ R (h x) (h z)⟩ and ⟨∧ x y. R (h x) (h y) ∨ R (h y) (h x)⟩
shows ⟨partition-between R h lo hi xs ≤ SPEC(uncurry (partition-spec R h xs lo hi))⟩
⟨proof⟩

```

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

definition *choose-pivot3* **where**

```

⟨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 ∧ R mid end) ∨ (R end mid ∧ R mid start) then RETURN k
  else if (R start end ∧ R end mid) ∨ (R mid end ∧ R end start) then RETURN hi
  else RETURN lo
}⟩

```

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

lemma *choose-pivot3-choose-pivot*:

```

assumes ⟨lo < length xs⟩ ⟨hi < length xs⟩ ⟨hi ≥ lo⟩
shows ⟨choose-pivot3 R h xs lo hi ≤ ↓ Id (choose-pivot R h xs lo hi)⟩
⟨proof⟩

```

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

definition *partition-between-ref*

```

:: ⟨('b ⇒ 'b ⇒ bool) ⇒ ('a ⇒ 'b) ⇒ nat ⇒ nat ⇒ 'a list ⇒ ('a list × nat) nres)⟩

```

where

```

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

```

lemma *partition-main-ref'*:

```

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

```

$\langle \text{proof} \rangle$

lemma *Down-id-eq*:

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

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'*:

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

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

Refined quicksort algorithm: We use the refined partition function.

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

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

$\text{do } \{$

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

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

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

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

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

$\text{ASSERT}(\text{mset } xs = \text{mset } xs0);$

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

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

$\}) \rangle$

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*:

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

$(\bigwedge x x'. P x' \implies (x, x') \in A \implies$
 $f x \leq \Downarrow B (g x'))$
 ⟨proof⟩

lemma *quicksort-ref-quicksort*:

assumes *bounds*: ⟨*hi* < *length xs*⟩ ⟨*lo* ≤ *hi*⟩ **and**

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

shows ⟨*quicksort-ref* *R h x0* ≤ \Downarrow *Id* (*quicksort R h x0*)⟩

⟨proof⟩

definition *full-quicksort* **where**

⟨*full-quicksort R h xs* ≡ if *xs* = [] then *RETURN xs* else *quicksort R h* (*0*, *length xs* - 1, *xs*)⟩

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

⟨*full-quicksort-ref R h xs* ≡

if *List.null xs* then *RETURN xs*

else *quicksort-ref R h* (*0*, *length xs* - 1, *xs*)⟩

definition *full-quicksort-impl* :: ⟨*nat list* ⇒ *nat list nres*⟩ **where**

⟨*full-quicksort-impl xs* = *full-quicksort-ref* (≤) *id xs*⟩

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

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

shows ⟨(*full-quicksort-ref R h*, *full-quicksort R h*) ∈

⟨*Id*⟩*list-rel* →_{*f*} ⟨⟨*Id*⟩*list-rel*⟩*nres-rel*⟩

⟨proof⟩

lemma *sublist-entire*:

⟨*sublist xs 0* (*length xs* - 1) = *xs*⟩

⟨proof⟩

lemma *sorted-sublist-wrt-entire*:

assumes ⟨*sorted-sublist-wrt R xs 0* (*length xs* - 1)⟩

shows ⟨*sorted-wrt R xs*⟩

⟨proof⟩

lemma *sorted-sublist-map-entire*:

assumes ⟨*sorted-sublist-map R h xs 0* (*length xs* - 1)⟩

shows ⟨*sorted-wrt* (λ *x y. R (h x) (h y)) xs*⟩

⟨proof⟩

Final correctness lemma

theorem *full-quicksort-correct-sorted*:

assumes

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

shows ⟨*full-quicksort R h xs* ≤ \Downarrow *Id* (*SPEC*(λ *xs'*. *mset xs'* = *mset xs* ∧ *sorted-wrt* (λ *x y. R (h x) (h y)) xs'))*⟩

⟨proof⟩

lemma *full-quicksort-correct*:

assumes

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

lin: $\langle \bigwedge x y. R(h x)(h y) \vee R(h y)(h x) \rangle$

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

$\langle \text{proof} \rangle$

end

theory *More-Loops*

imports

Refine-Monadic.Refine-While

Refine-Monadic.Refine-Foreach

HOL-Library.Rewrite

begin

3.3 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*:

$b \geq a \implies$

$do \{$

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

$([a..<b], \sigma);$

$\text{RETURN } \sigma$

$\} \leq do \{$

$(-, \sigma) \leftarrow \text{WHILE}_T(\lambda(i, x). i < b \wedge c \ x) (\lambda(i, x). do \{ \text{ASSERT}(i < b); \sigma' \leftarrow f \ i \ x; \text{RETURN}(i+1, \sigma') \}) (a, \sigma);$

$\text{RETURN } \sigma$

$\}$

$\langle \text{proof} \rangle$

lemma *while-upt-while-direct2*:

$b \geq a \implies$

$do \{$

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

$([a..<b], \sigma);$

$\text{RETURN } \sigma$

$\} \geq do \{$

$(-, \sigma) \leftarrow \text{WHILE}_T(\lambda(i, x). i < b \wedge c \ x) (\lambda(i, x). do \{ \text{ASSERT}(i < b); \sigma' \leftarrow f \ i \ x; \text{RETURN}(i+1, \sigma') \}) (a, \sigma);$

$\text{RETURN } \sigma$

$\}$

$\langle \text{proof} \rangle$

lemma *while-upt-while-direct*:

$b \geq a \implies$

$do \{$

```

  (-,σ) ← WHILET (FOREACH-cond c) (λx. do {ASSERT (FOREACH-cond c x); FOREACH-body
f x})
  ([a..<b],σ);
  RETURN σ
} = do {
  (-,σ) ← WHILET (λ(i, x). i < b ∧ c x) (λ(i, x). do {ASSERT (i < b); σ'←f i x; RETURN (i+1,σ')
}) (a,σ);
  RETURN σ
}
⟨proof⟩

```

lemma *while-nfoldli*:

```

do {
  (-,σ) ← WHILET (FOREACH-cond c) (λx. do {ASSERT (FOREACH-cond c x); FOREACH-body
f x}) (l,σ);
  RETURN σ
} ≤ nfoldli l c f σ
⟨proof⟩

```

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

```

≤
(WHILETI
  (FOREACH-cond c) (λx. do {ASSERT (FOREACH-cond c x); FOREACH-body f x}) (l, σ)
)
≫
(λ(-, σ). RETURN σ)
⟨proof⟩

```

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

```

  (-,σ) ← WHILET (FOREACH-cond c) (λx. do {ASSERT (FOREACH-cond c x); FOREACH-body
f x}) (l,σ);
  RETURN σ
} = nfoldli l c f σ
⟨proof⟩

```

end

theory *PAC-Specification*

imports *PAC-More-Poly*

begin

4 Specification of the PAC checker

4.1 Ideals

type-synonym *int-poly* = $\langle int\ mpoly \rangle$

definition *polynomial-bool* :: $\langle int-poly\ set \rangle$ **where**

$\langle polynomial-bool = (\lambda c. Var\ c\ \wedge\ 2 - Var\ c) \text{ ' UNIV} \rangle$

definition *pac-ideal* **where**

$\langle pac-ideal\ A \equiv ideal\ (A \cup polynomial-bool) \rangle$

lemma *X2-X-in-pac-ideal*:

$\langle Var\ c\ \wedge\ 2 - Var\ c \in pac-ideal\ A \rangle$

$\langle proof \rangle$

lemma *pac-idealI1* [*intro*]:

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

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

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

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

The equality on ideals is restricted to polynomials 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*:

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

lemma *pac-ideal-mono*:

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

4.2 PAC Format

The PAC format contains three kind of steps:

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

To model the simplification that happens, we add the $p - p' \in \text{polynomial-bool}$ 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

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

mult:

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

if

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

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

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

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 process we have in mind when we write the specification, but we must add it explicitly 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:

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

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

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

$\langle \bigwedge p' x r.$

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

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

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**

$p\text{-x-coeff}: \langle \text{coeff } p (\text{monomial } (\text{Suc } 0) x') = 1 \rangle$ **and**

$\text{vars-}q: \langle \text{vars } q \subseteq \mathcal{V} \rangle$ **and**

$q: \langle q \in \text{More-Modules.ideal } (\text{insert } p (\text{set-mset } A \cup \text{polynomial-bool})) \rangle$ **and**

leading: $\langle x' \notin \text{vars } (p - \text{Var } x') \rangle$ **and**

diff: $\langle (\text{Var } x' - p)^2 - (\text{Var } x' - p) \in \text{More-Modules.ideal polynomial-bool} \rangle$

shows

$\langle q \in \text{More-Modules.ideal } (\text{set-mset } A \cup \text{polynomial-bool}) \rangle$

$\langle \text{proof} \rangle$

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**

$p\text{-x-coeff}: \langle \text{coeff } p (\text{monomial } (\text{Suc } 0) x') = -1 \rangle$ **and**

$\text{vars-}q: \langle \text{vars } q \subseteq \mathcal{V} \rangle$ **and**

$q: \langle q \in \text{More-Modules.ideal } (\text{insert } p (\text{set-mset } A \cup \text{polynomial-bool})) \rangle$ **and**

leading: $\langle x' \notin \text{vars } (p + \text{Var } x') \rangle$ **and**

diff: $\langle (\text{Var } x' + p)^{\wedge 2} - (\text{Var } x' + p) \in \text{More-Modules.ideal polynomial-bool} \rangle$

shows

$\langle q \in \text{More-Modules.ideal } (\text{set-mset } A \cup \text{polynomial-bool}) \rangle$

$\langle \text{proof} \rangle$

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:*

fixes $p :: \langle \text{int mpoly} \rangle$

assumes $\langle x \notin \text{vars } p \rangle$

shows $\langle \text{vars } (p - \text{Var } x) = \text{insert } x (\text{vars } p) \rangle$

$\langle \text{proof} \rangle$

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*:

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

lemma *coeff-add-left-notin*:

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

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

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

$\langle \text{proof} \rangle$

lemma *PAC-Format-subset-ideal*:

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

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

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

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

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

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

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 $(\text{'a}, \text{'b}) \text{ fmap}$, that are nicer to use especially here where everything is finite.

definition *fmap-rel* **where**

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

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

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

$\text{fset } (\text{fmdom } m1) \subseteq \text{Domain } K \wedge \text{fset } (\text{fmdom } m2) \subseteq \text{Range } K \wedge$

$(\forall i j. (i, j) \in K \longrightarrow j \in | \text{fmdom } m2 \longleftrightarrow i \in | \text{fmdom } m1)\}$

lemma *fmap-rel-alt-def*:

$\langle \langle K, V \rangle \text{fmap-rel} \equiv$
 $\{ (m1, m2). \langle \langle \forall i j. i \in \# \text{ dom-m } m2 \longrightarrow$
 $(j, i) \in K \longrightarrow (\text{the } (\text{fmlookup } m1 \ j), \text{the } (\text{fmlookup } m2 \ i)) \in V \rangle \wedge$
 $\text{fset } (\text{fmdom } m1) \subseteq \text{Domain } K \wedge$
 $\text{fset } (\text{fmdom } m2) \subseteq \text{Range } K \wedge$
 $(\forall i j. (i, j) \in K \longrightarrow (j \in \# \text{ dom-m } m2) = (i \in \# \text{ dom-m } m1)) \rangle \}$
 \rangle
 $\langle \text{proof} \rangle$

lemma *fmdom-empty-fmempty-iff*[simp]: $\langle \text{fmdom } m = \{\} \longleftrightarrow m = \text{fmempty} \rangle$
 $\langle \text{proof} \rangle$

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

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 *remove1-mset-eq-add-mset-iff*:
 $\langle \text{remove1-mset } a \ A = \text{add-mset } a \ A' \longleftrightarrow A = \text{add-mset } a \ (\text{add-mset } a \ A') \rangle$
 $\langle \text{proof} \rangle$

lemma *fmap-rel-fmdrop-fmap-rel*:
 $\langle (\text{fmdrop } p \ A, \text{fmdrop } p' \ B) \in \langle K, R \rangle \text{fmap-rel} \rangle$
if *single*: *single-valued* K *single-valued* (K^{-1}) **and**
 $H0: \langle (A, B) \in \langle K, R \rangle \text{fmap-rel} \rangle \langle (p, p') \in K \rangle$
 $\langle \text{proof} \rangle$

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)*
 ⟨*proof*⟩

lemma *fmap-rel-nat-the-fmlookup[intro]*:
 ⟨(A, B) ∈ ⟨S, R⟩fmap-rel ⇒ (p, p') ∈ S ⇒ p' ∈# dom-m B ⇒
 (the (fmlookup A p), the (fmlookup B p')) ∈ R⟩
 ⟨*proof*⟩

lemma *fmap-rel-in-dom-iff*:
 ⟨(aa, a'a) ∈ ⟨K, V⟩fmap-rel ⇒
 (a, a') ∈ K ⇒
 a' ∈# dom-m a'a ⇔
 a ∈# dom-m aa⟩
 ⟨*proof*⟩

lemma *fmap-rel-fmlookup-rel*:
 ⟨(a, a') ∈ K ⇒ (aa, a'a) ∈ ⟨K, V⟩fmap-rel ⇒
 (fmlookup aa a, fmlookup a'a a') ∈ ⟨V⟩option-rel⟩
 ⟨*proof*⟩

sempref-decl-op *fmap-lookup*: *fmlookup* :: ⟨K, V⟩fmap-rel → K → ⟨V⟩option-rel
 ⟨*proof*⟩

lemma *in-fdom-alt*: *k* ∈# dom-m *m* ⇔ ¬is-None (fmlookup *m* *k*)
 ⟨*proof*⟩

sempref-decl-op *fmap-contains-key*: λ*k m*. *k* ∈# dom-m *m* :: K → ⟨K, V⟩fmap-rel → bool-rel
 ⟨*proof*⟩

5.2 Patterns

lemma *pat-fmap-empty[pat-rules]*: *fmempty* ≡ *op-fmap-empty* ⟨*proof*⟩

lemma *pat-map-is-empty[pat-rules]*:
 (=) \$*m*\$*fmempty* ≡ *op-fmap-is-empty*\$*m*
 (=) \$*fmempty*\$*m* ≡ *op-fmap-is-empty*\$*m*
 (=) \$(dom-m\$*m*)\$#{#} ≡ *op-fmap-is-empty*\$*m*
 (=) \$#{#}\$\$(dom-m\$*m*) ≡ *op-fmap-is-empty*\$*m*
 ⟨*proof*⟩

lemma *op-map-contains-key[pat-rules]*:
 (∈#) \$ *k* \$ (dom-m\$*m*) ≡ *op-fmap-contains-key*'*k*'*m*
 ⟨*proof*⟩

5.3 Mapping to Normal Hashmaps

abbreviation *map-of-fmap* :: ⟨('k ⇒ 'v option) ⇒ ('k, 'v) fmap⟩ **where**
 ⟨*map-of-fmap* *h* ≡ *Abs-fmap* *h*⟩

definition *map-fmap-rel* **where**
 ⟨*map-fmap-rel* = *br map-of-fmap* (λ*a*. *finite* (dom *a*))⟩

lemma *fmdrop-set-None*:
 ⟨(*op-map-delete*, *fmdrop*) ∈ Id → *map-fmap-rel* → *map-fmap-rel*⟩
 ⟨*proof*⟩

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

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

definition *fmlookup'* **where**
 $[simp]: \langle fmlookup' A k = fmlookup k A \rangle$

lemma [*def-pat-rules*]:
 $\langle ((\in \#) \$ k \$ (dom\text{-}m \$ A)) \equiv Not \$ (is\text{-}None \$ (fmlookup' \$ k \$ A)) \rangle$
 $\langle proof \rangle$

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

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

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*:

⟨(uncurry0 (RETURN op-map-empty), uncurry0 (RETURN op-iam-fmap-empty)) ∈ unit-rel →_f
 ⟨map-fmap-rel⟩ nres-rel
 ⟨proof⟩

lemmas [*sepref-fr-rules*] =

iam.empty-hnr[FCOMP *fmempty-empty*, *unfolded op-iam-fmap-empty-def*[*symmetric*]]

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

⟨*upper-bound-on-dom* A = SPEC(λn. ∀ i ∈ #(dom-m A). i < n)⟩

lemma [*sepref-fr-rules*]:

⟨((Array.len), *upper-bound-on-dom*) ∈ (iam-fmap-assn nat-assn V)^k →_a nat-assn⟩
 ⟨proof⟩

lemma *fmap-rel-nat-rel-dom-m*[*simp*]:

⟨(A, B) ∈ ⟨nat-rel, R⟩fmap-rel ⇒ dom-m A = dom-m B⟩
 ⟨proof⟩

lemma *ref-two-step'*:

⟨A ≤ B ⇒ ↓ R A ≤ ↓ R B⟩
 ⟨proof⟩

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*:

⟨*is-success* a ↔ a = SUCCESS⟩
 ⟨proof⟩

datatype ('a, 'b, 'lbls) *pac-step* =

Add (pac-src1: 'lbls) (pac-src2: 'lbls) (new-id: 'lbls) (pac-res: 'a) |
Mult (pac-src1: 'lbls) (pac-mult: 'a) (new-id: 'lbls) (pac-res: 'a) |
Extension (new-id: 'lbls) (new-var: 'b) (pac-res: 'a) |
Del (pac-src1: 'lbls)

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

definition $PAC\text{-}checker\text{-}specification$

$:: \langle int\text{-}poly \Rightarrow int\text{-}poly\ multiset \Rightarrow (status \times nat\ set \times int\text{-}poly\ multiset)\ nres \rangle$

where

$\langle PAC\text{-}checker\text{-}specification\ spec\ A = SPEC(\lambda(b, \mathcal{V}, B).$

$(\neg is\text{-}failed\ b \longrightarrow restricted\text{-}ideal\text{-}to_I (\bigcup (vars\ 'set\text{-}mset\ A) \cup vars\ spec)\ B \subseteq restricted\text{-}ideal\text{-}to_I (\bigcup (vars\ 'set\text{-}mset\ A) \cup vars\ spec)\ A) \wedge$
 $(is\text{-}found\ b \longrightarrow spec \in pac\text{-}ideal\ (set\text{-}mset\ A)) \rangle$

definition $PAC\text{-}checker\text{-}specification\text{-}spec$

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

where

$\langle PAC\text{-}checker\text{-}specification\text{-}spec\ spec = (\lambda(\mathcal{V}, A)\ (b, B). (\neg is\text{-}failed\ b \longrightarrow \bigcup (vars\ 'set\text{-}mset\ A) \subseteq \mathcal{V}) \wedge$

$(is\text{-}success\ b \longrightarrow PAC\text{-}Format^{**}(\mathcal{V}, A)\ B) \wedge$

$(is\text{-}found\ b \longrightarrow PAC\text{-}Format^{**}(\mathcal{V}, A)\ B \wedge spec \in pac\text{-}ideal\ (set\text{-}mset\ A)) \rangle$

abbreviation $PAC\text{-}checker\text{-}specification2$

$:: \langle int\text{-}poly \Rightarrow (nat\ set \times int\text{-}poly\ multiset) \Rightarrow (status \times (nat\ set \times int\text{-}poly\ multiset))\ nres \rangle$

where

$\langle PAC\text{-}checker\text{-}specification2\ spec\ A \equiv SPEC(PAC\text{-}checker\text{-}specification\text{-}spec\ spec\ A) \rangle$

definition $PAC\text{-}checker\text{-}specification\text{-}step\text{-}spec$

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

where

$\langle PAC\text{-}checker\text{-}specification\text{-}step\text{-}spec = (\lambda(\mathcal{V}_0, A_0)\ spec\ (\mathcal{V}, A)\ (b, B).$

$(is\text{-}success\ b \longrightarrow$

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

$\bigcup (vars\ 'set\text{-}mset\ A) \subseteq \mathcal{V} \wedge PAC\text{-}Format^{**}(\mathcal{V}_0, A_0)\ (\mathcal{V}, A) \wedge PAC\text{-}Format^{**}(\mathcal{V}, A)\ B) \wedge$

$(is\text{-}found\ b \longrightarrow$

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

$\bigcup (vars\ 'set\text{-}mset\ A) \subseteq \mathcal{V} \wedge PAC\text{-}Format^{**}(\mathcal{V}_0, A_0)\ (\mathcal{V}, A) \wedge PAC\text{-}Format^{**}(\mathcal{V}, A)\ B \wedge$

$spec \in pac\text{-}ideal\ (set\text{-}mset\ A_0)) \rangle$

abbreviation $PAC\text{-}checker\text{-}specification\text{-}step2$

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

where

$\langle PAC\text{-}checker\text{-}specification\text{-}step2\ A_0\ spec\ A \equiv SPEC(PAC\text{-}checker\text{-}specification\text{-}step\text{-}spec\ A_0\ spec\ A) \rangle$

definition $normalize\text{-}poly\text{-}spec :: \langle \rightarrow \rangle$ **where**

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

lemma $normalize\text{-}poly\text{-}spec\text{-}alt\text{-}def:$

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

$\langle proof \rangle$

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

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

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

$\langle check\text{-}add\ A\ \mathcal{V}\ p\ q\ i\ r =$

$SPEC(\lambda b. b \longrightarrow p \in \# dom\text{-}m\ A \wedge q \in \# dom\text{-}m\ A \wedge i \notin \# dom\text{-}m\ A \wedge vars\ r \subseteq \mathcal{V} \wedge$

the (fmlookup A p) + the (fmlookup A q) - r ∈ ideal polynomial-bool)

definition *check-mult* :: ⟨(nat, int mpoly) fmap ⇒ nat set ⇒ nat ⇒ int mpoly ⇒ nat ⇒ int mpoly ⇒ bool nres⟩ **where**

⟨check-mult A V p q i r =
SPEC(λb. b → p ∈# dom-m A ∧ i ∉# dom-m A ∧ vars q ⊆ V ∧ vars r ⊆ V ∧
the (fmlookup A p) * q - r ∈ ideal polynomial-bool)⟩

definition *check-extension* :: ⟨(nat, int mpoly) fmap ⇒ nat set ⇒ nat ⇒ nat ⇒ int mpoly ⇒ (bool) nres⟩ **where**

⟨check-extension A V i v p =
SPEC(λb. b → (i ∉# dom-m A ∧
(v ∉ V ∧
(p + Var v)² - (p + Var v) ∈ ideal polynomial-bool ∧
vars (p + Var v) ⊆ V))⟩

fun *merge-status* **where**

⟨merge-status (FAILED) - = FAILED⟩ |
⟨merge-status - (FAILED) = FAILED⟩ |
⟨merge-status FOUND - = FOUND⟩ |
⟨merge-status - FOUND = FOUND⟩ |
⟨merge-status - - = SUCCESS⟩

type-synonym *fpac-step* = ⟨nat set × (nat, int-poly) fmap⟩

definition *check-del* :: ⟨(nat, int mpoly) fmap ⇒ nat ⇒ bool nres⟩ **where**

⟨check-del A p =
SPEC(λb. b → True)⟩

6.2 Algorithm

definition *PAC-checker-step*

:: ⟨int-poly ⇒ (status × fpac-step) ⇒ (int-poly, nat, nat) pac-step ⇒
(status × fpac-step) nres⟩

where

⟨PAC-checker-step = (λspec (stat, (V, A)) st. case st of
Add - - - ⇒
do {
r ← normalize-poly-spec (pac-res st);
eq ← check-add A V (pac-src1 st) (pac-src2 st) (new-id st) r;
st' ← SPEC(λst'. (¬is-failed st' ∧ is-found st' → r - spec ∈ ideal polynomial-bool));
if eq
then RETURN (merge-status stat st',
V, fmdup (new-id st) r A)
else RETURN (FAILED, (V, A))
}
| Del - ⇒
do {
eq ← check-del A (pac-src1 st);
if eq
then RETURN (stat, (V, fmdrop (pac-src1 st) A))
else RETURN (FAILED, (V, A))
}
| Mult - - - ⇒
do {
r ← normalize-poly-spec (pac-res st);

