PAC Checker

Mathias Fleury and Daniela Kaufmann

February 10, 2021

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 Checker Algorithm **2**
- 1.1 Algorithm . . . 2
- 1.2 Full Checker . . . 9

2 Executable Checker **11**
- 2.1 Definitions . . . 11
- 2.2 Correctness . . . 15

3 Initial Normalisation of Polynomials **46**
- 3.1 Sorting . . . 46

4 Code Synthesis of the Complete Checker **48**

5 Perfectly sharing of elements **54**
- 5.1 Definition . . . 54

6 Refinement **55**

7 Correctness theorem **141**

theory *EPAC-Specification*
imports *PAC-Checker.PAC-More-Poly*
PAC-Checker.PAC-Specification
begin

end

theory *EPAC-Checker-Specification*
imports *EPAC-Specification*
Refine-Imperative-HOL.IICF
PAC-Checker.Finite-Map-Multiset

PAC-Checker.PAC-Checker-Specification
begin

1 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.

1.1 Algorithm

datatype (*'a*, *'b*, *'lbls*) *pac-step* =
  CL (*pac-srcs*: ‹(*'a* × *'lbls*) *list*›) (*new-id*: *'lbls*) (*pac-res*: *'a*) |
  Extension (*new-id*: *'lbls*) (*new-var*: *'b*) (*pac-res*: *'a*) |
  Del (*pac-src1*: *'lbls*)

definition *check-linear-comb* :: ‹(*nat*, *int mpoly*) *fmap* ⇒ *nat set* ⇒ (*int mpoly* × *nat*) *list* ⇒ *nat* ⇒ *int mpoly* ⇒ *bool nres*› **where**
  ‹*check-linear-comb* $\mathcal{A}$ $\mathcal{V}$ *xs n r* = *SPEC*(λ*b*. *b* ⟶ (∀ *i* ∈ *set xs*. *snd i* ∈# *dom-m* $\mathcal{A}$ ∧ *vars* (*fst i*) ⊆ $\mathcal{V}$) ∧ *n* ∉# *dom-m* $\mathcal{A}$ ∧
  vars r ⊆ $\mathcal{V}$ ∧ *xs* ≠ [] ∧ (∑ (*p*,*n*) ∈#*mset xs*. *the* (*fmlookup* $\mathcal{A}$ *n*) ∗ *p*) − *r* ∈ *ideal polynomial-bool*)›

lemma *PAC-Format-LC*:
  assumes
    i: ‹((𝒱, *A*), $\mathcal{V}_B$, *B*) ∈ *polys-rel-full*› **and**
    st: ‹*PAC-Format*** ($\mathcal{V}_0$, A_0) (𝒱, *B*)› **and**
    vars: ‹∀ *i*∈#*x11*. *snd i* ∈# *dom-m A* ∧ *vars* (*fst i*) ⊆ 𝒱› **and**
    AV: ‹⋃ (*vars* ‘ *set-mset* (*ran-m A*)) ⊆ 𝒱› **and**
    fin: ‹*x11* ≠ {#}› **and**
   r: ‹(∑ *x*∈#*x11*. *case x of* (*p*, *n*) ⇒ *the* (*fmlookup A n*) ∗ *p*) − *r* ∈ *More-Modules.ideal polynomial-bool*›
    ‹*vars r* ⊆ 𝒱›
  shows ‹*PAC-Format*** (𝒱, *B*) (𝒱, *add-mset r B*)›
proof −
  have *AB*: ‹*i* ∈# *dom-m A* ⟹ *the* (*fmlookup A i*) ∈# *B*› **and** *BA*: ‹*B* = *ran-m A*› **for** *i*
    using *i* **by** (*auto simp*: *polys-rel-full-def polys-rel-def*)
  have ‹*PAC-Format*** (𝒱, *B*) (𝒱, *add-mset* ((∑ *x*∈#*x11*. *case x of* (*p*, *n*) ⇒ *the* (*fmlookup A n*) ∗ *p*)) *B*)›
    using *fin vars*
  proof (*induction x11*)
    case *empty*
    then show *?case* **by** *auto*
  next
    case (*add x F*)
    then have *IH*: ‹*F* ≠ {#} ⟹ *PAC-Format*** (𝒱, *B*) (𝒱, *add-mset* (∑ (*p*,*n*)∈#*F*. *the* (*fmlookup A n*) ∗ *p*) *B*)› **and**
      x-A: ‹*snd x* ∈# *dom-m A*› **and**
      x-var: ‹*vars* (*fst x*) ⊆ 𝒱› **and**
      x-in: ‹*the* (*fmlookup A* (*snd x*)) ∈# *B*›
      using *AB*[*of* ‹*snd x*›] **by** *auto*
    have *vars-A*: ‹*vars* (*the* (*fmlookup A* (*snd x*))) ⊆ 𝒱›
      using *AV x-A*
      by (*auto simp*: *ran-m-def*)
    let *?B* = ‹(*add-mset* (∑ (*p*,*n*)∈#*F*. *the* (*fmlookup A n*) ∗ *p*) *B*)›
    let *?p* = ‹(∑ (*p*,*n*)∈#*F*. *the* (*fmlookup A n*) ∗ *p*)›
    let *?q* = ‹(∑ (*p*,*n*)∈#{#*x*#}. *the* (*fmlookup A n*) ∗ *p*)›