```

    q ← normalize-poly-spec (pac-mult st);
    eq ← check-mult A  $\mathcal{V}$  (pac-src1 st) q (new-id st) r;
    st' ← SPEC( $\lambda st'. (\neg \text{is-failed } st' \wedge \text{is-found } st' \longrightarrow r - \text{spec} \in \text{ideal polynomial-bool})$ );
    if eq
    then RETURN (merge-status stat st',
       $\mathcal{V}$ , fmupd (new-id st) r A)
    else RETURN (FAILED, ( $\mathcal{V}$ , A))
  }
| Extension - - -  $\Rightarrow$ 
  do {
    r ← normalize-poly-spec (pac-res st - Var (new-var st));
    (eq) ← check-extension A  $\mathcal{V}$  (new-id st) (new-var st) r;
    if eq
    then do {
      RETURN (stat,
        insert (new-var st)  $\mathcal{V}$ , fmupd (new-id st) (r) A)}
    else RETURN (FAILED, ( $\mathcal{V}$ , A))
  }
)

```

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

definition *polys-rel-full* :: $\langle ((\text{nat set} \times (\text{nat}, \text{int mpoly})\text{fmap}) \times -) \text{set} \rangle$ **where**
 $\langle \text{polys-rel-full} = \{((\mathcal{V}, A), (\mathcal{V}', B)). (A, B) \in \text{polys-rel} \wedge \mathcal{V} = \mathcal{V}'\} \rangle$

lemma *polys-rel-update-remove*:

$\langle x13 \notin \# \text{dom-m } A \Longrightarrow x11 \in \# \text{dom-m } A \Longrightarrow x12 \in \# \text{dom-m } A \Longrightarrow x11 \neq x12 \Longrightarrow (A, B) \in \text{polys-rel} \Longrightarrow$
 \Longrightarrow
 $(\text{fmupd } x13 \text{ r } (\text{fmdrop } x11 (\text{fmdrop } x12 \text{ A})),$
 $\text{add-mset } r \text{ B} - \{\# \text{the } (\text{fmlookup } A \text{ } x11), \text{the } (\text{fmlookup } A \text{ } x12)\#\}$
 $\in \text{polys-rel} \rangle$
 $\langle x13 \notin \# \text{dom-m } A \Longrightarrow x11 \in \# \text{dom-m } A \Longrightarrow (A, B) \in \text{polys-rel} \Longrightarrow$
 $(\text{fmupd } x13 \text{ r } (\text{fmdrop } x11 \text{ A}), \text{add-mset } r \text{ B} - \{\# \text{the } (\text{fmlookup } A \text{ } x11)\#\})$
 $\in \text{polys-rel} \rangle$
 $\langle x13 \notin \# \text{dom-m } A \Longrightarrow (A, B) \in \text{polys-rel} \Longrightarrow$
 $(\text{fmupd } x13 \text{ r } A, \text{add-mset } r \text{ B}) \in \text{polys-rel} \rangle$
 $\langle x13 \in \# \text{dom-m } A \Longrightarrow (A, B) \in \text{polys-rel} \Longrightarrow$
 $(\text{fmdrop } x13 \text{ A}, \text{remove1-mset } (\text{the } (\text{fmlookup } A \text{ } x13)) \text{ B}) \in \text{polys-rel} \rangle$
 $\langle \text{proof} \rangle$

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

$\langle (A, B) \in \text{polys-rel} \Longrightarrow$
 $x12 \in \# \text{dom-m } A \Longrightarrow$
 $\text{the } (\text{fmlookup } A \text{ } x12) \in \# B \rangle$
 $\langle \text{proof} \rangle$

lemma *PAC-Format-add-and-remove*:

$\langle r - x14 \in \text{More-Modules.ideal polynomial-bool} \Longrightarrow$
 $(A, B) \in \text{polys-rel} \Longrightarrow$
 $x12 \in \# \text{dom-m } A \Longrightarrow$
 $x13 \notin \# \text{dom-m } A \Longrightarrow$
 $\text{vars } r \subseteq \mathcal{V} \Longrightarrow$
 $2 * \text{the } (\text{fmlookup } A \text{ } x12) - r \in \text{More-Modules.ideal polynomial-bool} \Longrightarrow$
 $\text{PAC-Format}^{**} (\mathcal{V}, B) (\mathcal{V}, \text{remove1-mset } (\text{the } (\text{fmlookup } A \text{ } x12)) (\text{add-mset } r \text{ B})) \rangle$

$\langle r - x14 \in \text{More-Modules.ideal polynomial-bool} \implies$
 $(A, B) \in \text{polys-rel} \implies$
 $\text{the (fmlookup A x11) + the (fmlookup A x12) - r} \in \text{More-Modules.ideal polynomial-bool} \implies$
 $x11 \in \# \text{ dom-m A} \implies$
 $x12 \in \# \text{ dom-m A} \implies$
 $\text{vars } r \subseteq \mathcal{V} \implies$
 $\text{PAC-Format}^{**} (\mathcal{V}, B) (\mathcal{V}, \text{add-mset } r B)\rangle$

$\langle r - x14 \in \text{More-Modules.ideal polynomial-bool} \implies$
 $(A, B) \in \text{polys-rel} \implies$
 $x11 \in \# \text{ dom-m A} \implies$
 $x12 \in \# \text{ dom-m A} \implies$
 $\text{the (fmlookup A x11) + the (fmlookup A x12) - r} \in \text{More-Modules.ideal polynomial-bool} \implies$
 $\text{vars } r \subseteq \mathcal{V} \implies$
 $x11 \neq x12 \implies$
 $\text{PAC-Format}^{**} (\mathcal{V}, B)$
 $(\mathcal{V}, \text{add-mset } r B - \{\# \text{the (fmlookup A x11), the (fmlookup A x12)}\})\rangle$

$\langle (A, B) \in \text{polys-rel} \implies$
 $r - x34 \in \text{More-Modules.ideal polynomial-bool} \implies$
 $x11 \in \# \text{ dom-m A} \implies$
 $\text{the (fmlookup A x11) * x32 - r} \in \text{More-Modules.ideal polynomial-bool} \implies$
 $\text{vars } x32 \subseteq \mathcal{V} \implies$
 $\text{vars } r \subseteq \mathcal{V} \implies$
 $\text{PAC-Format}^{**} (\mathcal{V}, B) (\mathcal{V}, \text{add-mset } r B)\rangle$

$\langle (A, B) \in \text{polys-rel} \implies$
 $r - x34 \in \text{More-Modules.ideal polynomial-bool} \implies$
 $x11 \in \# \text{ dom-m A} \implies$
 $\text{the (fmlookup A x11) * x32 - r} \in \text{More-Modules.ideal polynomial-bool} \implies$
 $\text{vars } x32 \subseteq \mathcal{V} \implies$
 $\text{vars } r \subseteq \mathcal{V} \implies$
 $\text{PAC-Format}^{**} (\mathcal{V}, B) (\mathcal{V}, \text{remove1-mset (the (fmlookup A x11)) (add-mset } r B))\rangle$

$\langle (A, B) \in \text{polys-rel} \implies$
 $x12 \in \# \text{ dom-m A} \implies$
 $\text{PAC-Format}^{**} (\mathcal{V}, B) (\mathcal{V}, \text{remove1-mset (the (fmlookup A x12)) } B)\rangle$

$\langle (A, B) \in \text{polys-rel} \implies$
 $(p' + \text{Var } x)^2 - (p' + \text{Var } x) \in \text{ideal polynomial-bool} \implies$
 $x \notin \mathcal{V} \implies$
 $x \notin \text{vars}(p' + \text{Var } x) \implies$
 $\text{vars}(p' + \text{Var } x) \subseteq \mathcal{V} \implies$
 $\text{PAC-Format}^{**} (\mathcal{V}, B)$
 $(\text{insert } x \mathcal{V}, \text{add-mset } p' B)\rangle$

$\langle \text{proof} \rangle$

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

$\langle \text{status-rel} \equiv \text{Id} \rangle$

lemma $\text{is-merge-status}[\text{simp}]$:

$\langle \text{is-failed (merge-status a st')} \longleftrightarrow \text{is-failed } a \vee \text{is-failed } st' \rangle$

$\langle \text{is-found (merge-status a st')} \longleftrightarrow \neg \text{is-failed } a \wedge \neg \text{is-failed } st' \wedge (\text{is-found } a \vee \text{is-found } st') \rangle$

$\langle \text{is-success (merge-status a st')} \longleftrightarrow (\text{is-success } a \wedge \text{is-success } st') \rangle$

$\langle \text{proof} \rangle$

lemma $\text{status-rel-merge-status}$:

$\langle (\text{merge-status } a \text{ b, SUCCESS}) \notin \text{status-rel} \longleftrightarrow$

$(a = \text{FAILED}) \vee (b = \text{FAILED}) \vee$

$a = FOUND \vee (b = FOUND)$
 ⟨proof⟩

lemma *Ex-status-iff*:

⟨ $(\exists a. P a) \longleftrightarrow P SUCCESS \vee P FOUND \vee (P (FAILED))$ ⟩
 ⟨proof⟩

lemma *is-failed-alt-def*:

⟨*is-failed* $st' \longleftrightarrow \neg is-success\ st' \wedge \neg is-found\ st'$ ⟩
 ⟨proof⟩

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

⟨*merge-status* $a SUCCESS = SUCCESS \longleftrightarrow a = SUCCESS$ ⟩
 ⟨*merge-status* $a SUCCESS = FOUND \longleftrightarrow a = FOUND$ ⟩
 ⟨*merge-status* $SUCCESS a = SUCCESS \longleftrightarrow a = SUCCESS$ ⟩
 ⟨*merge-status* $SUCCESS a = FOUND \longleftrightarrow a = FOUND$ ⟩
 ⟨*merge-status* $SUCCESS a = FAILED \longleftrightarrow a = FAILED$ ⟩
 ⟨*merge-status* $a SUCCESS = FAILED \longleftrightarrow a = FAILED$ ⟩
 ⟨*merge-status* $FOUND a = FAILED \longleftrightarrow a = FAILED$ ⟩
 ⟨*merge-status* $a FOUND = FAILED \longleftrightarrow a = FAILED$ ⟩
 ⟨*merge-status* $a FOUND = SUCCESS \longleftrightarrow False$ ⟩
 ⟨*merge-status* $a b = FOUND \longleftrightarrow (a = FOUND \vee b = FOUND) \wedge (a \neq FAILED \wedge b \neq FAILED)$ ⟩
 ⟨proof⟩

lemma *fmdrop-irrelevant*: ⟨ $x11 \notin \# dom-m\ A \implies fmdrop\ x11\ A = A$ ⟩

⟨proof⟩

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

fixes $a :: \langle status \rangle$

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

⟨ $\neg is-failed\ a$ ⟩ **and**

[*simp, intro*]: ⟨ $a = FOUND \implies spec \in pac-ideal\ (set-mset\ A_0)$ ⟩ **and**

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

$spec_0: \langle vars\ spec \subseteq \mathcal{V}_0 \rangle$ **and**

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

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

⟨proof⟩

definition *PAC-checker*

:: ⟨*int-poly* $\implies fpac-step \implies status \implies (int-poly, nat, nat)\ pac-step\ list \implies$

$(status \times fpac-step)\ nres$ ⟩

where

⟨*PAC-checker spec* $A\ b\ st = do\ \{$

$(S, -) \leftarrow WHILE_T$

$(\lambda((b :: status, A :: fpac-step), st). \neg is-failed\ b \wedge st \neq [])$

$(\lambda((bA), st). do\ \{$

$ASSERT(st \neq [])$;

$S \leftarrow PAC-checker-step\ spec\ (bA)\ (hd\ st)$;

$RETURN\ (S, tl\ st)$

$\})$

$((b, A), st)$;

$RETURN\ S$

$\}\rangle$

lemma *PAC-checker-specification-spec-trans*:

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

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-polynomial-bool* :: $\langle \text{int mpolys} \Rightarrow \text{nat set} \Rightarrow (\text{nat}, \text{int-poly}) \text{ fmap} \Rightarrow (\text{status} \times \text{fpac-step}) \text{ nres} \rangle$ **where**

$\langle \text{remap-polys-polynomial-bool spec} = (\lambda \mathcal{V} A.$
 $\text{SPEC}(\lambda(\text{st}, \mathcal{V}', A'). (\neg \text{is-failed } \text{st} \longrightarrow$
 $\text{dom-m } A = \text{dom-m } A' \wedge$
 $(\forall i \in \# \text{dom-m } A. \text{the } (\text{fmlookup } A \ i) - \text{the } (\text{fmlookup } A' \ i) \in \text{ideal polynomial-bool}) \wedge$
 $\bigcup (\text{vars } ' \text{set-mset} (\text{ran-m } A)) \subseteq \mathcal{V}' \wedge$
 $\bigcup (\text{vars } ' \text{set-mset} (\text{ran-m } A')) \subseteq \mathcal{V}') \wedge$
 $(\text{st} = \text{FOUND} \longrightarrow \text{spec} \in \# \text{ran-m } A')) \rangle$

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

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

lemma *fmap-eq-dom-iff*:

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

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*:

$\langle \text{dom-m } b = \text{dom-m } ba \implies$
 $\forall i \in \# \text{dom-m } ba.$

$the (fmlookup\ b\ i) - the (fmlookup\ ba\ i)$
 $\in More-Modules.ideal\ polynomial\ bool \implies$
 $pac\ ideal\ ((\lambda x. the (fmlookup\ b\ x))\ 'set\ mset\ (dom\ m\ ba)) = pac\ ideal\ ((\lambda x. the (fmlookup\ ba\ x))\ 'set\ mset\ (dom\ m\ ba))$
 <proof>

lemma *remap-polys-polynomial-bool-remap-polys-change-all:*
 <remap-polys-polynomial-bool spec \mathcal{V} $A \leq$ remap-polys-change-all spec \mathcal{V} A >
 <proof>

definition *remap-polys* :: <int mpoly \Rightarrow nat set \Rightarrow (nat, int-poly) fmap \Rightarrow (status \times fpac-step) nres>
where

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

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

lemma *remap-polys-spec:*
 <remap-polys spec \mathcal{V} $A \leq$ remap-polys-polynomial-bool spec \mathcal{V} A >
 <proof>

6.3 Full Checker

definition *full-checker*

:: <int-poly \Rightarrow (nat, int-poly) fmap \Rightarrow (int-poly, nat, nat) pac-step list \Rightarrow (status \times -) nres>

where

<full-checker spec0 A pac = do {
 spec \leftarrow normalize-poly-spec spec0;
 (st , \mathcal{V} , A) \leftarrow remap-polys-change-all spec {} A ;
 if is-failed st then
 RETURN (st , \mathcal{V} , A)
 else do {
 $\mathcal{V} \leftarrow$ SPEC($\lambda \mathcal{V}'$. $\mathcal{V} \cup vars\ spec0 \subseteq \mathcal{V}'$);
 PAC-checker spec (\mathcal{V} , A) st pac
 }
 }>

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 *pac-ideal-idemp*: $\langle \text{pac-ideal} (\text{pac-ideal } A) = \text{pac-ideal } A \rangle$
 $\langle \text{proof} \rangle$

lemma *full-checker-spec*:
assumes $\langle (A, A') \in \text{polys-rel} \rangle$
shows
 $\langle \text{full-checker spec } A \text{ pac} \leq \Downarrow \{((st, G), (st', G')). (st, st') \in \text{status-rel} \wedge$
 $(st \neq \text{FAILED} \longrightarrow (G, G') \in \text{polys-rel-full})\}$
 $(\text{PAC-checker-specification spec } (A')) \rangle$
 $\langle \text{proof} \rangle$

lemma *full-checker-spec'*:
shows
 $\langle (\text{uncurry2 full-checker}, \text{uncurry2 } (\lambda \text{spec } A. \text{PAC-checker-specification spec } A)) \in$
 $(\text{Id} \times_r \text{polys-rel}) \times_r \text{Id} \rightarrow_f \{((st, G), (st', G')). (st, st') \in \text{status-rel} \wedge$
 $(st \neq \text{FAILED} \longrightarrow (G, G') \in \text{polys-rel-full})\} \text{nres-rel} \rangle$
 $\langle \text{proof} \rangle$

end
theory *PAC-Polynomials*
imports *PAC-Specification Finite-Map-Multiset*
begin

7 Polynomials of strings

Isabelle's definition of polynomials only work with variables of type *nat*. Therefore, we introduce a version that uses strings by using an injective function that converts a string to a natural number. It exists because strings are countable. Remark that the whole development is independent of the function.

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

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

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

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

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

definition *mult-poly-raw* :: $\langle \text{mset-polynomial} \Rightarrow \text{mset-polynomial} \Rightarrow \text{mset-polynomial} \rangle$ **where**

$\langle \text{mult-poly-raw } p q =$
 $(\text{sum-mset } ((\lambda y. \text{mult-poly-by-monom } y q) \# p)) \rangle$

definition *remove-powers* :: $\langle \text{mset-polynomial} \Rightarrow \text{mset-polynomial} \rangle$ **where**

$\langle \text{remove-powers } x s = \text{image-mset } (\text{apfst } \text{remdups-mset}) x s \rangle$

definition *all-vars-mset* :: $\langle \text{mset-polynomial} \Rightarrow \text{string multiset} \rangle$ **where**

$\langle \text{all-vars-mset } p = \sum \# (\text{fst } \# p) \rangle$

abbreviation *all-vars* :: $\langle \text{mset-polynomial} \Rightarrow \text{string set} \rangle$ **where**

$\langle \text{all-vars } p \equiv \text{set-mset } (\text{all-vars-mset } p) \rangle$

definition *add-to-coefficient* :: $\langle - \Rightarrow \text{mset-polynomial} \Rightarrow \text{mset-polynomial} \rangle$ **where**

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

definition *normalize-poly* :: $\langle \text{mset-polynomial} \Rightarrow \text{mset-polynomial} \rangle$ **where**

$\langle \text{normalize-poly } p = \text{fold-mset } \text{add-to-coefficient } \{ \# \} p \rangle$

lemma *add-to-coefficient-simps*:

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

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

$\langle \text{proof} \rangle$

lemma *normalized-poly-normalize-poly*[simp]:
 ⟨normalized-poly (normalize-poly p)⟩
 ⟨proof⟩

7.2 Addition

inductive *add-poly-p* :: ⟨mset-polynomial × mset-polynomial × mset-polynomial ⇒ mset-polynomial × mset-polynomial × mset-polynomial ⇒ bool⟩ **where**

add-new-coeff-r:

⟨add-poly-p (p, add-mset x q, r) (p, q, add-mset x r)⟩ |

add-new-coeff-l:

⟨add-poly-p (add-mset x p, q, r) (p, q, add-mset x r)⟩ |

add-same-coeff-l:

⟨add-poly-p (add-mset (x, n) p, q, add-mset (x, m) r) (p, q, add-mset (x, n + m) r)⟩ |

add-same-coeff-r:

⟨add-poly-p (p, add-mset (x, n) q, add-mset (x, m) r) (p, q, add-mset (x, n + m) r)⟩ |

rem-0-coeff:

⟨add-poly-p (p, q, add-mset (x, 0) r) (p, q, r)⟩

inductive-cases *add-poly-pE*: ⟨add-poly-p S T⟩

lemmas *add-poly-p-induct* =

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

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

⟨add-poly-p** (p, q, r) ({#}, q, p + r)⟩

⟨proof⟩

lemma *add-poly-p-empty-r*:

⟨add-poly-p** (p, q, r) (p, {#}, q + r)⟩

⟨proof⟩

lemma *add-poly-p-sym*:

⟨add-poly-p (p, q, r) (p', q', r') ⟷ add-poly-p (q, p, r) (q', p', r')⟩

⟨proof⟩

lemma *wf-if-measure-in-wf*:

⟨wf R ⟹ (∧ a b. (a, b) ∈ S ⟹ (ν a, ν b) ∈ R) ⟹ wf S⟩

⟨proof⟩

lemma *lexn-n*:

⟨n > 0 ⟹ (x # xs, y # ys) ∈ lexn r n ⟷

(length xs = n - 1 ∧ length ys = n - 1) ∧ ((x, y) ∈ r ∨ (x = y ∧ (xs, ys) ∈ lexn r (n - 1)))⟩

⟨proof⟩

lemma *wf-add-poly-p*:

⟨wf {(x, y). add-poly-p y x}⟩

⟨proof⟩

lemma *mult-poly-by-monom-simps*[simp]:

⟨mult-poly-by-monom t {#} = {#}⟩

⟨mult-poly-by-monom t (ps + qs) = mult-poly-by-monom t ps + mult-poly-by-monom t qs⟩

⟨mult-poly-by-monom a (add-mset p ps) = add-mset (fst a + fst p, snd a * snd p) (mult-poly-by-monom a ps)⟩

⟨proof⟩

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

for $q :: \text{mset-polynomial}$ **where**

mult-step:

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

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

7.3 Normalisation

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

rem-0-coeff[*simp*, *intro*]:

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

merge-dup-coeff[*simp*, *intro*]:

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

same[*simp*, *intro*]:

$\langle \text{normalize-poly-p } p \ p \rangle \mid$

keep-coeff[*simp*, *intro*]:

$\langle \text{normalize-poly-p } p \ q \Longrightarrow \text{normalize-poly-p (add-mset } x \text{ p) (add-mset } x \text{ q) \rangle$

7.4 Correctness

This locales maps string polynomials to real polynomials.

locale *poly-embed* =

fixes $\varphi :: \langle \text{string} \Rightarrow \text{nat} \rangle$

assumes $\varphi\text{-inj}: \langle \text{inj } \varphi \rangle$

begin

definition *poly-of-vars* :: $\text{term-poly} \Rightarrow ('a :: \{\text{comm-semiring-1}\}) \text{ mpoly}$ **where**

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

lemma *poly-of-vars-simps*[*simp*]:

shows

$\langle \text{poly-of-vars (add-mset } x \text{ xs)} = \text{Var } (\varphi \ x) * (\text{poly-of-vars } xs :: ('a :: \{\text{comm-semiring-1}\}) \ \text{mpoly}) \rangle$ **(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 *comp-fun-commute* $\langle \text{mononom-of-vars} \rangle$

$\langle \text{proof} \rangle$

lemma [*simp*]:

$\langle \text{poly-of-vars } \{\#\} = 1 \rangle$

$\langle \text{proof} \rangle$

lemma *mononom-of-vars-add*[*simp*]:

$\langle \text{NO-MATCH } 0 \ b \Longrightarrow \text{mononom-of-vars } xs \ b = \text{Const (snd } xs) * \text{poly-of-vars (fst } xs) + b \rangle$

⟨proof⟩

definition *polynomial-of-mset* :: ⟨mset-polynomial ⇒ -⟩ **where**
⟨polynomial-of-mset p = sum-mset (mononom-of-vars '# p) 0⟩

lemma *polynomial-of-mset-append[simp]*:
⟨polynomial-of-mset (xs + ys) = polynomial-of-mset xs + polynomial-of-mset ys⟩
⟨proof⟩

lemma *polynomial-of-mset-Cons[simp]*:
⟨polynomial-of-mset (add-mset x ys) = Const (snd x) * poly-of-vars (fst x) + polynomial-of-mset ys⟩
⟨proof⟩

lemma *polynomial-of-mset-empty[simp]*:
⟨polynomial-of-mset {#} = 0⟩
⟨proof⟩

lemma *polynomial-of-mset-mult-poly-by-monom[simp]*:
⟨polynomial-of-mset (mult-poly-by-monom x ys) =
(Const (snd x) * poly-of-vars (fst x) * polynomial-of-mset ys)⟩
⟨proof⟩

lemma *polynomial-of-mset-mult-poly-raw[simp]*:
⟨polynomial-of-mset (mult-poly-raw xs ys) = polynomial-of-mset xs * polynomial-of-mset ys⟩
⟨proof⟩

lemma *polynomial-of-mset-uminus*:
⟨polynomial-of-mset {#case x of (a, b) ⇒ (a, - b). x ∈# za#} =
- polynomial-of-mset za⟩
⟨proof⟩

lemma *X2-X-polynomial-bool-mult-in*:
⟨Var (x1) * (Var (x1) * p) - Var (x1) * p ∈ More-Modules.ideal polynomial-bool⟩
⟨proof⟩

lemma *polynomial-of-list-remove-powers-polynomial-bool*:
⟨(polynomial-of-mset xs) - polynomial-of-mset (remove-powers xs) ∈ ideal polynomial-bool⟩
⟨proof⟩

lemma *add-poly-p-polynomial-of-mset*:
⟨add-poly-p (p, q, r) (p', q', r') ⇒
polynomial-of-mset r + (polynomial-of-mset p + polynomial-of-mset q) =
polynomial-of-mset r' + (polynomial-of-mset p' + polynomial-of-mset q')⟩
⟨proof⟩

lemma *rtranclp-add-poly-p-polynomial-of-mset*:
⟨add-poly-p** (p, q, r) (p', q', r') ⇒
polynomial-of-mset r + (polynomial-of-mset p + polynomial-of-mset q) =
polynomial-of-mset r' + (polynomial-of-mset p' + polynomial-of-mset q')⟩
⟨proof⟩

lemma *rtranclp-add-poly-p-polynomial-of-mset-full*:

$\langle \text{add-poly-p}^{**} (p, q, \{\#\}) (\{\#\}, \{\#\}, r') \implies$
 $\text{polynomial-of-mset } r' = (\text{polynomial-of-mset } p + \text{polynomial-of-mset } q) \rangle$
 $\langle \text{proof} \rangle$

lemma *poly-of-vars-remdups-mset:*

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

lemma *polynomial-of-mset-mult-map:*

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

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

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

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

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

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

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

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

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

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

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

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

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

$\langle \text{proof} \rangle$

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

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

$\langle \text{proof} \rangle$

end

end

theory *PAC-Polynomials-Term*

imports *PAC-Polynomials*

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**

$A :: \langle ('a \times 'b) \times 'd \ \text{set} \rangle$

shows

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

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

$\langle \text{proof} \rangle$

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**

$A :: \langle 'd \times ('a \times 'b) \ \text{set} \rangle$

shows

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

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

$\langle \text{proof} \rangle$

type-synonym *term-poly-list* = $\langle \text{string list} \rangle$

type-synonym *llist-polynomial* = $\langle (\text{term-poly-list} \times \text{int}) \ \text{list} \rangle$

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*

⟨proof⟩

abbreviation *less-than-char* :: ⟨(char × char) set⟩ **where**
⟨*less-than-char* ≡ *p2rel less-char*⟩

lemma *less-than-char-def*:
⟨(x,y) ∈ *less-than-char* ⟷ *less-char* x y⟩
⟨proof⟩

lemma *trans-less-than-char[simp]*:
⟨*trans less-than-char*⟩ **and**
irrefl-less-than-char:
⟨*irrefl less-than-char*⟩ **and**
antisym-less-than-char:
⟨*antisym less-than-char*⟩
⟨proof⟩

8.2 Polynomials

definition *var-order-rel* :: ⟨(string × string) set⟩ **where**
⟨*var-order-rel* ≡ *lexord less-than-char*⟩

abbreviation *var-order* :: ⟨string ⇒ string ⇒ bool⟩ **where**
⟨*var-order* ≡ *rel2p var-order-rel*⟩

abbreviation *term-order-rel* :: ⟨(term-poly-list × term-poly-list) set⟩ **where**
⟨*term-order-rel* ≡ *lexord var-order-rel*⟩

abbreviation *term-order* :: ⟨term-poly-list ⇒ term-poly-list ⇒ bool⟩ **where**
⟨*term-order* ≡ *rel2p term-order-rel*⟩

definition *term-poly-list-rel* :: ⟨(term-poly-list × term-poly) set⟩ **where**
⟨*term-poly-list-rel* = {(xs, ys).
ys = *mset xs* ∧
distinct xs ∧
sorted-wrt (*rel2p var-order-rel*) xs}⟩

definition *unsorted-term-poly-list-rel* :: ⟨(term-poly-list × term-poly) set⟩ **where**
⟨*unsorted-term-poly-list-rel* = {(xs, ys).
ys = *mset xs* ∧ distinct xs}⟩

definition *poly-list-rel* :: ⟨- ⇒ (('a × int) list × *mset-polynomial*) set⟩ **where**
⟨*poly-list-rel* R = {(xs, ys).
(xs, ys) ∈ (R ×_r *int-rel*)list-rel O *list-mset-rel* ∧
0 ∉ # snd '# ys}⟩

definition *sorted-poly-list-rel-wrt* :: ⟨('a ⇒ 'a ⇒ bool)
⇒ ('a × string multiset) set ⇒ (('a × int) list × *mset-polynomial*) set⟩ **where**
⟨*sorted-poly-list-rel-wrt* S R = {(xs, ys).
(xs, ys) ∈ (R ×_r *int-rel*)list-rel O *list-mset-rel* ∧
sorted-wrt S (map fst xs) ∧
distinct (map fst xs) ∧
0 ∉ # snd '# ys}⟩

abbreviation *sorted-poly-list-rel* **where**
⟨*sorted-poly-list-rel* R ≡ *sorted-poly-list-rel-wrt* R *term-poly-list-rel*⟩

abbreviation *sorted-poly-rel* **where**

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

definition *sorted-repeat-poly-list-rel-wrt* $:: \langle ('a \Rightarrow 'a \Rightarrow \text{bool})$

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

abbreviation *sorted-repeat-poly-list-rel* **where**

$\langle \text{sorted-repeat-poly-list-rel } R \equiv \text{sorted-repeat-poly-list-rel-wrt } R \ \text{term-poly-list-rel} \rangle$

abbreviation *sorted-repeat-poly-rel* **where**

$\langle \text{sorted-repeat-poly-rel} \equiv \text{sorted-repeat-poly-list-rel } (\text{rel2p } (\text{Id} \cup \text{lexord var-order-rel})) \rangle$

abbreviation *unsorted-poly-rel* **where**

$\langle \text{unsorted-poly-rel} \equiv \text{poly-list-rel term-poly-list-rel} \rangle$

lemma *sorted-poly-list-rel-empty-l[simp]*:

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

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

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

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* $:: \langle - \Rightarrow (('a \times \text{int}) \text{ list} \times \text{mset-polynomial}) \text{ set} \rangle$ **where**

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

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* $:: \langle ('a \Rightarrow 'a \Rightarrow \text{bool})$

$\Rightarrow ('a \times \text{string multiset}) \text{ set} \Rightarrow (('a \times \text{int}) \text{ list} \times \text{mset-polynomial}) \text{ set} \rangle$ **where**

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

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**

$\langle \text{sorted-repeat-poly-rel-with0} \equiv \text{sorted-repeat-poly-list-rel-with0 } (\text{rel2p } (\text{Id} \cup \text{lexord } \text{var-order-rel})) \rangle$

lemma *term-poly-list-relD*:

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

end

theory *PAC-Polynomials-Operations*

imports *PAC-Polynomials-Term PAC-Checker-Specification*

begin

8.3 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'* :: $\langle \text{l-list-polynomial} \times \text{l-list-polynomial} \Rightarrow \text{l-list-polynomial} \rangle$ **where**

$\langle \text{add-poly-l}' (p, []) = p \rangle \mid$
 $\langle \text{add-poly-l}' ([], q) = q \rangle \mid$
 $\langle \text{add-poly-l}' ((xs, n) \# p, (ys, m) \# q) =$
 $\quad (\text{if } xs = ys \text{ then if } n + m = 0 \text{ then } \text{add-poly-l}' (p, q) \text{ else}$
 $\quad \quad \text{let } pq = \text{add-poly-l}' (p, q) \text{ in}$
 $\quad \quad ((xs, n + m) \# pq)$
 $\quad \text{else if } (xs, ys) \in \text{term-order-rel}$
 $\quad \text{then}$
 $\quad \quad \text{let } pq = \text{add-poly-l}' (p, (ys, m) \# q) \text{ in}$
 $\quad \quad ((xs, n) \# pq)$
 $\quad \text{else}$
 $\quad \quad \text{let } pq = \text{add-poly-l}' ((xs, n) \# p, q) \text{ in}$
 $\quad \quad ((ys, m) \# pq)$
 $\quad \rangle$

definition *add-poly-l* :: $\langle \text{l-list-polynomial} \times \text{l-list-polynomial} \Rightarrow \text{l-list-polynomial nres} \rangle$ **where**

$\langle \text{add-poly-l} = \text{REC}_T$
 $\quad (\lambda \text{add-poly-l } (p, q).$
 $\quad \text{case } (p, q) \text{ of}$
 $\quad \quad (p, []) \Rightarrow \text{RETURN } p$
 $\quad \quad | ([], q) \Rightarrow \text{RETURN } q$
 $\quad \quad | ((xs, n) \# p, (ys, m) \# q) \Rightarrow$
 $\quad \quad \quad (\text{if } xs = ys \text{ then if } n + m = 0 \text{ then } \text{add-poly-l } (p, q) \text{ else}$
 $\quad \quad \quad \text{do } \{$

```

    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*:

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

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*:

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

$(ys, xs) \in \text{lexord } (\text{lexord less-than-char})$
 ⟨proof⟩

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

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

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

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

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

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

lemma *add-poly-l-spec*:

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

definition *sort-poly-spec* :: $\langle \text{l-list-polynomial} \Rightarrow \text{l-list-polynomial nres} \rangle$ **where**

⟨*sort-poly-spec* $p =$
 $\text{SPEC}(\lambda p'. \text{mset } p = \text{mset } p' \wedge \text{sorted-wrt } (\text{rel2p } (\text{Id} \cup \text{term-order-rel})) (\text{map } \text{fst } p')) \rangle$

lemma *sort-poly-spec-id*:

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

8.4 Multiplication

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

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

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

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

lemma *mset-eqD-set-mset*: $\langle \text{mset } xs = A \implies \text{set } xs = \text{set-mset } A \rangle$

⟨proof⟩

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

⟨ $(y \# p, p') \in \text{term-poly-list-rel} \longleftrightarrow$
 $(p, \text{remove1-mset } y p') \in \text{term-poly-list-rel} \wedge$

$y \in \# p' \wedge y \notin \text{set } p \wedge y \notin \# \text{remove1-mset } y p' \wedge$
 $(\forall x \in \# \text{mset } p. (y, x) \in \text{var-order-rel})$
 ⟨proof⟩

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

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

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

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

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

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

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

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

lemma *mult-monoms-spec*:

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

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

where

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

definition *mult-poly-raw* :: $\langle \text{l-list-polynomial} \Rightarrow \text{l-list-polynomial} \Rightarrow \text{l-list-polynomial} \rangle$ **where**

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

fun *map-append* **where**

$\langle \text{map-append } f \ b \ [] = b \rangle \mid$
 $\langle \text{map-append } f \ b \ (x \ # \ xs) = f \ x \ # \ \text{map-append } f \ b \ xs \rangle$

lemma *map-append-alt-def*:

$\langle \text{map-append } f \ b \ xs = \text{map } f \ xs \ @ \ b \rangle$
 ⟨proof⟩

lemma *foldl-append-empty*:

$\langle \text{NO-MATCH } [] \ xs \implies \text{foldl } (\lambda b \ x. f \ x \ @ \ b) \ xs \ p = \text{foldl } (\lambda b \ x. f \ x \ @ \ b) \ [] \ p \ @ \ xs \rangle$
 ⟨proof⟩