let *?vars* = ‹λA. ⋃ (*vars* ‘ *set-mset* (A)) ⊆ 𝒱›
consider
(*empty*) ‹F = {#}› |
(*nempty*) ‹F ≠ {#}›
by *blast*
then show *?case*
proof *cases*
case *empty2*: *empty*
have ‹*PAC-Format* (𝒱, B) (𝒱, *add-mset* ((∑ x∈#{#x#}. *case* x *of* (p, n) ⇒ *the* (*fmlookup* A n) ∗ p)) B)›
apply (*cases* x)
apply (*rule PAC-Format.intros*(2)[*OF x-in*, *of* ‹*fst* x›])
by (*use x-var vars-A* **in** ‹*auto simp*: *ideal.span-zero elim*!: *vars-unE*›)
then show *?thesis*
using *empty2* **by** *auto*
next
case *nempty*
then have *IH*: ‹*PAC-Format*** (𝒱, B) (𝒱, *add-mset ?p* B)›
using *IH* **by** *auto*
from *rtranclp-PAC-Format-subset-ideal*[*OF this*] **have** *vars2*: ‹*?vars ?B*›
using *AV* **unfolding** *BA*[*symmetric*] **by** *auto*
have *1*:
‹*PAC-Format* (𝒱, *?B*) (𝒱, *add-mset* (*the* (*fmlookup* A (*snd* x)) ∗ *fst* x) *?B*)› (**is** ‹*PAC-Format* - (-, *?C*)›)
apply (*cases* x)
apply (*rule PAC-Format.intros*(2)[*of* ‹*the* (*fmlookup* A (*snd* x))› - ‹*fst* x›])
by (*use x-in x-var vars-A* **in** ‹*auto simp*: *ideal.span-zero elim*!: *vars-unE*›)
from *PAC-Format-subset-ideal*[*OF this*] **have** ‹*?vars* (*add-mset* (*the* (*fmlookup* A (*snd* x)) ∗ *fst* x) *?B*)›
using *vars2* **by** *auto*
have *2*: ‹*PAC-Format* (𝒱, *?C*) (𝒱, *add-mset* (∑ (p,n)∈#*add-mset* x F. *the* (*fmlookup* A n) ∗ p) *?C*)› (**is** ‹*PAC-Format* - (-, *?D*)›)
apply (*cases* x)
apply (*rule PAC-Format.intros*(*1*)[*of* ‹*?p*› - *?q*])
by (*use insert x-in x-var vars-A vars2* **in** ‹*auto simp*: *ideal.span-zero elim*!: *in-vars-addE vars-unE*›)
then have *3*: ‹*PAC-Format*** (𝒱, *?D*) (𝒱, *add-mset* (∑ (p,n)∈# *add-mset* x F. *the* (*fmlookup* A n) ∗ p) B)›
using *PAC-Format.del*[*of ?p ?D* 𝒱]
PAC-Format.del[*of* ‹*the* (*fmlookup* A (*snd* x)) ∗ *fst* x› ‹*remove1-mset ?p ?D*› 𝒱]
by (*auto 4 7*)
show *?thesis*
using *IH 1 2 3* **by** *auto*
qed
qed
moreover have ‹*PAC-Format* (𝒱, *add-mset* (∑ (p,n)∈#*x11*. *the* (*fmlookup* A n) ∗ p) B)
(𝒱, *add-mset* r (*add-mset* (∑ (p,n)∈#*x11*. *the* (*fmlookup* A n) ∗ p) B))› (**is** ‹*PAC-Format* - *?E*›)
by (*rule PAC-Format.intros*(*2*)[*of* ‹(∑ (p,n)∈#*x11*. *the* (*fmlookup* A n) ∗ p)› - *1*])
(*use* r **in** *auto*)
moreover have ‹*PAC-Format ?E* (𝒱, *add-mset* r B)›