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

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

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

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

lemma *sorted-poly-list-relD*:

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

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

$\langle \text{list-all2 } R \ p \ q \implies x \in \text{set } p \implies \exists y \in \text{set } q. R \ x \ y \rangle$
 $\langle \text{proof} \rangle$

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

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

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

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

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

fun *merge-coeffs* :: $\langle \text{l-list-polynomial} \Rightarrow \text{l-list-polynomial} \rangle$ **where**

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

abbreviation **(in** $-$)*mononomys* :: $\langle \text{l-list-polynomial} \Rightarrow \text{term-poly-list set} \rangle$ **where**

$\langle \text{mononomys } p \equiv \text{fst } \text{'set } p \rangle$

lemma *fst-normalize-polynomial-subset*:

$\langle \text{mononomys } (\text{merge-coeffs } p) \subseteq \text{mononomys } p \rangle$
 $\langle \text{proof} \rangle$

lemma *fst-normalize-polynomial-subsetD*:

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

lemma *distinct-merge-coeffs*:

assumes $\langle \text{sorted-wrt } R \ (\text{map } \text{fst } xs) \rangle$ **and** $\langle \text{transp } R \rangle \langle \text{antisymp } R \rangle$
shows $\langle \text{distinct } (\text{map } \text{fst } (\text{merge-coeffs } xs)) \rangle$
 $\langle \text{proof} \rangle$

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*:

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

$\langle \text{proof} \rangle$

8.5 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**

$\langle \text{sort-coeff } ys = \text{SPEC}(\lambda xs. \text{mset } xs = \text{mset } ys \wedge \text{sorted-wrt } (\text{rel2p } (\text{Id} \cup \text{var-order-rel})) \text{ } xs) \rangle$

lemma *distinct-var-order-Id-var-order*:

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

$\langle \text{proof} \rangle$

definition *sort-all-coeffs* :: $\langle \text{l-list-polynomial} \Rightarrow \text{l-list-polynomial nres} \rangle$ **where**

$\langle \text{sort-all-coeffs } xs = \text{monadic-nfoldli } xs \ (\lambda -. \text{RETURN } \text{True}) \ (\lambda(a, n) \ b. \text{do } \{ a \leftarrow \text{sort-coeff } a; \text{RETURN } ((a, n) \# b) \}) \ [] \rangle$

lemma *sort-all-coeffs-gen*:

assumes $\langle (\forall xs \in \text{mononoms } xs'. \text{sorted-wrt } (\text{rel2p } (\text{var-order-rel})) \text{ } xs) \rangle$ **and**

$\langle \forall x \in \text{mononoms } (xs \ @ \ xs'). \text{distinct } x \rangle$

shows $\langle \text{monadic-nfoldli } xs \ (\lambda -. \text{RETURN } \text{True}) \ (\lambda(a, n) \ b. \text{do } \{ a \leftarrow \text{sort-coeff } a; \text{RETURN } ((a, n) \# b) \}) \ xs' \leq$

$\Downarrow \text{Id } (\text{SPEC}(\lambda ys. \text{map } (\lambda(a, b). (\text{mset } a, b)) (\text{rev } xs \ @ \ xs') = \text{map } (\lambda(a, b). (\text{mset } a, b)) \text{ } (ys) \wedge$
 $\quad (\forall xs \in \text{mononoms } ys. \text{sorted-wrt } (\text{rel2p } (\text{var-order-rel})) \text{ } xs))) \rangle$

$\langle \text{proof} \rangle$

definition *shuffle-coefficients where*

$\langle \text{shuffle-coefficients } xs = (\text{SPEC}(\lambda ys. \text{map } (\lambda(a, b). (\text{mset } a, b)) (\text{rev } xs) = \text{map } (\lambda(a, b). (\text{mset } a, b)) \text{ } ys \wedge$

$\quad (\forall xs \in \text{mononoms } ys. \text{sorted-wrt } (\text{rel2p } (\text{var-order-rel})) \text{ } xs))) \rangle$

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*:

$\langle \text{RETURN } o \ (\text{map } f) =$

$\text{REC}_T \ (\lambda g \ xs.$

$\text{case } xs \text{ of}$

$\quad [] \Rightarrow \text{RETURN } []$

$\langle x \# xs \Rightarrow do \{xs \leftarrow g \ xs; RETURN \ (f \ x \ \# \ xs)\} \rangle$
 $\langle proof \rangle$

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

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

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

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

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

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

lemma *shuffle-terms-distinct-iff*:

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

lemma

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

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

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

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

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

fun *merge-coeffs0* :: $\langle \text{l}list\text{-polynomial} \Rightarrow \text{l}list\text{-polynomial} \rangle$ **where**

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

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

$\langle ((ys, n) \# p, a) \in \text{sorted-repeat-poly-list-rel-with0-wrt } S \text{ term-poly-list-rel} \implies$

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

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

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

lemma *fst-normalize0-polynomial-subsetD*:

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

lemma *in-set-merge-coeffs0D*:

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

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

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

definition *full-normalize-poly where*

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

fun *sorted-remdups where*

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

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

$\langle \text{set } (\text{sorted-remdups } xs) = \text{set } xs \rangle$
 ⟨proof⟩

lemma *distinct-sorted-remdups*:

$\langle \text{sorted-wrt } R \text{ } xs \implies \text{antisymp } R \implies \text{distinct } (\text{sorted-remdups } xs) \rangle$
 ⟨proof⟩

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

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

definition *mult-poly-full :: ⟨-⟩ where*

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

normalize-poly pq
 }>

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

assumes $\langle (p, p') \in \text{unsorted-poly-rel} \rangle$

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

<proof>

8.6 Multiplication and normalisation

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

$\langle \text{mult-poly-p}' p' q' = \text{do } \{$

$pq \leftarrow \text{SPEC}(\lambda r. (\text{mult-poly-p } q')^{**} (p', \{\#\})) (\{\#\}, r);$

$\text{SPEC } (\lambda r. \text{normalize-poly-p}^{**} pq r)$

$\} \rangle$

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

$\langle \text{unsorted-poly-rel} \subseteq \text{fully-unsorted-poly-rel} \rangle$

<proof>

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

assumes $\langle (p, p') \in \text{sorted-poly-rel} \rangle \langle (q, q') \in \text{sorted-poly-rel} \rangle$

shows $\langle \text{mult-poly-full } p q \leq \Downarrow (\text{sorted-poly-rel}) (\text{mult-poly-p}' p' q') \rangle$

<proof>

definition *add-poly-spec* :: $\langle - \rangle$ **where**

$\langle \text{add-poly-spec } p q = \text{SPEC } (\lambda r. p + q - r \in \text{ideal polynomial-bool}) \rangle$

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

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

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

assumes $\langle (p, p') \in \text{sorted-poly-rel} \rangle \langle (q, q') \in \text{sorted-poly-rel} \rangle$

shows $\langle \text{add-poly-l } (p, q) \leq \Downarrow (\text{sorted-poly-rel}) (\text{add-poly-p}' p' q') \rangle$

<proof>

8.7 Correctness

context *poly-embed*

begin

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

$\langle \text{mset-poly-rel} = \{(a, b). b = \text{polynomial-of-mset } a\} \rangle$

definition *var-rel* **where**

$\langle \text{var-rel} = \text{br } \varphi (\lambda -. \text{True}) \rangle$

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

$\langle (p, p') \in \text{mset-poly-rel} \implies$

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

<proof>

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

$\langle (p, p') \in \text{mset-poly-rel} \implies (q, q') \in \text{mset-poly-rel} \implies$

$\text{mult-poly-p}' p q \leq \Downarrow \text{mset-poly-rel } (\text{mult-poly-spec } p' q') \rangle$

⟨proof⟩

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

⟨ $(p, p') \in \text{mset-poly-rel} \implies (q, q') \in \text{mset-poly-rel} \implies$
 $\text{add-poly-p}' p q \leq \Downarrow \text{mset-poly-rel} (\text{add-poly-spec } p' q')$ ⟩
⟨proof⟩

end

definition *weak-equality-l* :: ⟨*l*list-polynomial \Rightarrow *l*list-polynomial \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-polynomial \Rightarrow mset-polynomial \Rightarrow bool nres⟩ **where**

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

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

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

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

⟨ $(xa, y) \in \langle \text{term-poly-list-rel} \times_r \text{int-rel} \rangle \text{list-rel} \implies$
 $(xa, ya) \in \langle \text{term-poly-list-rel} \times_r \text{int-rel} \rangle \text{list-rel} \implies$
 $y = ya$ ⟩
⟨proof⟩

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

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

end

theory *PAC-Misc*

imports *Main*

begin

I believe this should be added to the simplifier by default...

lemma *Collect-eq-comp'*:

⟨ $\{(x, y). P x y\} O \{(c, a). c = f a\} = \{(x, a). P x (f a)\}$ ⟩
⟨proof⟩

lemma *in-set-conv-iff*:

$x \in \text{set } (\text{take } n \text{ } xs) \iff (\exists i < n. i < \text{length } xs \wedge xs ! i = x)$
⟨proof⟩

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

$x \in \text{set } (\text{take } n \text{ } xs) \iff (\exists i < \min n (\text{length } xs). xs ! i = x)$
⟨proof⟩

lemma *in-set-remove1D*:

$a \in \text{set } (\text{remove1 } x \text{ } xs) \implies a \in \text{set } xs$

⟨proof⟩

end

```
theory PAC-Checker
  imports PAC-Polynomials-Operations
         PAC-Checker-Specification
         PAC-Map-Rel
         Show.Show
         Show.Show-Instances
         PAC-Misc
begin
```

9 Executable Checker

In this layer we finally refine the checker to executable code.

9.1 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 another error message; hence we do not attempt to concatenate error messages.

```
fun merge-cstatus where
  ⟨merge-cstatus (CFAILED a) - = CFAILED a⟩ |
  ⟨merge-cstatus - (CFAILED a) = CFAILED a⟩ |
  ⟨merge-cstatus CFOUND - = CFOUND⟩ |
  ⟨merge-cstatus - CFOUND = CFOUND⟩ |
  ⟨merge-cstatus - - = CSUCCESS⟩
```

```
definition code-status-status-rel :: ⟨('a code-status × status) set⟩ where
  ⟨code-status-status-rel =
    {(CFOUND, FOUND), (CSUCCESS, SUCCESS)} ∪
    {(CFAILED a, FAILED) | a. True}⟩
```

```
lemma in-code-status-status-rel-iff[simp]:
  ⟨(CFOUND, b) ∈ code-status-status-rel ⟷ b = FOUND⟩
  ⟨(a, FOUND) ∈ code-status-status-rel ⟷ a = CFOUND⟩
  ⟨(CSUCCESS, b) ∈ code-status-status-rel ⟷ b = SUCCESS⟩
  ⟨(a, SUCCESS) ∈ code-status-status-rel ⟷ a = CSUCCESS⟩
  ⟨(a, FAILED) ∈ code-status-status-rel ⟷ is-cfailed a⟩
  ⟨(CFAILED C, b) ∈ code-status-status-rel ⟷ b = FAILED⟩
  ⟨proof⟩
```

```
Refinement relation fun pac-step-rel-raw :: ⟨('olbl × 'lbl) set ⟹ ('a × 'b) set ⟹ ('c × 'd) set ⟹
  ('a, 'c, 'olbl) pac-step ⟹ ('b, 'd, 'lbl) pac-step ⟹ bool⟩ where
```

$\langle \text{pac-step-rel-raw } R1 \ R2 \ R3 \ (\text{Add } p1 \ p2 \ i \ r) \ (\text{Add } p1' \ p2' \ i' \ r') \longleftrightarrow$
 $(p1, p1') \in R1 \wedge (p2, p2') \in R1 \wedge (i, i') \in R1 \wedge$
 $(r, r') \in R2 \rangle \mid$
 $\langle \text{pac-step-rel-raw } R1 \ R2 \ R3 \ (\text{Mult } p1 \ p2 \ i \ r) \ (\text{Mult } p1' \ p2' \ i' \ r') \longleftrightarrow$
 $(p1, p1') \in R1 \wedge (p2, p2') \in R2 \wedge (i, i') \in R1 \wedge$
 $(r, r') \in R2 \rangle \mid$
 $\langle \text{pac-step-rel-raw } R1 \ R2 \ R3 \ (\text{Del } p1) \ (\text{Del } p1') \longleftrightarrow$
 $(p1, p1') \in R1 \rangle \mid$
 $\langle \text{pac-step-rel-raw } R1 \ R2 \ R3 \ (\text{Extension } i \ x \ p1) \ (\text{Extension } j \ x' \ p1') \longleftrightarrow$
 $(i, j) \in R1 \wedge (x, x') \in R3 \wedge (p1, p1') \in R2 \rangle \mid$
 $\langle \text{pac-step-rel-raw } R1 \ R2 \ R3 \ - \ - \longleftrightarrow \text{False} \rangle$

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**

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

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

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

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

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

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**

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

definition *check-not-equal-dom-err* :: $\langle \text{l-list-polynomial} \Rightarrow \text{l-list-polynomial} \Rightarrow \text{l-list-polynomial} \Rightarrow \text{l-list-polynomial} \Rightarrow \text{string nres} \rangle$ **where**
 $\langle \text{check-not-equal-dom-err } p \ q \ pq \ r = \text{SPEC } (\lambda-. \text{True}) \rangle$

definition *vars-llist* :: $\langle \text{l-list-polynomial} \Rightarrow \text{string set} \rangle$ **where**
 $\langle \text{vars-llist } xs = \bigcup (\text{set } ' \text{fst } ' \text{set } xs) \rangle$

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

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

$\langle \text{check-mult-l-mult-err } p \ q \ pq \ r = \text{SPEC } (\lambda-. \text{True}) \rangle$

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

$\langle \text{check-mult-l spec } A \ \mathcal{V} \ p \ q \ i \ r = \text{do } \{$
 $\text{let } b = p \in \# \text{ dom-}m \ A \wedge i \notin \# \text{ dom-}m \ A \wedge \text{vars-llist } q \subseteq \mathcal{V} \wedge \text{vars-llist } r \subseteq \mathcal{V};$
 $\text{if } \neg b$
 $\text{then do } \{$
 $c \leftarrow \text{check-mult-l-dom-err } (p \notin \# \text{ dom-}m \ A) \ p \ (i \in \# \text{ dom-}m \ A) \ i;$
 $\text{RETURN } (\text{error-msg } i \ c)$
 $\}$
 $\text{else do } \{$
 $\text{ASSERT } (p \in \# \text{ dom-}m \ A);$
 $\text{let } p = \text{the } (\text{fmlookup } A \ p);$
 $pq \leftarrow \text{mult-poly-full } p \ q;$
 $\}$
 $\}$
 \rangle


```

}
}
}>

```

lemma *check-extension-alt-def*:

```

⟨check-extension A V i v p ≥ do {
  b ← SPEC(λb. b → i ∉# dom-m A ∧ v ∉ V);
  if ¬b
  then RETURN (False)
  else do {
    p' ← RETURN (p + Var v);
    b ← SPEC(λb. b → vars p' ⊆ V);
    if ¬b
    then RETURN (False)
    else do {
      pq ← mult-poly-spec p' p';
      let p' = - p';
      p ← add-poly-spec pq p';
      eq ← weak-equality p 0;
      if eq then RETURN(True)
      else RETURN (False)
    }
  }
}
}
}>
⟨proof⟩

```

lemma *RES-RES-RETURN-RES*: $\langle RES A \gg (\lambda T. RES (f T)) = RES (\bigcup (f ' A)) \rangle$
 ⟨proof⟩

lemma *check-add-alt-def*:

```

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

```

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

```

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

```

⟨totalp-on (insert x (set xs)) R ⇒ 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-polynomial, 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 ¬is-cfailed eq
      then RETURN (merge-cstatus st' eq, V, fmdrop (pac-src1 st) A)
      else RETURN (eq, V, A)
    }
  | Mult - - - ⇒
    do {
      r ← full-normalize-poly (pac-res st);
      q ← full-normalize-poly (pac-mult st);
      eq ← check-mult-l spec A 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:

```

 $\langle \langle K, V, R \rangle$  pac-step-rel-raw = pac-step-rel-raw K V R  $\rangle$ 
 $\langle$ proof $\rangle$ 
```

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

```

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

definition normalize-poly-l **where**

```

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

definition remap-polys-l-dom-err :: \langle string nres \rangle **where**

```

 $\langle$ remap-polys-l-dom-err = SPEC  $(\lambda-. True)$   $\rangle$ 
```

definition remap-polys-l :: \langle llist-polynomial \Rightarrow string set \Rightarrow (nat, llist-polynomial) fmap \Rightarrow

(- code-status \times string set \times (nat, llist-polynomial) fmap) nres \rangle **where**

```

 $\langle$ remap-polys-l spec =  $(\lambda V\ A. do\{$ 
  dom  $\leftarrow$  SPEC  $(\lambda dom. set-mset (dom-m\ A) \subseteq dom \wedge finite\ dom)$ ;
  failed  $\leftarrow$  SPEC  $(\lambda-::bool. True)$ ;
  if failed
  then do {
    c  $\leftarrow$  remap-polys-l-dom-err;
    RETURN (error-msg (0 :: nat) c,  $\mathcal{V}$ , fmempty)
  }
  else do {
    (b,  $\mathcal{V}$ , A)  $\leftarrow$  FOREACH dom
     $(\lambda i (b, \mathcal{V}, A').$ 
      if  $i \in \# dom-m\ A$ 
      then do {
        p  $\leftarrow$  full-normalize-poly (the (fmlookup A i));
```

```

    eq ← weak-equality-l p spec;
    V ← RETURN(V ∪ vars-llist (the (fmlookup A i)));
    RETURN(b ∨ eq, V, fmupd i p A^
  } else RETURN (b, V, A')
  (False, V, fmempty);
  RETURN (if b then CFOUND else CSUCCESS, V, A)
}})

```

definition *PAC-checker-l* **where**

```

⟨PAC-checker-l spec A b st = do {
  (S, -) ← WHILE_T
  (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
}⟩

```

9.2 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-polynomial-of-mset*:

```

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

```

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

```

⟨(A, B) ∈ fully-unsorted-poly-rel O mset-poly-rel ⇒ vars B ⊆ φ ‘ vars-llist A⟩
⟨proof⟩

```

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

```

⟨(A, B) ∈ fully-unsorted-poly-rel O mset-poly-rel ⇒
  (x1c, x1a) ∈ ⟨var-rel⟩set-rel ⇒
  RETURN (x1c ∪ vars-llist A)

```

$\leq \Downarrow (\langle \text{var-rel} \rangle \text{set-rel})$
 $(\text{SPEC } (\langle \subseteq \rangle (x1a \cup \text{vars } (B))))$
 $\langle \text{proof} \rangle$

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

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

(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*:

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

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

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

$\langle \text{proof} \rangle$

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

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

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

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

lemma *check-extension-l-check-extension*:

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

shows

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

$\langle \text{proof} \rangle$

lemma *full-normalize-poly-diff-ideal*:

fixes *dom*

assumes $\langle p, p' \rangle \in \text{fully-unsorted-poly-rel } O \ \text{mset-poly-rel}$

shows

$\langle \text{full-normalize-poly } p$
 $\leq \Downarrow (\text{sorted-poly-rel } O \ \text{mset-poly-rel})$
 $(\text{normalize-poly-spec } p') \rangle$

$\langle \text{proof} \rangle$

lemma *insort-key-rel-decomp*:

$\langle \exists ys \ zs. \ xs = ys @ zs \wedge \text{insort-key-rel } R \ x \ xs = ys @ x \# zs \rangle$

$\langle \text{proof} \rangle$

lemma *list-rel-append-same-length*:

$\langle \text{length } xs = \text{length } xs' \implies (xs @ ys, xs' @ ys') \in \langle R \rangle \text{list-rel} \longleftrightarrow (xs, xs') \in \langle R \rangle \text{list-rel} \wedge (ys, ys') \in \langle R \rangle \text{list-rel} \rangle$

$\langle \text{proof} \rangle$

lemma *term-poly-list-rel-list-relD*: $\langle (ys, cs) \in \langle \text{term-poly-list-rel} \times_r \ \text{int-rel} \rangle \text{list-rel} \implies$

$cs = \text{map } (\lambda(a, y). (\text{mset } a, y)) \ ys \rangle$

$\langle \text{proof} \rangle$

lemma *term-poly-list-rel-single*: $\langle ([x32], \{\#x32\# \}) \in \text{term-poly-list-rel} \rangle$

$\langle \text{proof} \rangle$

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*: $\langle \{\#(a, b)\# \}, v' \rangle \in \text{mset-poly-rel} \implies$
 $(\text{mset } yc, r') \in \text{mset-poly-rel} \implies$
 (r, yc)
 $\in \langle \text{unsorted-term-poly-list-rel} \times_r \text{int-rel} \rangle \text{list-rel} \implies$
 $(\text{add-mset } (a, b) (\text{mset } yc),$
 $v' + r')$
 $\in \text{mset-poly-rel}$
 $\langle \text{proof} \rangle$

lemma *fully-unsorted-poly-rel-diff*:
 $\langle ([v], v') \in \text{fully-unsorted-poly-rel} \ O \ \text{mset-poly-rel} \implies$
 $(r, r') \in \text{fully-unsorted-poly-rel} \ O \ \text{mset-poly-rel} \implies$
 $(v \# r,$
 $v' + r')$
 $\in \text{fully-unsorted-poly-rel} \ O \ \text{mset-poly-rel} \rangle$
 $\langle \text{proof} \rangle$

lemma *PAC-checker-l-step-PAC-checker-step*:

assumes

$\langle (Ast, Bst) \in \text{code-status-status-rel} \times_r \langle \text{var-rel} \rangle \text{set-rel} \times_r \text{fmap-polys-rel} \rangle$ **and**

$\langle (st, st') \in \text{pac-step-rel} \rangle$ **and**

spec: $\langle (\text{spec}, \text{spec}') \in \text{sorted-poly-rel} \ O \ \text{mset-poly-rel} \rangle$

shows

$\langle \text{PAC-checker-l-step } \text{spec } Ast \ st \leq \Downarrow (\text{code-status-status-rel} \times_r \langle \text{var-rel} \rangle \text{set-rel} \times_r \text{fmap-polys-rel})$
 $(\text{PAC-checker-step } \text{spec}' \ Bst \ st') \rangle$

$\langle \text{proof} \rangle$

lemma *code-status-status-rel-discrim-iff*:

$\langle (x1a, x1c) \in \text{code-status-status-rel} \implies \text{is-cfailed } x1a \iff \text{is-failed } x1c \rangle$

$\langle (x1a, x1c) \in \text{code-status-status-rel} \implies \text{is-cfound } x1a \iff \text{is-found } x1c \rangle$

$\langle \text{proof} \rangle$

lemma *PAC-checker-l-PAC-checker*:

assumes

$\langle (A, B) \in \langle \text{var-rel} \rangle \text{set-rel} \times_r \text{fmap-polys-rel} \rangle$ **and**

$\langle (st, st') \in \langle \text{pac-step-rel} \rangle \text{list-rel} \rangle$ **and**

$\langle (\text{spec}, \text{spec}') \in \text{sorted-poly-rel} \ O \ \text{mset-poly-rel} \rangle$ **and**

$\langle (b, b') \in \text{code-status-status-rel} \rangle$

shows

$\langle \text{PAC-checker-l } \text{spec } A \ b \ st \leq \Downarrow (\text{code-status-status-rel} \times_r \langle \text{var-rel} \rangle \text{set-rel} \times_r \text{fmap-polys-rel}) (\text{PAC-checker}$
 $\text{spec}' \ B \ b' \ st') \rangle$

$\langle \text{proof} \rangle$

end

lemma *less-than-char-of-char[code-unfold]*:

$\langle (x, y) \in \text{less-than-char} \iff (\text{of-char } x :: \text{nat}) < \text{of-char } y \rangle$

$\langle \text{proof} \rangle$

lemmas $[\text{code}] =$

$\text{add-poly-l'.simps}[\text{unfolded var-order-rel-def}]$

export-code *add-poly-l'* in *SML module-name test*

definition *full-checker-l*

$\langle \langle \text{llist-polynomial} \Rightarrow (\text{nat}, \text{llist-polynomial}) \text{ fmap} \Rightarrow (-, \text{string}, \text{nat}) \text{ pac-step list} \Rightarrow (\text{string code-status} \times -) \text{ nres} \rangle \rangle$

where

```
 $\langle \text{full-checker-l spec } A \text{ st} = \text{do} \{$   
   $\text{spec}' \leftarrow \text{full-normalize-poly spec};$   
   $(b, \mathcal{V}, A) \leftarrow \text{remap-polys-l spec}' \{\} A;$   
   $\text{if is-cfailed } b$   
   $\text{then RETURN } (b, \mathcal{V}, A)$   
   $\text{else do} \{$   
     $\text{let } \mathcal{V} = \mathcal{V} \cup \text{vars-llist spec};$   
     $\text{PAC-checker-l spec}' (\mathcal{V}, A) \text{ b st}$   
   $\}$   
 $\}$   
 $\rangle$ 
```

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**
 $\langle (\text{spec}, \text{spec}') \in \text{fully-unsorted-poly-rel } O \text{ mset-poly-rel} \rangle$

shows

$\langle \text{full-checker-l spec } A \text{ st} \leq \Downarrow (\text{code-status-status-rel} \times_r \langle \text{var-rel} \rangle \text{set-rel} \times_r \text{fmap-polys-rel}) (\text{full-checker spec}' B \text{ st}') \rangle$

$\langle \text{proof} \rangle$

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* $:: \langle \text{llist-polynomial} \Rightarrow \text{string set} \Rightarrow (\text{nat}, \text{llist-polynomial}) \text{ fmap} \Rightarrow - \text{ nres} \rangle$

where

```
 $\langle \text{remap-polys-l2 spec} = (\lambda \mathcal{V} A. \text{do} \{$   
   $n \leftarrow \text{upper-bound-on-dom } A;$   
   $b \leftarrow \text{RETURN } (n \geq 2^{\wedge} 64);$   
   $\text{if } b$   
   $\text{then do} \{$   
     $c \leftarrow \text{remap-polys-l-dom-err};$   
 $\}$   
 $\}$ 
```

```

    RETURN (error-msg (0 :: nat) c, V, fmempty)
  }
  else do {
    (b, V, A) ← nfoldli ([0..<n]) (λ-. True)
    (λi (b, V, A)'.
      if i ∈# dom-m A
      then do {
        ASSERT(fmlookup A i ≠ None);
        p ← full-normalize-poly (the (fmlookup A i));
        eq ← weak-equality-l p spec;
        V ← RETURN (V ∪ vars-llist (the (fmlookup A i)));
        RETURN(b ∨ eq, V, fmapd i p A')
      } else RETURN (b, V, A')
    )
    (False, V, fmempty);
    RETURN (if b then CFOUND else CSUCCESS, V, A)
  }
}⟩

```

lemma *remap-polys-l2-remap-polys-l*:
 ⟨remap-polys-l2 spec V A ≤ ↓ Id (remap-polys-l spec V A)⟩
 ⟨proof⟩

end

theory *PAC-Checker-Relation*
imports *PAC-Checker WB-Sort Native-Word.Uint64*
begin

10 Various Refinement Relations

When writing this, it was not possible to share the definition with the IsaSAT version.

definition *uint64-nat-rel* :: (uint64 × nat) set **where**
 ⟨uint64-nat-rel = br nat-of-uint64 (λ-. True)⟩

abbreviation *uint64-nat-assn* **where**
 ⟨uint64-nat-assn ≡ pure uint64-nat-rel⟩

instantiation *uint32* :: hashable

begin

definition *hashcode-uint32* :: ⟨uint32 ⇒ uint32⟩ **where**
 ⟨hashcode-uint32 n = n⟩

definition *def-hashmap-size-uint32* :: ⟨uint32 itself ⇒ nat⟩ **where**
 ⟨def-hashmap-size-uint32 = (λ-. 16)⟩
 — same as nat

instance

⟨proof⟩

end

instantiation *uint64* :: hashable

begin

context

includes *bit-operations-syntax*

begin

definition *hashcode-uint64* :: $\langle \text{uint64} \Rightarrow \text{uint32} \rangle$ **where**

$\langle \text{hashcode-uint64 } n = (\text{uint32-of-nat } (\text{nat-of-uint64 } ((n) \text{ AND } ((2 :: \text{uint64})^{\wedge} 32 - 1)))) \rangle$

end

definition *def-hashmap-size-uint64* :: $\langle \text{uint64} \text{ itself} \Rightarrow \text{nat} \rangle$ **where**

$\langle \text{def-hashmap-size-uint64} = (\lambda -. 16) \rangle$

— same as *nat*

instance

$\langle \text{proof} \rangle$

end

lemma *word-nat-of-uint64-Rep-inject[simp]*: $\langle \text{nat-of-uint64 } ai = \text{nat-of-uint64 } bi \longleftrightarrow ai = bi \rangle$

$\langle \text{proof} \rangle$

instance *uint64* :: *heap*

$\langle \text{proof} \rangle$

instance *uint64* :: *semiring-numeral*

$\langle \text{proof} \rangle$

lemma *nat-of-uint64-012[simp]*: $\langle \text{nat-of-uint64 } 0 = 0 \rangle \langle \text{nat-of-uint64 } 2 = 2 \rangle \langle \text{nat-of-uint64 } 1 = 1 \rangle$

$\langle \text{proof} \rangle$

definition *uint64-of-nat-conv* **where**

$\langle \text{uint64-of-nat-conv } (x :: \text{nat}) = x \rangle$

lemma *less-upper-bintrunc-id*: $\langle n < 2^{\wedge} b \implies n \geq 0 \implies \text{take-bit } b \ n = n \rangle$ **for** $n :: \text{int}$

$\langle \text{proof} \rangle$

lemma *nat-of-uint64-uint64-of-nat-id*: $\langle n < 2^{\wedge} 64 \implies \text{nat-of-uint64 } (\text{uint64-of-nat } n) = n \rangle$

$\langle \text{proof} \rangle$

lemma *[sepref-fr-rules]*:

$\langle (\text{return } o \ \text{uint64-of-nat}, \text{RETURN } o \ \text{uint64-of-nat-conv}) \in [\lambda a. a < 2^{\wedge} 64]_a \ \text{nat-assn}^k \rightarrow \text{uint64-nat-assn} \rangle$

$\langle \text{proof} \rangle$

definition *string-rel* :: $\langle (\text{String.literal} \times \text{string}) \ \text{set} \rangle$ **where**

$\langle \text{string-rel} = \{(x, y). y = \text{String.explode } x\} \rangle$

abbreviation *string-assn* :: $\langle \text{string} \Rightarrow \text{String.literal} \Rightarrow \text{assn} \rangle$ **where**

$\langle \text{string-assn} \equiv \text{pure } \text{string-rel} \rangle$

lemma *eq-string-eq*:

$\langle ((=), (=)) \in \text{string-rel} \rightarrow \text{string-rel} \rightarrow \text{bool-rel} \rangle$

$\langle \text{proof} \rangle$

lemmas *eq-string-eq-hnr* =

eq-string-eq[*sepref-import-param*]

definition *string2-rel* :: $\langle (\text{string} \times \text{string}) \ \text{set} \rangle$ **where**

$\langle \text{string2-rel} \equiv \langle \text{Id} \rangle \text{list-rel} \rangle$

abbreviation *string2-assn* :: $\langle \text{string} \Rightarrow \text{string} \Rightarrow \text{assn} \rangle$ **where**
 $\langle \text{string2-assn} \equiv \text{pure string2-rel} \rangle$

abbreviation *monom-rel* **where**
 $\langle \text{monom-rel} \equiv \langle \text{string-rel} \rangle \text{list-rel} \rangle$

abbreviation *monom-assn* **where**
 $\langle \text{monom-assn} \equiv \text{list-assn string-assn} \rangle$

abbreviation *monomial-rel* **where**
 $\langle \text{monomial-rel} \equiv \text{monom-rel} \times_r \text{int-rel} \rangle$

abbreviation *monomial-assn* **where**
 $\langle \text{monomial-assn} \equiv \text{monom-assn} \times_a \text{int-assn} \rangle$

abbreviation *poly-rel* **where**
 $\langle \text{poly-rel} \equiv \langle \text{monomial-rel} \rangle \text{list-rel} \rangle$

abbreviation *poly-assn* **where**
 $\langle \text{poly-assn} \equiv \text{list-assn monomial-assn} \rangle$

lemma *poly-assn-alt-def*:
 $\langle \text{poly-assn} = \text{pure poly-rel} \rangle$
 $\langle \text{proof} \rangle$

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-Polynomials-Term.less-char*

definition *less-eq-char* **where** [*symmetric, simp*]: *less-eq-char* = *PAC-Polynomials-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 $\langle (a, b) \in \langle \text{string-rel} \rangle \text{list-rel} \rangle \langle (a', b') \in \langle \text{string-rel} \rangle \text{list-rel} \rangle$
shows $\langle a < a' \longleftrightarrow b < b' \rangle$
 $\langle \text{proof} \rangle$

lemma *string-rel-le*[*sepref-import-param*]:
shows $\langle ((<), (<)) \in \langle \text{string-rel} \rangle \text{list-rel} \rightarrow \langle \text{string-rel} \rangle \text{list-rel} \rightarrow \text{bool-rel} \rangle$
 $\langle \text{proof} \rangle$

lemma [*sepref-import-param*]:
assumes $\langle \text{CONSTRAINT IS-LEFT-UNIQUE } R \rangle \langle \text{CONSTRAINT IS-RIGHT-UNIQUE } R \rangle$
shows $\langle (\text{remove1}, \text{remove1}) \in R \rightarrow \langle R \rangle \text{list-rel} \rightarrow \langle R \rangle \text{list-rel} \rangle$
 $\langle \text{proof} \rangle$

instantiation *pac-step* :: (*heap, heap, heap*) *heap*
begin

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

end

end

theory *PAC-Assoc-Map-Rel*
imports *PAC-Map-Rel*
begin

11 Hash Map as association list

type-synonym (*'k, 'v*) *hash-assoc* = $\langle ('k \times 'v) \text{ list} \rangle$

definition *hassoc-map-rel-raw* :: $\langle (('k, 'v) \text{ hash-assoc} \times -) \text{ set} \rangle$ **where**
 $\langle \text{hassoc-map-rel-raw} = \text{br map-of } (\lambda-. \text{ True}) \rangle$

abbreviation *hassoc-map-assn* :: $\langle ('k \Rightarrow 'v \text{ option}) \Rightarrow ('k, 'v) \text{ hash-assoc} \Rightarrow \text{assn} \rangle$ **where**
 $\langle \text{hassoc-map-assn} \equiv \text{pure } (\text{hassoc-map-rel-raw}) \rangle$

lemma *hassoc-map-rel-raw-empty*[*simp*]:
 $\langle ([], m) \in \text{hassoc-map-rel-raw} \longleftrightarrow m = \text{Map.empty} \rangle$
 $\langle (p, \text{Map.empty}) \in \text{hassoc-map-rel-raw} \longleftrightarrow p = [] \rangle$
 $\langle \text{hassoc-map-assn } \text{Map.empty } [] = \text{emp} \rangle$
 $\langle \text{proof} \rangle$

definition *hassoc-new* :: $\langle ('k, 'v) \text{ hash-assoc } \text{Heap} \rangle$ **where**
 $\langle \text{hassoc-new} = \text{return } [] \rangle$

lemma *precise-hassoc-map-assn*: $\langle \text{precise } \text{hassoc-map-assn} \rangle$
 $\langle \text{proof} \rangle$

definition *hassoc-isEmpty* :: $\langle ('k \times 'v) \text{ list} \Rightarrow \text{bool } \text{Heap} \rangle$ **where**
 $\langle \text{hassoc-isEmpty } ht \equiv \text{return } (\text{length } ht = 0) \rangle$

interpretation *hassoc*: *bind-map-empty* *hassoc-map-assn* *hassoc-new*

⟨proof⟩

interpretation *hassoc*: *bind-map-is-empty* *hassoc-map-assn* *hassoc-isEmpty*

⟨proof⟩

definition *op-assoc-empty* \equiv *IICF-Map.op-map-empty*

interpretation *hassoc*: *map-custom-empty* *op-assoc-empty*

⟨proof⟩

lemmas [*sepref-fr-rules*] = *hassoc.empty-hnr*[*folded op-assoc-empty-def*]

definition *hassoc-update* :: $'k \Rightarrow 'v \Rightarrow ('k, 'v) \text{ hash-assoc} \Rightarrow ('k, 'v) \text{ hash-assoc Heap}$ **where**
hassoc-update *k v ht* = *return* ((*k*, *v*) # *ht*)

lemma *hassoc-map-assn-Cons*:

⟨*hassoc-map-assn* (*m*) (*p*) \Longrightarrow_A *hassoc-map-assn* (*m*(*k* \mapsto *v*)) ((*k*, *v*) # *p*) * *true*⟩
⟨proof⟩

interpretation *hassoc*: *bind-map-update* *hassoc-map-assn* *hassoc-update*

⟨proof⟩

definition *hassoc-delete* :: $'k \Rightarrow ('k, 'v) \text{ hash-assoc} \Rightarrow ('k, 'v) \text{ hash-assoc Heap}$ **where**
hassoc-delete *k ht* = *return* (*filter* ($\lambda(a, b). a \neq k$) *ht*)

lemma *hassoc-map-of-filter-all*:

⟨*map-of* *p* |' (- {*k*}) = *map-of* (*filter* ($\lambda(a, b). a \neq k$) *p*)⟩
⟨proof⟩

lemma *hassoc-map-assn-hassoc-delete*: ⟨⟨*hassoc-map-assn* *m p*⟩ *hassoc-delete* *k p* ⟨*hassoc-map-assn* (*m* |' (- {*k*}))⟩⟩_{*t*}⟩

⟨proof⟩

interpretation *hassoc*: *bind-map-delete* *hassoc-map-assn* *hassoc-delete*

⟨proof⟩

definition *hassoc-lookup* :: $'k \Rightarrow ('k, 'v) \text{ hash-assoc} \Rightarrow 'v \text{ option Heap}$ **where**

⟨*hassoc-lookup* *k ht* = *return* (*map-of* *ht k*)⟩

lemma *hassoc-map-assn-hassoc-lookup*:

⟨⟨*hassoc-map-assn* *m p*⟩ *hassoc-lookup* *k p* ⟨ $\lambda r. \text{hassoc-map-assn } m \text{ } p * \uparrow (r = m \text{ } k)$ ⟩⟩_{*t*}⟩
⟨proof⟩

interpretation *hassoc*: *bind-map-lookup* *hassoc-map-assn* *hassoc-lookup*

⟨proof⟩

⟨ML⟩

interpretation *hassoc*: *gen-contains-key-by-lookup* *hassoc-map-assn* *hassoc-lookup*

⟨proof⟩

⟨ML⟩

interpretation *hassoc*: *bind-map-contains-key* *hassoc-map-assn* *hassoc.contains-key*
 ⟨*proof*⟩