using *PAC-Format.del*[*of* ‹(∑ (p,n)∈#*x11*. *the* (*fmlookup* A n) ∗ p)› ‹*snd ?E*› ‹*fst ?E*›]
by *auto*
ultimately show *?thesis*
using *st* **by** *auto*
qed

```
definition PAC-checker-step-inv where
  ‹PAC-checker-step-inv spec stat 𝒱 A ⟷
  (∀ i∈#dom-m A. vars (the (fmlookup A i)) ⊆ 𝒱) ∧
  vars spec ⊆ 𝒱›

definition check-extension-precalc
  :: ‹(nat, int mpoly) fmap ⇒ nat set ⇒ nat ⇒ nat ⇒ int mpoly ⇒ (bool) nres›
 where
  ‹check-extension-precalc A 𝒱 i v p' =
     SPEC(λb. b ⟶ (i ∉# dom-m A ∧
     (v ∉ 𝒱 ∧
           (p')² − (p') ∈ ideal polynomial-bool ∧
            vars (p') ⊆ 𝒱)))›

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
     CL - - - ⇒
  do {
        ASSERT(PAC-checker-step-inv spec stat 𝒱 A);
        r ← normalize-poly-spec (pac-res st);
       eq ← check-linear-comb A 𝒱 (pac-srcs 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, fmupd (new-id st) r A)
       else RETURN (FAILED, (𝒱, A))
   }
   | Del - ⇒
       do {
        ASSERT(PAC-checker-step-inv spec stat 𝒱 A);
        eq ← check-del A (pac-src1 st);
        if eq
        then RETURN (stat, (𝒱, fmdrop (pac-src1 st) A))
        else RETURN (FAILED, (𝒱, A))
   }
   | Extension - - - ⇒
       do {
         ASSERT(PAC-checker-step-inv spec stat 𝒱 A);
        r ← normalize-poly-spec (pac-res st);
        (eq) ← check-extension-precalc A 𝒱 (new-id st) (new-var st) r;
        if eq
        then do {
          r0 ← SPEC(λr0. r0  =  (r − Var (new-var st)) ∧
               vars r0 = vars (r) ∪ {new-var st});
         RETURN (stat,
          insert (new-var st) 𝒱, fmupd (new-id st) (r0) A)}
        else RETURN (FAILED, (𝒱, A))
   }
          )›

lemma PAC-checker-step-PAC-checker-specification2:
  fixes a :: ‹status›
```

```
  assumes AB: ⟨((𝒱, A),(𝒱_B, B)) ∈ polys-rel-full⟩ and
    ⟨¬is-failed a⟩ and
    [simp,intro]: ⟨a = FOUND ⟹ spec ∈ pac-ideal (set-mset A_0)⟩ and
    A_0B: ⟨PAC-Format** (𝒱_0, A_0) (𝒱, B)⟩ and
    spec_0: ⟨vars spec ⊆ 𝒱_0⟩  and
    vars-A_0: ⟨⋃ (vars ‘ set-mset A_0) ⊆ 𝒱_0⟩
  shows ⟨PAC-checker-step spec (a, (𝒱, A)) st ≤ ⇓ (status-rel ×_r polys-rel-full) (PAC-checker-specification-step2
(𝒱_0, A_0) spec (𝒱, B))⟩
proof -
  have
    ⟨𝒱_B = 𝒱⟩and
    [simp, intro]:⟨(A, B) ∈ polys-rel⟩
    using AB
    by (auto simp: polys-rel-full-def)
  have H0: ⟨2 * the (fmlookup A x12) - r ∈ More-Modules.ideal polynomial-bool ⟹