11.1 Conversion from assoc to other map

definition *hash-of-assoc-map* **where**
 ⟨*hash-of-assoc-map* *xs* = *fold* ($\lambda(k, v) m. \text{if } m \ k \neq \text{None then } m \ \text{else } m(k \mapsto v)$) *xs* *Map.empty*⟩

lemma *map-upd-map-add-left*:
 ⟨ $m(a \mapsto b) ++ m' = m ++ (\text{if } a \notin \text{dom } m' \text{ then } m'(a \mapsto b) \ \text{else } m')$ ⟩
 ⟨*proof*⟩

lemma *fold-map-of-alt*:
 ⟨*fold* ($\lambda(k, v) m. \text{if } m \ k \neq \text{None then } m \ \text{else } m(k \mapsto v)$) *xs* *m'* = *map-of* *xs* ++ *m'*⟩
 ⟨*proof*⟩

lemma *map-of-alt-def*:
 ⟨*map-of* *xs* = *hash-of-assoc-map* *xs*⟩
 ⟨*proof*⟩

definition *hashmap-conv* **where**
 [*simp*]: ⟨*hashmap-conv* *x* = *x*⟩

lemma *hash-of-assoc-map-id*:
 ⟨(*hash-of-assoc-map*, *hashmap-conv*) ∈ *hassoc-map-rel-raw* → *Id*⟩
 ⟨*proof*⟩

definition *hassoc-map-rel* **where**
hassoc-map-rel-internal-def:
 ⟨*hassoc-map-rel* *K* *V* = *hassoc-map-rel-raw* *O* ⟨*K*, *V*⟩*map-rel*⟩

lemma *hassoc-map-rel-def*:
 ⟨⟨*K*, *V*⟩ *hassoc-map-rel* = *hassoc-map-rel-raw* *O* ⟨*K*, *V*⟩*map-rel*⟩
 ⟨*proof*⟩

end

theory *PAC-Checker-Init*
imports *PAC-Checker* *WB-Sort* *PAC-Checker-Relation*
begin

12 Initial Normalisation of Polynomials

12.1 Sorting

Adapted from the theory *HOL-ex.MergeSort* by Tobias Nipkow. We did not change much, but we refine it to executable code and try to improve efficiency.

fun *merge* :: *a* list ⇒ *a* list ⇒ *a* list
where
merge *f* (*x*#*xs*) (*y*#*ys*) =
 (*if* *f* *x* *y* *then* *x* # *merge* *f* *xs* (*y*#*ys*) *else* *y* # *merge* *f* (*x*#*xs*) *ys*)
 | *merge* *f* *xs* [] = *xs*
 | *merge* *f* [] *ys* = *ys*

lemma *mset-merge* [*simp*]:
 $mset (merge f xs ys) = mset xs + mset ys$
 ⟨*proof*⟩

lemma *set-merge* [*simp*]:
 $set (merge f xs ys) = set xs \cup set ys$
 ⟨*proof*⟩

lemma *sorted-merge*:
 $transp f \implies (\bigwedge x y. f x y \vee f y x) \implies$
 $sorted-wrt f (merge f xs ys) \longleftrightarrow sorted-wrt f xs \wedge sorted-wrt f ys$
 ⟨*proof*⟩

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* :: $\langle - \Rightarrow nat \Rightarrow nat \Rightarrow ('a \times 'a \times 'a) \Rightarrow ('a \times 'a \times 'a) \rangle$ **where**

$\langle swap-ternary f m n =$
 $(if (m = 0 \wedge n = 1)$
 $then (\lambda(a, b, c). if f a b then (a, b, c)$
 $else (b,a,c))$
 $else if (m = 0 \wedge n = 2)$
 $then (\lambda(a, b, c). if f a c then (a, b, c)$
 $else (c,b,a))$
 $else if (m = 1 \wedge n = 2)$
 $then (\lambda(a, b, c). if f b c then (a, b, c)$
 $else (a,c,b))$
 $else (\lambda(a, b, c). (a,b,c))) \rangle$

fun *msort2* :: $- \Rightarrow 'a list \Rightarrow 'a list$

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 $xs :: \langle 'a :: linorder list \rangle$

shows $\langle msort (\leq) xs = msort2 (\leq) xs \rangle$

⟨proof⟩

lemma *sorted-msort*:

$\text{transp } f \implies (\bigwedge x y. f x y \vee f y x) \implies$
 $\text{sorted-wrt } f \text{ (msort } f \text{ } xs)$
⟨proof⟩

lemma *mset-msort[simp]*:

$\text{mset (msort } f \text{ } xs) = \text{mset } xs$
⟨proof⟩

12.2 Sorting applied to monomials

lemma *merge-coeffs-alt-def*:

⟨RETURN o merge-coeffs⟩ $p =$
 $\text{REC}_T(\lambda f p.$
 (case p of
 [] \Rightarrow RETURN []
 | [-] \Rightarrow RETURN p
 | ((xs, n) # (ys, m) # p) \Rightarrow
 (if $xs = ys$
 then if $n + m \neq 0$ then $f ((xs, n + m) \# p)$ else $f p$
 else do { $p \leftarrow f ((ys, m) \# p)$; RETURN ((xs, n) # p)}))
 p)
⟨proof⟩

lemma *hn-invalid-recover*:

⟨is-pure $R \implies \text{hn-invalid } R = (\lambda x y. R x y * \text{true})$ ⟩
⟨is-pure $R \implies \text{invalid-assn } R = (\lambda x y. R x y * \text{true})$ ⟩
⟨proof⟩

lemma *safe-poly-vars*:

shows
 [*safe-constraint-rules*]:
 is-pure (poly-assn) **and**
 [*safe-constraint-rules*]:
 is-pure (monom-assn) **and**
 [*safe-constraint-rules*]:
 is-pure (monomial-assn) **and**
 [*safe-constraint-rules*]:
 is-pure string-assn
⟨proof⟩

lemma *invalid-assn-distrib*:

⟨invalid-assn monom-assn \times_a invalid-assn int-assn = invalid-assn (monom-assn \times_a int-assn)⟩
⟨proof⟩

lemma *WTF-RF-recover*:

⟨hn-ctxt (invalid-assn monom-assn \times_a invalid-assn int-assn) xb
 $x'a \vee_A$
 hn-ctxt monomial-assn $xb x'a \implies_t$
 hn-ctxt (monomial-assn) $xb x'a$ ⟩
⟨proof⟩

lemma *WTF-RF*:

⟨hn-ctxt (invalid-assn monom-assn \times_a invalid-assn int-assn) $xb x'a *$

$(hn\text{-invalid poly-assign } la\ l'a * hn\text{-invalid int-assign } a2'\ a2 * \\
hn\text{-invalid monom-assign } a1'\ a1 * \\
hn\text{-invalid poly-assign } l\ l' * \\
hn\text{-invalid monomial-assign } xa\ x' * \\
hn\text{-invalid poly-assign } ax\ px) \implies_t \\
hn\text{-ctxt (monomial-assign) } xb\ x'a * \\
hn\text{-ctxt poly-assign} \\
la\ l'a * \\
hn\text{-ctxt poly-assign } l\ l' * \\
(hn\text{-invalid int-assign } a2'\ a2 * \\
hn\text{-invalid monom-assign } a1'\ a1 * \\
hn\text{-invalid monomial-assign } xa\ x' * \\
hn\text{-invalid poly-assign } ax\ px) \rangle \\
\langle hn\text{-ctxt (invalid-assign monom-assign } \times_a\ \text{invalid-assign int-assign) } xa\ x' * \\
(hn\text{-ctxt poly-assign } l\ l' * hn\text{-invalid poly-assign } ax\ px) \implies_t \\
hn\text{-ctxt (monomial-assign) } xa\ x' * \\
hn\text{-ctxt poly-assign } l\ l' * \\
hn\text{-ctxt poly-assign } ax\ px * \\
emp \rangle \\
\langle proof \rangle$

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 $\langle RETURN\ o\ merge\ coeffs \rangle \\
:: \langle poly\ assign^d \rightarrow_a\ poly\ assign \rangle \\
\langle proof \rangle$

definition *full-quicksort-poly where*

$\langle full\ quicksort\ poly = full\ quicksort\ ref\ (\lambda x\ y.\ x = y \vee (x, y) \in term\ order\ rel)\ fst \rangle$

lemma *down-eq-id-list-rel*: $\langle \Downarrow ((Id)\ list\ rel)\ x = x \rangle$

$\langle proof \rangle$

definition *quicksort-poly*: $\langle nat \Rightarrow nat \Rightarrow llist\ polynomial \Rightarrow (llist\ polynomial)\ nres \rangle$ **where**

$\langle quicksort\ poly\ x\ y\ z = quicksort\ ref\ (\leq)\ fst\ (x, y, z) \rangle$

term *partition-between-ref*

definition *partition-between-poly* :: $\langle nat \Rightarrow nat \Rightarrow llist\ polynomial \Rightarrow (llist\ polynomial \times nat)\ nres \rangle$ **where**

$\langle partition\ between\ poly = partition\ between\ ref\ (\leq)\ fst \rangle$

definition *partition-main-poly* :: $\langle nat \Rightarrow nat \Rightarrow llist\ polynomial \Rightarrow (llist\ polynomial \times nat)\ nres \rangle$ **where**

$\langle partition\ main\ poly = partition\ main\ (\leq)\ fst \rangle$

lemma *string-list-trans*:

$\langle (xa :: char\ list\ list,\ ya) \in lexord\ (lexord\ \{(x, y).\ x < y\}) \implies \\
(ya, z) \in lexord\ (lexord\ \{(x, y).\ x < y\}) \implies \\
(xa, z) \in lexord\ (lexord\ \{(x, y).\ x < y\}) \rangle \\
\langle proof \rangle$

lemma *full-quicksort-sort-poly-spec*:

$\langle (full\ quicksort\ poly,\ sort\ poly\ spec) \in \langle Id \rangle list\ rel \rightarrow_f\ \langle \langle Id \rangle list\ rel \rangle nres\ rel \rangle \\
\langle proof \rangle$

12.3 Lifting to polynomials

definition *merge-sort-poly* :: $\langle \rightarrow \rangle$ **where**
 $\langle \text{merge-sort-poly} = \text{msort } (\lambda a b. \text{fst } a \leq \text{fst } b) \rangle$

definition *merge-monoms-poly* :: $\langle \rightarrow \rangle$ **where**
 $\langle \text{merge-monoms-poly} = \text{msort } (\leq) \rangle$

definition *merge-poly* :: $\langle \rightarrow \rangle$ **where**
 $\langle \text{merge-poly} = \text{merge } (\lambda a b. \text{fst } a \leq \text{fst } b) \rangle$

definition *merge-monoms* :: $\langle \rightarrow \rangle$ **where**
 $\langle \text{merge-monoms} = \text{merge } (\leq) \rangle$

definition *msort-poly-impl* :: $\langle (\text{String.literal list} \times \text{int}) \text{ list} \Rightarrow \rightarrow \rangle$ **where**
 $\langle \text{msort-poly-impl} = \text{msort } (\lambda a b. \text{fst } a \leq \text{fst } b) \rangle$

definition *msort-monoms-impl* :: $\langle (\text{String.literal list}) \Rightarrow \rightarrow \rangle$ **where**
 $\langle \text{msort-monoms-impl} = \text{msort } (\leq) \rangle$

lemma *msort-poly-impl-alt-def*:

$\langle \text{msort-poly-impl } xs =$
 (case xs of
 | [] \Rightarrow []
 | [a] \Rightarrow [a]
 | [a,b] \Rightarrow if $\text{fst } a \leq \text{fst } b$ then [a,b] else [b,a]
 | xs \Rightarrow merge-poly
 (msort-poly-impl (take ((length xs) div 2) xs))
 (msort-poly-impl (drop ((length xs) div 2) xs)))
 $\langle \text{proof} \rangle$

lemma *le-term-order-rel'*:

$\langle (\leq) = (\lambda x y. x = y \vee \text{term-order-rel}' x y) \rangle$
 $\langle \text{proof} \rangle$

fun *lexord-eq* **where**

$\langle \text{lexord-eq } [] - = \text{True} \rangle$ |
 $\langle \text{lexord-eq } (x \# xs) (y \# ys) = (x < y \vee (x = y \wedge \text{lexord-eq } xs ys)) \rangle$ |
 $\langle \text{lexord-eq } - - = \text{False} \rangle$

lemma [*simp*]:

$\langle \text{lexord-eq } [] [] = \text{True} \rangle$
 $\langle \text{lexord-eq } (a \# b) [] = \text{False} \rangle$
 $\langle \text{lexord-eq } [] (a \# b) = \text{True} \rangle$
 $\langle \text{proof} \rangle$

lemma *var-order-rel'*:

$\langle (\leq) = (\lambda x y. x = y \vee (x,y) \in \text{var-order-rel}) \rangle$
 $\langle \text{proof} \rangle$

lemma *var-order-rel''*:

$\langle (x,y) \in \text{var-order-rel} \iff x < y \rangle$
 $\langle \text{proof} \rangle$

lemma *lexord-eq-alt-def1*:

$\langle a \leq b = \text{lexord-eq } a \ b \rangle$ **for** $a \ b :: \langle \text{String.literal list} \rangle$
 $\langle \text{proof} \rangle$

lemma *lexord-eq-alt-def2*:

$\langle (\text{RETURN } \text{oo } \text{lexord-eq}) \ x \ y =$
 $\text{RECT } (\lambda f \ (x \ y)).$
 $\text{case } (x \ y) \ \text{of}$
 $\quad ([], \ -) \Rightarrow \text{RETURN } \text{True}$
 $\quad | \ (x \ \# \ x \ y, \ y \ \# \ y \ x) \Rightarrow$
 $\quad \quad \text{if } x < y \ \text{then } \text{RETURN } \text{True}$
 $\quad \quad \text{else if } x = y \ \text{then } f \ (x \ y) \ \text{else } \text{RETURN } \text{False}$
 $\quad | \ - \Rightarrow \text{RETURN } \text{False}$
 $\quad (x \ y) \rangle$
 $\langle \text{proof} \rangle$

definition *var-order'* **where**

$[\text{simp}]: \langle \text{var-order}' = \text{var-order} \rangle$

lemma *var-order-rel[def-pat-rules]*:

$\langle (\in) \$ (x, y) \$ \text{var-order-rel} \equiv \text{var-order}' \$ x \$ y \rangle$
 $\langle \text{proof} \rangle$

lemma *var-order-rel-alt-def*:

$\langle \text{var-order-rel} = \text{p2rel } \text{char.lexordp} \rangle$
 $\langle \text{proof} \rangle$

lemma *var-order-rel-var-order*:

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

lemma *var-order-string-le[sepref-import-param]*:

$\langle ((<), \text{var-order}') \in \text{string-rel} \rightarrow \text{string-rel} \rightarrow \text{bool-rel} \rangle$
 $\langle \text{proof} \rangle$

lemma $[\text{sepref-import-param}]$:

$\langle ((\leq), (\leq)) \in \text{monom-rel} \rightarrow \text{monom-rel} \rightarrow \text{bool-rel} \rangle$
 $\langle \text{proof} \rangle$

lemma $[\text{sepref-import-param}]$:

$\langle ((<), (<)) \in \text{string-rel} \rightarrow \text{string-rel} \rightarrow \text{bool-rel} \rangle$
 $\langle \text{proof} \rangle$

lemma *lexordp-char-char*: $\langle \text{ord-class.lexordp} = \text{char.lexordp} \rangle$

$\langle \text{proof} \rangle$

lemma $[\text{sepref-import-param}]$:

$\langle ((\leq), (\leq)) \in \text{string-rel} \rightarrow \text{string-rel} \rightarrow \text{bool-rel} \rangle$
 $\langle \text{proof} \rangle$

sepref-register *lexord-eq*

sepref-definition *lexord-eq-term*

is $\langle \text{uncurry } (\text{RETURN } \text{oo } \text{lexord-eq}) \rangle$
 $:: \langle \text{monom-assn}^k \ *_a \ \text{monom-assn}^k \ \rightarrow_a \ \text{bool-assn} \rangle$

⟨proof⟩

declare *lexord-eq-term.refine*[*sepref-fr-rules*]

lemmas [*code del*] = *msort-poly-impl-def msort-monomys-impl-def*

lemmas [*code*] =

msort-poly-impl-def[*unfolded lexord-eq-alt-def1*[*abs-def*]]

msort-monomys-impl-def[*unfolded msort-msort2*]

lemma *term-order-rel-trans*:

⟨(*a*, *aa*) ∈ *term-order-rel* ⇒

(*aa*, *ab*) ∈ *term-order-rel* ⇒ (*a*, *ab*) ∈ *term-order-rel*⟩

⟨proof⟩

lemma *merge-sort-poly-sort-poly-spec*:

⟨(*RETURN o merge-sort-poly*, *sort-poly-spec*) ∈ ⟨*Id*⟩*list-rel* →_{*f*} ⟨⟨*Id*⟩*list-rel*⟩*nres-rel*⟩

⟨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⟩
⟨proof⟩

lemma merge-sort-poly[sepref-import-param]:
⟨(msort-poly-impl, merge-sort-poly)
∈ poly-rel → poly-rel⟩
⟨proof⟩

lemmas [sepref-fr-rules] = merge-sort-poly[FCOMP merge-sort-poly-sort-poly-spec]

sepref-definition partition-main-poly-impl
is ⟨uncurry2 partition-main-poly⟩
:: ⟨nat-assn^k *_a nat-assn^k *_a poly-assn^k →_a prod-assn poly-assn nat-assn ⟩
⟨proof⟩

declare partition-main-poly-impl.refine[sepref-fr-rules]

sepref-definition partition-between-poly-impl
is ⟨uncurry2 partition-between-poly⟩
:: ⟨nat-assn^k *_a nat-assn^k *_a poly-assn^k →_a prod-assn poly-assn nat-assn ⟩
⟨proof⟩

declare partition-between-poly-impl.refine[sepref-fr-rules]

sepref-definition quicksort-poly-impl
is ⟨uncurry2 quicksort-poly⟩
:: ⟨nat-assn^k *_a nat-assn^k *_a poly-assn^k →_a poly-assn⟩
⟨proof⟩

lemmas [sepref-fr-rules] = quicksort-poly-impl.refine

sepref-register quicksort-poly

sepref-definition full-quicksort-poly-impl
is ⟨full-quicksort-poly⟩
:: ⟨poly-assn^k →_a poly-assn⟩
⟨proof⟩

lemmas *sort-poly-spec-hnr* =
full-quicksort-poly-impl.refine[*FCOMP full-quicksort-sort-poly-spec*]

declare *merge-coeffs-impl.refine*[*sepref-fr-rules*]

sepref-definition *normalize-poly-impl*
is $\langle \text{normalize-poly} \rangle$
 $:: \langle \text{poly-assn}^k \rightarrow_a \text{poly-assn} \rangle$
 $\langle \text{proof} \rangle$

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

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*:
 $\langle (\text{full-quicksort-vars}, \text{sort-coeff}) \in \langle \text{Id} \rangle \text{list-rel} \rightarrow_f \langle \langle \text{Id} \rangle \text{list-rel} \rangle \text{nres-rel} \rangle$
 $\langle \text{proof} \rangle$

sepref-definition *partition-main-vars-impl*
is $\langle \text{uncurry2 } \text{partition-main-vars} \rangle$
 $:: \langle \text{nat-assn}^k *_{\text{a}} \text{nat-assn}^k *_{\text{a}} (\text{monom-assn})^k \rightarrow_a \text{prod-assn } (\text{monom-assn}) \text{ nat-assn} \rangle$
 $\langle \text{proof} \rangle$

declare *partition-main-vars-impl.refine*[*sepref-fr-rules*]

sepref-definition *partition-between-vars-impl*
is $\langle \text{uncurry2 } \text{partition-between-vars} \rangle$
 $:: \langle \text{nat-assn}^k *_{\text{a}} \text{nat-assn}^k *_{\text{a}} \text{monom-assn}^k \rightarrow_a \text{prod-assn } \text{monom-assn } \text{nat-assn} \rangle$
 $\langle \text{proof} \rangle$

declare *partition-between-vars-impl.refine*[*sepref-fr-rules*]

sepref-definition *quicksort-vars-impl*

is $\langle \text{uncurry2 } \textit{quicksort-vars} \rangle$
:: $\langle \textit{nat-assn}^k *_{\alpha} \textit{nat-assn}^k *_{\alpha} \textit{monom-assn}^k \rightarrow_{\alpha} \textit{monom-assn} \rangle$
 $\langle \textit{proof} \rangle$

lemmas [*sepref-fr-rules*] = *quicksort-vars-impl.refine*

sepref-register *quicksort-vars*

lemma *le-var-order-rel*:

$\langle (\leq) = (\lambda x y. x = y \vee (x, y) \in \textit{var-order-rel}) \rangle$
 $\langle \textit{proof} \rangle$

sepref-definition *full-quicksort-vars-impl*

is $\langle \textit{full-quicksort-vars} \rangle$
:: $\langle \textit{monom-assn}^k \rightarrow_{\alpha} \textit{monom-assn} \rangle$
 $\langle \textit{proof} \rangle$

lemmas *sort-vars-spec-hnr* =

full-quicksort-vars-impl.refine[*FCOMP full-quicksort-sort-vars-spec*]

lemma *string-rel-order-map*:

$\langle (x, a) \in \textit{string-rel} \implies$
 $(y, aa) \in \textit{string-rel} \implies$
 $x \leq y \iff a \leq aa \rangle$
 $\langle \textit{proof} \rangle$

lemma *merge-monoms-merge-monoms*:

$\langle (\textit{merge-monoms}, \textit{merge-monoms}) \in \textit{monom-rel} \rightarrow \textit{monom-rel} \rightarrow \textit{monom-rel} \rangle$
 $\langle \textit{proof} \rangle$

lemma *merge-monoms-merge-monoms2*:

$\langle (a, b) \in \textit{monom-rel} \implies (a', b') \in \textit{monom-rel} \implies$
 $(\textit{merge-monoms } a \ a', \textit{merge-monoms } b \ b') \in \textit{monom-rel} \rangle$
 $\langle \textit{proof} \rangle$

lemma *msort-monoms-impl*:

$\langle (\textit{msort-monoms-impl}, \textit{merge-monoms-poly})$
 $\in \textit{monom-rel} \rightarrow \textit{monom-rel} \rangle$
 $\langle \textit{proof} \rangle$

lemma *merge-sort-monoms-sort-monoms-spec*:

$\langle (\textit{RETURN } o \ \textit{merge-monoms-poly}, \textit{sort-coeff}) \in \langle \textit{Id} \rangle \textit{list-rel} \rightarrow_f \langle \langle \textit{Id} \rangle \textit{list-rel} \rangle \textit{nres-rel} \rangle$
 $\langle \textit{proof} \rangle$

sepref-register *sort-coeff*

lemma [*sepref-fr-rules*]:

$\langle (\textit{return } o \ \textit{msort-monoms-impl}, \textit{sort-coeff}) \in \textit{monom-assn}^k \rightarrow_{\alpha} \textit{monom-assn} \rangle$
 $\langle \textit{proof} \rangle$

```

sepref-definition sort-all-coeffs-impl
  is ⟨sort-all-coeffs⟩
  :: ⟨poly-assnk →a poly-assn⟩
  ⟨proof⟩

```

```

declare sort-all-coeffs-impl.refine[sepref-fr-rules]

```

```

lemma merge-coeffs0-alt-def:
  ⟨(RETURN o merge-coeffs0) p =
    RECT(λf p.
      (case p of
        [] ⇒ RETURN []
      | [p] ⇒ if snd p = 0 then RETURN [] else RETURN [p]
      | ((xs, n) # (ys, m) # p) ⇒
        (if xs = ys
          then if n + m ≠ 0 then f ((xs, n + m) # p) else f p
          else if n = 0 then
            do {p ← f ((ys, m) # p);
              RETURN p}
          else do {p ← f ((ys, m) # p);
                 RETURN ((xs, n) # p)})
      p
    )
  ⟨proof⟩

```

Again, Sepref does not understand what is going here.

```

sepref-definition merge-coeffs0-impl
  is ⟨RETURN o merge-coeffs0⟩
  :: ⟨poly-assnk →a poly-assn⟩
  ⟨proof⟩

```

```

declare merge-coeffs0-impl.refine[sepref-fr-rules]

```

```

sepref-definition fully-normalize-poly-impl
  is ⟨full-normalize-poly⟩
  :: ⟨poly-assnk →a poly-assn⟩
  ⟨proof⟩

```

```

declare fully-normalize-poly-impl.refine[sepref-fr-rules]

```

```

end

```

```

theory PAC-Version
  imports Main
begin

```

This code was taken from IsaFoR. However, for the AFP, we use the version name *AFP*, instead of a mercurial version.

```

⟨ML⟩

```

```

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 and adding more efficient data structures (mostly replacing the set of variables by a more efficient hash map).

```

abbreviation vars-assn where
  ⟨vars-assn ≡ hs.assn string-assn⟩

```

```

fun vars-of-monom-in where
  ⟨vars-of-monom-in [] - = True⟩ |
  ⟨vars-of-monom-in (x # xs) V ⟷ x ∈ V ∧ vars-of-monom-in xs V⟩

```

```

fun vars-of-poly-in where
  ⟨vars-of-poly-in [] - = True⟩ |
  ⟨vars-of-poly-in ((x, -) # xs) V ⟷ vars-of-monom-in x V ∧ vars-of-poly-in xs V⟩

```

```

lemma vars-of-monom-in-alt-def:
  ⟨vars-of-monom-in xs V ⟷ set xs ⊆ V⟩
  ⟨proof⟩

```

```

lemma vars-llist-alt-def:
  ⟨vars-llist xs ⊆ V ⟷ vars-of-poly-in xs V⟩
  ⟨proof⟩

```

```

lemma vars-of-monom-in-alt-def2:
  ⟨vars-of-monom-in xs V ⟷ fold (λx b. b ∧ x ∈ V) xs True⟩
  ⟨proof⟩

```

```

sempref-definition vars-of-monom-in-impl
  is ⟨uncurry (RETURN oo vars-of-monom-in)⟩
  :: ⟨(list-assn string-assn)k *a vars-assnk →a bool-assn⟩
  ⟨proof⟩

```

```

declare vars-of-monom-in-impl.refine[sempref-fr-rules]

```

```

lemma vars-of-poly-in-alt-def2:
  ⟨vars-of-poly-in xs V ⟷ fold (λ(x, -) b. b ∧ vars-of-monom-in x V) xs True⟩
  ⟨proof⟩

```

```

sempref-definition vars-of-poly-in-impl
  is ⟨uncurry (RETURN oo vars-of-poly-in)⟩
  :: ⟨(poly-assn)k *a vars-assnk →a bool-assn⟩
  ⟨proof⟩

```

```

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* :: $\langle \text{l-list-polynomial} \Rightarrow \text{string set} \Rightarrow \text{string set} \rangle$ **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 $\langle \text{uncurry } (\text{RETURN } oo \text{ union-vars-monom}) \rangle$
 $\langle :: \langle \text{monom-assn}^k *_a \text{ vars-assn}^d \rightarrow_a \text{ vars-assn} \rangle$
 $\langle \text{proof} \rangle$

declare *union-vars-monom-impl.refine*[sempref-fr-rules]

sempref-definition *union-vars-poly-impl*
is $\langle \text{uncurry } (\text{RETURN } oo \text{ union-vars-poly}) \rangle$
 $\langle :: \langle \text{poly-assn}^k *_a \text{ vars-assn}^d \rightarrow_a \text{ vars-assn} \rangle$
 $\langle \text{proof} \rangle$

declare *union-vars-poly-impl.refine*[sempref-fr-rules]

hide-const (**open**) *Autoref-Fix-Rel.CONSTRAINT*

fun *status-assn* **where**
 $\langle \text{status-assn } - \text{ CSUCCESS } \text{ CSUCCESS} = \text{emp} \rangle \mid$
 $\langle \text{status-assn } - \text{ CFOUND } \text{ CFOUND} = \text{emp} \rangle \mid$
 $\langle \text{status-assn } R \text{ (CFAILED } a) \text{ (CFAILED } b) = R \text{ } a \text{ } b \rangle \mid$
 $\langle \text{status-assn } - - = \text{false} \rangle$

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*[*sepref-fr-rules*]:
 $\langle \text{CONSTRAINT } \textit{is-pure } R \implies$
 $(\text{uncurry } (\text{return } \textit{oo } \textit{merge-cstatus}), \text{uncurry } (\text{RETURN } \textit{oo } \textit{merge-cstatus})) \in$
 $(\text{status-assn } R)^k *_{\alpha} (\text{status-assn } R)^k \rightarrow_{\alpha} \text{status-assn } R \rangle$
 $\langle \textit{proof} \rangle$

sepref-definition *add-poly-impl*
is $\langle \textit{add-poly-l} \rangle$
 $:: \langle (\text{poly-assn } \times_{\alpha} \text{poly-assn})^k \rightarrow_{\alpha} \text{poly-assn} \rangle$
 $\langle \textit{proof} \rangle$

declare *add-poly-impl.refine*[*sepref-fr-rules*]

sepref-register *mult-monomials*

lemma *mult-monomials-alt-def*:
 $\langle (\text{RETURN } \textit{oo } \textit{mult-monomials}) \ x \ y = \text{REC}_T$
 $(\lambda f \ (p, q).$
 $\textit{case } (p, q) \ \textit{of}$
 $\quad (\ [], -) \Rightarrow \text{RETURN } q$
 $\quad | \ (-, []) \Rightarrow \text{RETURN } p$
 $\quad | \ (x \# p, y \# q) \Rightarrow$
 $\quad (\textit{if } x = y \ \textit{then } \textit{do } \{$
 $\quad \quad pq \leftarrow f \ (p, q);$
 $\quad \quad \text{RETURN } (x \# pq)\}$
 $\quad \textit{else } \textit{if } (x, y) \in \textit{var-order-rel}$
 $\quad \textit{then } \textit{do } \{$
 $\quad \quad pq \leftarrow f \ (p, y \# q);$
 $\quad \quad \text{RETURN } (x \# pq)\}$
 $\quad \textit{else } \textit{do } \{$
 $\quad \quad pq \leftarrow f \ (x \# p, q);$
 $\quad \quad \text{RETURN } (y \# pq)\}\})$
 $\ (x, y) \rangle$
 $\langle \textit{proof} \rangle$

sepref-definition *mult-monomials-impl*
is $\langle \text{uncurry } (\text{RETURN } \textit{oo } \textit{mult-monomials}) \rangle$
 $:: \langle (\text{monom-assn})^k *_{\alpha} (\text{monom-assn})^k \rightarrow_{\alpha} (\text{monom-assn}) \rangle$
 $\langle \textit{proof} \rangle$

declare *mult-monomials-impl.refine*[*sepref-fr-rules*]

sepref-definition *mult-monomials-impl*
is $\langle \text{uncurry } (\text{RETURN } \textit{oo } \textit{mult-monomials}) \rangle$
 $:: \langle (\text{monomial-assn})^k *_{\alpha} (\text{monomial-assn})^k \rightarrow_{\alpha} (\text{monomial-assn}) \rangle$
 $\langle \textit{proof} \rangle$

lemma *map-append-alt-def2*:
 $\langle (\text{RETURN } \textit{o } (\text{map-append } f \ b)) \ xs = \text{REC}_T$
 $(\lambda g \ xs. \ \textit{case } xs \ \textit{of } [] \Rightarrow \text{RETURN } b$
 $\quad | \ x \# xs \Rightarrow \textit{do } \{$
 $\quad \quad y \leftarrow g \ xs;$
 $\quad \quad \text{RETURN } (x \# y)\}$
 $\ \rangle$

$RETURN (f x \# y)$
 $\rangle xs\rangle$
 $\langle proof\rangle$

definition *map-append-poly-mult* **where**
 $\langle map-append-poly-mult x = map-append (mult-monomials x)\rangle$

declare *mult-monomials-impl.refine*[*sepref-fr-rules*]

sepref-definition *map-append-poly-mult-impl*
is $\langle uncurry2 (RETURN \ ooo \ map-append-poly-mult)\rangle$
 $:: \langle monomial-assn^k *_a poly-assn^k *_a poly-assn^k \rightarrow_a poly-assn\rangle$
 $\langle proof\rangle$

declare *map-append-poly-mult-impl.refine*[*sepref-fr-rules*]

TODO *foldl* $(\lambda l x. l @ [?f x]) [] ?l = map ?f ?l$ is the worst possible implementation of *map!*

sepref-definition *mult-poly-raw-impl*
is $\langle uncurry (RETURN \ oo \ mult-poly-raw)\rangle$
 $:: \langle poly-assn^k *_a poly-assn^k \rightarrow_a poly-assn\rangle$
 $\langle proof\rangle$

declare *mult-poly-raw-impl.refine*[*sepref-fr-rules*]

sepref-definition *mult-poly-impl*
is $\langle uncurry \ mult-poly-full\rangle$
 $:: \langle poly-assn^k *_a poly-assn^k \rightarrow_a poly-assn\rangle$
 $\langle proof\rangle$

declare *mult-poly-impl.refine*[*sepref-fr-rules*]

lemma *inverse-monomial*:
 $\langle monom-rel^{-1} \times_r int-rel = (monom-rel \times_r int-rel)^{-1}\rangle$
 $\langle proof\rangle$

lemma *eq-poly-rel-eq*[*sepref-import-param*]:
 $\langle ((=), (=)) \in poly-rel \rightarrow poly-rel \rightarrow bool-rel\rangle$
 $\langle proof\rangle$

sepref-definition *weak-equality-l-impl*
is $\langle uncurry \ weak-equality-l\rangle$
 $:: \langle poly-assn^k *_a poly-assn^k \rightarrow_a bool-assn\rangle$
 $\langle proof\rangle$

declare *weak-equality-l-impl.refine*[*sepref-fr-rules*]

sepref-register *add-poly-l* *mult-poly-full*

abbreviation *raw-string-assn* $:: \langle string \Rightarrow string \Rightarrow assn\rangle$ **where**
 $\langle raw-string-assn \equiv list-assn \ id-assn\rangle$

definition *show-nat* $:: \langle nat \Rightarrow string\rangle$ **where**
 $\langle show-nat i = show i\rangle$

lemma [sepref-import-param]:
 ⟨(show-nat, show-nat) ∈ nat-rel → ⟨Id⟩list-rel⟩
 ⟨proof⟩

lemma status-assn-pure-conv:
 ⟨status-assn (id-assn) a b = id-assn a b⟩
 ⟨proof⟩

lemma [sepref-fr-rules]:
 ⟨(uncurry3 (λx y. return oo (error-msg-not-equal-dom x y)), uncurry3 check-not-equal-dom-err) ∈
 poly-assn^k *_a poly-assn^k *_a poly-assn^k *_a poly-assn^k →_a raw-string-assn⟩
 ⟨proof⟩

lemma [sepref-fr-rules]:
 ⟨(return o (error-msg-notin-dom o nat-of-uint64), RETURN o error-msg-notin-dom)
 ∈ uint64-nat-assn^k →_a raw-string-assn⟩
 ⟨(return o (error-msg-reused-dom o nat-of-uint64), RETURN o error-msg-reused-dom)
 ∈ uint64-nat-assn^k →_a raw-string-assn⟩
 ⟨(uncurry (return oo (λi. error-msg (nat-of-uint64 i))), uncurry (RETURN oo error-msg))
 ∈ uint64-nat-assn^k *_a raw-string-assn^k →_a status-assn raw-string-assn⟩
 ⟨(uncurry (return oo error-msg), uncurry (RETURN oo error-msg))
 ∈ nat-assn^k *_a raw-string-assn^k →_a status-assn raw-string-assn⟩
 ⟨proof⟩

sepref-definition check-addition-l-impl
is ⟨uncurry6 check-addition-l⟩
 :: ⟨poly-assn^k *_a polys-assn^k *_a vars-assn^k *_a uint64-nat-assn^k *_a uint64-nat-assn^k *_a
 uint64-nat-assn^k *_a poly-assn^k →_a status-assn raw-string-assn⟩
 ⟨proof⟩

declare check-addition-l-impl.refine[sepref-fr-rules]

sepref-register check-mult-l-dom-err

definition check-mult-l-dom-err-impl **where**
 ⟨check-mult-l-dom-err-impl pd p ia i =
 (if pd then "The polynomial with id " @ show (nat-of-uint64 p) @ " was not found" else "'') @
 (if ia then "The id of the resulting id " @ show (nat-of-uint64 i) @ " was already given" else "'')⟩

definition check-mult-l-mult-err-impl **where**
 ⟨check-mult-l-mult-err-impl p q pq r =
 "Multiplying " @ show p @ " by " @ show q @ " gives " @ show pq @ " and not " @ show r⟩

lemma [sepref-fr-rules]:
 ⟨(uncurry3 ((λx y. return oo (check-mult-l-dom-err-impl x y))),
 uncurry3 (check-mult-l-dom-err)) ∈ bool-assn^k *_a uint64-nat-assn^k *_a bool-assn^k *_a uint64-nat-assn^k
 →_a raw-string-assn⟩
 ⟨proof⟩

lemma [sepref-fr-rules]:
 ⟨(uncurry3 ((λx y. return oo (check-mult-l-mult-err-impl x y))),
 uncurry3 (check-mult-l-mult-err)) ∈ poly-assn^k *_a poly-assn^k *_a poly-assn^k *_a poly-assn^k →_a raw-string-assn⟩

⟨proof⟩

sempref-definition *check-mult-l-impl*

is ⟨*uncurry6 check-mult-l*⟩
:: ⟨*poly-assn^k *_a polys-assn^k *_a vars-assn^k *_a uint64-nat-assn^k *_a poly-assn^k *_a uint64-nat-assn^k *_a*
poly-assn^k →_a status-assn raw-string-assn⟩
⟨proof⟩

declare *check-mult-l-impl.refine[sempref-fr-rules]*

definition *check-ext-l-dom-err-impl* :: ⟨*uint64 ⇒ ->*⟩ **where**

⟨*check-ext-l-dom-err-impl p =*
"There is already a polynomial with index " @ show (*nat-of-uint64 p*)⟩

lemma [*sempref-fr-rules*]:

⟨(((*return o (check-ext-l-dom-err-impl)*))),
(*check-extension-l-dom-err*) ∈ *uint64-nat-assn^k →_a raw-string-assn*⟩
⟨proof⟩

definition *check-extension-l-no-new-var-err-impl* :: ⟨*- ⇒ ->*⟩ **where**

⟨*check-extension-l-no-new-var-err-impl p =*
"No new variable could be found in polynomial " @ show *p*⟩

lemma [*sempref-fr-rules*]:

⟨(((*return o (check-extension-l-no-new-var-err-impl)*))),
(*check-extension-l-no-new-var-err*) ∈ *poly-assn^k →_a raw-string-assn*⟩
⟨proof⟩

definition *check-extension-l-side-cond-err-impl* :: ⟨*- ⇒ ->*⟩ **where**

⟨*check-extension-l-side-cond-err-impl v p r s =*
"Error while checking side conditions of extensions polynow, var is " @ show *v* @
" polynomial is " @ show *p* @ "side condition *p*p - p =* " @ show *s* @ " and should be 0"⟩

lemma [*sempref-fr-rules*]:

⟨(((*uncurry3 (λx y. return oo (check-extension-l-side-cond-err-impl x y))*))),
uncurry3 (check-extension-l-side-cond-err) ∈ *string-assn^k *_a poly-assn^k *_a poly-assn^k *_a poly-assn^k*
→_a raw-string-assn⟩
⟨proof⟩

definition *check-extension-l-new-var-multiple-err-impl* :: ⟨*- ⇒ ->*⟩ **where**

⟨*check-extension-l-new-var-multiple-err-impl v p =*
"Error while checking side conditions of extensions polynow, var is " @ show *v* @
" but it either appears at least once in the polynomial or another new variable is created " @
show *p* @ " but should not."⟩

lemma [*sempref-fr-rules*]:

⟨(((*uncurry (return oo (check-extension-l-new-var-multiple-err-impl))*))),
uncurry (check-extension-l-new-var-multiple-err) ∈ *string-assn^k *_a poly-assn^k →_a raw-string-assn*⟩
⟨proof⟩

sempref-register *check-extension-l-dom-err fmllookup'*

check-extension-l-side-cond-err check-extension-l-no-new-var-err
check-extension-l-new-var-multiple-err

definition *uminus-poly* :: $\langle \text{l-list-polynomial} \Rightarrow \text{l-list-polynomial} \rangle$ **where**
 $\langle \text{uminus-poly } p' = \text{map } (\lambda(a, b). (a, - b)) p' \rangle$

sepref-register *uminus-poly*

lemma [*sepref-import-param*]:

$\langle (\text{map } (\lambda(a, b). (a, - b)), \text{uminus-poly}) \in \text{poly-rel} \rightarrow \text{poly-rel} \rangle$
 $\langle \text{proof} \rangle$

sepref-register *vars-of-poly-in*
weak-equality-l

lemma [*safe-constraint-rules*]:

$\langle \text{Sepref-Constraints.CONSTRAINT single-valued (the-pure monomial-assn)} \rangle$ **and**
single-valued-the-monomial-assn:
 $\langle \text{single-valued (the-pure monomial-assn)} \rangle$
 $\langle \text{single-valued ((the-pure monomial-assn)}^{-1}) \rangle$
 $\langle \text{proof} \rangle$

sepref-definition *check-extension-l-impl*

is $\langle \text{uncurry5 check-extension-l} \rangle$

:: $\langle \text{poly-assn}^k *_a \text{polys-assn}^k *_a \text{vars-assn}^k *_a \text{uint64-nat-assn}^k *_a \text{string-assn}^k *_a \text{poly-assn}^k \rightarrow_a$
status-assn raw-string-assn
 $\langle \text{proof} \rangle$

declare *check-extension-l-impl.refine*[*sepref-fr-rules*]

sepref-definition *check-del-l-impl*

is $\langle \text{uncurry2 check-del-l} \rangle$

:: $\langle \text{poly-assn}^k *_a \text{polys-assn}^k *_a \text{uint64-nat-assn}^k \rightarrow_a \text{status-assn raw-string-assn} \rangle$
 $\langle \text{proof} \rangle$

lemmas [*sepref-fr-rules*] = *check-del-l-impl.refine*

abbreviation *pac-step-rel* **where**

$\langle \text{pac-step-rel} \equiv \text{p2rel } (\langle \text{Id}, \langle \text{monomial-rel} \rangle \text{list-rel}, \text{Id} \rangle \text{pac-step-rel-raw}) \rangle$

sepref-register *PAC-Polynomials-Operations.normalize-poly*

pac-src1 pac-src2 new-id pac-mult case-pac-step check-mult-l
check-addition-l check-del-l check-extension-l

lemma *pac-step-rel-assn-alt-def2*:

$\langle \text{hn-ctxt } (\text{pac-step-rel-assn nat-assn poly-assn id-assn}) b \text{ bi} =$
 hn-val
 $(\text{p2rel}$
 $((\text{nat-rel}, \text{poly-rel}, \text{Id} :: (\text{string} \times -) \text{set}) \text{pac-step-rel-raw})) b \text{ bi} \rangle$
 $\langle \text{proof} \rangle$

lemma *is-AddD-import*[*sepref-fr-rules*]:

assumes $\langle \text{CONSTRAINT is-pure } K \rangle$ $\langle \text{CONSTRAINT is-pure } V \rangle$

shows

$\langle (\text{return } o \text{ pac-res}, \text{RETURN } o \text{ pac-res}) \in [\lambda x. \text{is-Add } x \vee \text{is-Mult } x \vee \text{is-Extension } x]_a$
 $(\text{pac-step-rel-assn } K V R)^k \rightarrow V \rangle$

$\langle (\text{return } o \text{ pac-src1}, \text{RETURN } o \text{ pac-src1}) \in [\lambda x. \text{is-Add } x \vee \text{is-Mult } x \vee \text{is-Del } x]_a (\text{pac-step-rel-assn } K \ V \ R)^k \rightarrow K \rangle$
 $\langle (\text{return } o \text{ new-id}, \text{RETURN } o \text{ new-id}) \in [\lambda x. \text{is-Add } x \vee \text{is-Mult } x \vee \text{is-Extension } x]_a (\text{pac-step-rel-assn } K \ V \ R)^k \rightarrow K \rangle$
 $\langle (\text{return } o \text{ is-Add}, \text{RETURN } o \text{ is-Add}) \in (\text{pac-step-rel-assn } K \ V \ R)^k \rightarrow_a \text{bool-assn} \rangle$
 $\langle (\text{return } o \text{ is-Mult}, \text{RETURN } o \text{ is-Mult}) \in (\text{pac-step-rel-assn } K \ V \ R)^k \rightarrow_a \text{bool-assn} \rangle$
 $\langle (\text{return } o \text{ is-Del}, \text{RETURN } o \text{ is-Del}) \in (\text{pac-step-rel-assn } K \ V \ R)^k \rightarrow_a \text{bool-assn} \rangle$
 $\langle (\text{return } o \text{ is-Extension}, \text{RETURN } o \text{ is-Extension}) \in (\text{pac-step-rel-assn } K \ V \ R)^k \rightarrow_a \text{bool-assn} \rangle$
 $\langle \text{proof} \rangle$

lemma [*sepref-fr-rules*]:

$\langle \text{CONSTRAINT is-pure } K \implies$
 $(\text{return } o \text{ pac-src2}, \text{RETURN } o \text{ pac-src2}) \in [\lambda x. \text{is-Add } x]_a (\text{pac-step-rel-assn } K \ V \ R)^k \rightarrow K \rangle$
 $\langle \text{CONSTRAINT is-pure } V \implies$
 $(\text{return } o \text{ pac-mult}, \text{RETURN } o \text{ pac-mult}) \in [\lambda x. \text{is-Mult } x]_a (\text{pac-step-rel-assn } K \ V \ R)^k \rightarrow V \rangle$
 $\langle \text{CONSTRAINT is-pure } R \implies$
 $(\text{return } o \text{ new-var}, \text{RETURN } o \text{ new-var}) \in [\lambda x. \text{is-Extension } x]_a (\text{pac-step-rel-assn } K \ V \ R)^k \rightarrow R \rangle$
 $\langle \text{proof} \rangle$

lemma *is-Mult-lastI*:

$\langle \neg \text{is-Add } b \implies \neg \text{is-Mult } b \implies \neg \text{is-Extension } b \implies \text{is-Del } b \rangle$
 $\langle \text{proof} \rangle$

sepref-register *is-cfailed is-Del*

definition *PAC-checker-l-step'* :: - **where**

$\langle \text{PAC-checker-l-step}' a \ b \ c \ d = \text{PAC-checker-l-step } a \ (b, c, d) \rangle$

lemma *PAC-checker-l-step-alt-def*:

$\langle \text{PAC-checker-l-step } a \ b \ c \ d \ e = (\text{let } (b,c,d) = bcd \text{ in } \text{PAC-checker-l-step}' a \ b \ c \ d \ e) \rangle$
 $\langle \text{proof} \rangle$

sepref-decl-intf ('k) *acode-status is* ('k) *code-status*

sepref-decl-intf ('k, 'b, 'lbl) *apac-step is* ('k, 'b, 'lbl) *pac-step*

sepref-register *merge-cstatus full-normalize-poly new-var is-Add*

lemma *poly-rel-the-pure*:

$\langle \text{poly-rel} = \text{the-pure poly-assn} \rangle$ **and**
 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**
 $\text{single-valued-uint64-nat-rel}[\text{safe-constraint-rules}]$:
 $\langle \text{CONSTRAINT single-valued uint64-nat-rel} \rangle$
 $\langle \text{proof} \rangle$

sepref-definition *check-step-impl*

is $\langle \text{uncurry4 PAC-checker-l-step}' \rangle$
 $:: \langle \text{poly-assn}^k *_a (\text{status-assn raw-string-assn})^d *_a \text{vars-assn}^d *_a \text{polys-assn}^d *_a (\text{pac-step-rel-assn } (\text{uint64-nat-assn}) \text{ poly-assn } (\text{string-assn} :: \text{string} \implies -))^d \rightarrow_a \text{status-assn raw-string-assn} \times_a \text{vars-assn} \times_a \text{polys-assn} \rangle$

⟨proof⟩

declare *check-step-impl.refine*[sepref-fr-rules]

sepref-register *PAC-checker-l-step PAC-checker-l-step' fully-normalize-poly-impl*

definition *PAC-checker-l'* **where**

⟨*PAC-checker-l' p* \mathcal{V} *A* *status steps* = *PAC-checker-l p* (\mathcal{V} , *A*) *status steps*⟩

lemma *PAC-checker-l-alt-def*:

⟨*PAC-checker-l p* \mathcal{V} *A* *status steps* =
(*let* (\mathcal{V} , *A*) = \mathcal{V} *A* *in* *PAC-checker-l' p* \mathcal{V} *A* *status steps*)⟩
⟨proof⟩

sepref-definition *PAC-checker-l-impl*

is ⟨*uncurry4* *PAC-checker-l'*⟩
:: ⟨*poly-assn*^{*k*} *_{*a*} *vars-assn*^{*d*} *_{*a*} *polys-assn*^{*d*} *_{*a*} (*status-assn raw-string-assn*)^{*d*} *_{*a*}
(*list-assn* (*pac-step-rel-assn* (*uint64-nat-assn*) *poly-assn string-assn*))^{*k*} →_{*a*}
status-assn raw-string-assn ×_{*a*} *vars-assn* ×_{*a*} *polys-assn*⟩
⟨proof⟩

declare *PAC-checker-l-impl.refine*[sepref-fr-rules]

abbreviation *polys-assn-input* **where**

⟨*polys-assn-input* ≡ *iam-fmap-assn nat-assn poly-assn*⟩

definition *remap-polys-l-dom-err-impl* :: ⟨-⟩ **where**

⟨*remap-polys-l-dom-err-impl* =
"Error during initialisation. Too many polynomials where provided. If this happens," @
"please report the example to the authors, because something went wrong during " @
"code generation (code generation to arrays is likely to be broken)."⟩

lemma [*sepref-fr-rules*]:

⟨((*uncurry0* (*return* (*remap-polys-l-dom-err-impl*))),
uncurry0 (*remap-polys-l-dom-err*)) ∈ *unit-assn*^{*k*} →_{*a*} *raw-string-assn*⟩
⟨proof⟩

MLton is not able to optimise the calls to pow.

lemma *pow-2-64*: ⟨(2::nat) ^ 64 = 18446744073709551616⟩

⟨proof⟩

sepref-register *upper-bound-on-dom op-fmap-empty*

sepref-definition *remap-polys-l-impl*

is ⟨*uncurry2* *remap-polys-l2*⟩
:: ⟨*poly-assn*^{*k*} *_{*a*} *vars-assn*^{*d*} *_{*a*} *polys-assn-input*^{*d*} →_{*a*}
status-assn raw-string-assn ×_{*a*} *vars-assn* ×_{*a*} *polys-assn*⟩
⟨proof⟩

lemma *remap-polys-l2-remap-polys-l*:

⟨(*uncurry2* *remap-polys-l2*, *uncurry2* *remap-polys-l*) ∈ (*Id* ×_{*r*} ⟨*Id*⟩*set-rel*) ×_{*r*} *Id* →_{*f*} ⟨*Id*⟩*nres-rel*⟩
⟨proof⟩

lemma [*sepref-fr-rules*]:

\langle (*uncurry2* *remap-polys-l-impl*,
uncurry2 *remap-polys-l*) \in *poly-assn*^k *_a *vars-assn*^d *_a *polys-assn-input*^d \rightarrow_a
status-assn *raw-string-assn* \times_a *vars-assn* \times_a *polys-assn* \rangle
 \langle *proof* \rangle

sepref-register *remap-polys-l*

sepref-definition *full-checker-l-impl*

is \langle *uncurry2* *full-checker-l* \rangle
:: \langle *poly-assn*^k *_a *polys-assn-input*^d *_a (*list-assn* (*pac-step-rel-assn* (*uint64-nat-assn*) *poly-assn* *string-assn*))^k
 \rightarrow_a
status-assn *raw-string-assn* \times_a *vars-assn* \times_a *polys-assn* \rangle
 \langle *proof* \rangle

sepref-definition *PAC-update-impl*

is \langle *uncurry2* (*RETURN* *ooo* *fmupd*) \rangle
:: \langle *nat-assn*^k *_a *poly-assn*^k *_a (*polys-assn-input*)^d \rightarrow_a *polys-assn-input* \rangle
 \langle *proof* \rangle

sepref-definition *PAC-empty-impl*

is \langle *uncurry0* (*RETURN* *fmempty*) \rangle
:: \langle *unit-assn*^k \rightarrow_a *polys-assn-input* \rangle
 \langle *proof* \rangle

sepref-definition *empty-vars-impl*

is \langle *uncurry0* (*RETURN* $\{\}$) \rangle
:: \langle *unit-assn*^k \rightarrow_a *vars-assn* \rangle
 \langle *proof* \rangle

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-asciis-of-literal* **::** \langle - \rangle **where**

\langle *unsafe-asciis-of-literal* *xs* = *String.asciis-of-literal* *xs* \rangle

definition *unsafe-asciis-of-literal'* **::** \langle - \rangle **where**

\langle *simp*, *symmetric*, *code* \rangle : \langle *unsafe-asciis-of-literal'* = *unsafe-asciis-of-literal* \rangle

code-printing

constant *unsafe-asciis-of-literal'* \rightarrow
(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**

\langle *simp* \rangle : \langle *raw-explode* = *String.explode* \rangle

code-printing

constant *raw-explode* \rightarrow
(SML) *String.explode*

definition \langle *hashcode-literal'* *s* \equiv

foldl ($\lambda h x. h * 33 + \text{uint32-of-int (of-char } x)$) 5381
(raw-explode s)

lemmas [*code*] =
hashcode-literal-def[unfolding String.explode-code
unsafe-ascii-of-literal-def[symmetric]]

definition *uint32-of-char* **where**
[symmetric, code-unfold]: $\langle \text{uint32-of-char } x = \text{uint32-of-int (int-of-char } x) \rangle$

code-printing

constant *uint32-of-char* \rightarrow
(SML) !(Word32.fromInt /o (Char.ord))

lemma [*code*]: $\langle \text{hashcode } s = \text{hashcode-literal' } s \rangle$
<proof>

We compile Pastèque in `PAC_Checker_MLton.thy`.

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*

14 Correctness theorem

context *poly-embed*
begin

definition *full-poly-assn* **where**
 $\langle \text{full-poly-assn} = \text{hr-comp poly-assn (fully-unsorted-poly-rel } O \text{ mset-poly-rel)} \rangle$

definition *full-poly-input-assn* **where**
 $\langle \text{full-poly-input-assn} = \text{hr-comp}$
 $(\text{hr-comp polys-assn-input}$
 $((\text{nat-rel, fully-unsorted-poly-rel } O \text{ mset-poly-rel})\text{fmap-rel}))$
 $\text{polys-rel} \rangle$

definition *fully-pac-assn* **where**
 $\langle \text{fully-pac-assn} = (\text{list-assn}$
 $(\text{hr-comp (pac-step-rel-assn uint64-nat-assn poly-assn string-assn)}$
 p2rel
 $(\langle \text{nat-rel,}$
 $\text{fully-unsorted-poly-rel } O$
 $\text{mset-poly-rel, var-rel} \rangle \text{pac-step-rel-raw})) \rangle$

definition *code-status-assn* **where**
 $\langle \text{code-status-assn} = \text{hr-comp (status-assn raw-string-assn)}$
 $\text{code-status-status-rel} \rangle$

definition *full-vars-assn* **where**
 $\langle \text{full-vars-assn} = \text{hr-comp (hs.assn string-assn)}$
 $(\langle \text{var-rel} \rangle \text{set-rel}) \rangle$

lemma *polys-rel-full-polys-rel*:
 $\langle polys-rel-full = Id \times_r polys-rel \rangle$
 $\langle proof \rangle$

definition *full-polys-assn* :: $\langle \rightarrow \rangle$ **where**
 $\langle full-polys-assn = hr-comp (hr-comp polys-assn$
 $\quad ((nat-rel,$
 $\quad \quad sorted-poly-rel \ O \ mset-poly-rel) fmap-rel))$
 $\quad polys-rel \rangle$

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 polynomial represented as a list
5. the input polynomials as hash map (as an array of option polynomial)
6. a representation of the PAC proofs.

lemma *PAC-full-correctness*:
 $\langle (uncurry2 \ full-checker-l-impl,$
 $\quad uncurry2 (\lambda spec \ A \ -. \ PAC-checker-specification \ spec \ A))$
 $\in (full-poly-assn)^k *_{\alpha} (full-poly-input-assn)^d *_{\alpha} (fully-pac-assn)^k \rightarrow_{\alpha} hr-comp$
 $(code-status-assn \times_{\alpha} full-vars-assn \times_{\alpha} hr-comp \ polys-assn$
 $\quad ((nat-rel, \ sorted-poly-rel \ O \ mset-poly-rel) fmap-rel))$
 $\{((st, \ G), \ st', \ G') .$
 $\quad st = st' \wedge (st \neq FAILED \longrightarrow (G, \ G') \in Id \times_r \ polys-rel)\}$
 $\langle proof \rangle$

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 $(\text{uncurry2 } \text{full-checker-l-impl}, \text{uncurry2 } (\lambda \text{spec } A -. \text{PAC-checker-specification spec } A)) \in \text{PAC.full-poly-assn}^k *_a \text{PAC.full-poly-input-assn}^d *_a \text{PAC.fully-pac-assn}^k \rightarrow_a \text{hr-comp}$
 $(\text{PAC.code-status-assn} \times_a \text{PAC.full-vars-assn} \times_a \text{hr-comp polys-assn } (\langle \text{nat-rel}, \text{sorted-poly-rel}$
 $O \text{PAC.mset-poly-rel} \rangle \text{fmap-rel})) \{((st, G), st', G'). st = st' \wedge (st \neq \text{FAILED} \longrightarrow (G, G') \in$
 $\text{Id} \times_r \text{polys-rel})\}$.

end

theory *PAC-Checker-MLton*
imports *PAC-Checker-Synthesis*
begin

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*

Here is how to compile it:

compile-generated-files -
external-files

$\langle \text{code}/\text{parser.sml} \rangle$
 $\langle \text{code}/\text{pasteque.sml} \rangle$
 $\langle \text{code}/\text{pasteque.mlb} \rangle$

where $\langle \text{fn } \text{dir} \Rightarrow$

let

val exec = Generated-Files.execute (dir + Path.basic code);

val - =

exec $\langle \text{Compilation} \rangle$

$(\text{verbatim } \langle \text{\$ISABELLE-MLTON } \text{\$ISABELLE-MLTON-OPTIONS } \rangle \wedge$

$\text{--const 'MLton.safe false' --verbose 1 --default-type int64 --output pasteque } \wedge$

$\text{--codegen native --inline 700 --cc-opt -O3 pasteque.mlb});$

in () end

end

Acknowledgment

This work is supported by Austrian Science Fund (FWF), NFN S11408-N23 (RiSE), and LIT AI Lab funded by the State of Upper Austria.

References

- [1] D. Kaufmann, M. Fleury, and A. Biere. The proof checkers pacheck and pasteque for the practical algebraic calculus. In O. Strichman and A. Ivrii, editors, *Formal Methods in Computer-Aided Design, FMCAD 2020, September 21-24, 2020*. IEEE, 2020.