          r ∈ pac-ideal
                (insert (the (fmlookup A x12))
                  ((λx. the (fmlookup A x)) ‘ set-mset Aa))⟩ for x12 r Aa
    by (metis (no-types, lifting) ab-semigroup-mult-class.mult.commute
        diff-in-polynomial-bool-pac-idealI
      ideal.span-base pac-idealI3 set-image-mset set-mset-add-mset-insert union-single-eq-member)+
  then have H0': ⟨⋀Aa. 2 * the (fmlookup A x12) - r ∈ More-Modules.ideal polynomial-bool ⟹
          r - spec ∈ More-Modules.ideal polynomial-bool ⟹
          spec ∈ pac-ideal (insert (the (fmlookup A x12)) ((λx. the (fmlookup A x)) ‘ set-mset Aa))⟩
    for r x12
    by (metis (no-types, lifting) diff-in-polynomial-bool-pac-idealI)

  have H1: ⟨ x12 ∈# dom-m A ⟹
       2 * the (fmlookup A x12) - r ∈ More-Modules.ideal polynomial-bool ⟹
       r - spec ∈ More-Modules.ideal polynomial-bool ⟹
       vars spec ⊆ vars r ⟹
       spec ∈ pac-ideal (set-mset B)⟩ for x12 r
     using ⟨(A,B) ∈ polys-rel⟩
      ideal.span-add[OF ideal.span-add[OF ideal.span-neg ideal.span-neg,
         of ⟨the (fmlookup A x12)⟩ - ⟨the (fmlookup A x12)⟩],
      of ⟨set-mset B ∪ polynomial-bool⟩ ⟨2 * the (fmlookup A x12) - r⟩]
     unfolding polys-rel-def
     by (auto dest!: multi-member-split simp: ran-m-def
        intro: H0')
   have H2': ⟨the (fmlookup A x11) + the (fmlookup A x12) - r ∈ More-Modules.ideal polynomial-bool
⟹
       B = add-mset (the (fmlookup A x11)) {#the (fmlookup A x). x ∈# Aa#} ⟹
       (the (fmlookup A x11) + the (fmlookup A x12) - r
        ∈ More-Modules.ideal
            (insert (the (fmlookup A x11))
              ((λx. the (fmlookup A x)) ‘ set-mset Aa ∪ polynomial-bool)) ⟹
        - r
        ∈ More-Modules.ideal
            (insert (the (fmlookup A x11))
              ((λx. the (fmlookup A x)) ‘ set-mset Aa ∪ polynomial-bool))) ⟹
       r ∈ pac-ideal (insert (the (fmlookup A x11)) ((λx. the (fmlookup A x)) ‘ set-mset Aa))⟩
     for r x12 x11 A Aa
     by (metis (mono-tags, lifting) Un-insert-left diff-diff-eq2 diff-in-polynomial-bool-pac-idealI diff-zero
       ideal.span-diff ideal.span-neg minus-diff-eq pac-idealI1 pac-ideal-def set-image-mset
       set-mset-add-mset-insert union-single-eq-member)
```

have *H2*: ‹*x11* ∈# *dom-m A* ⟹
x12 ∈# *dom-m A* ⟹
the (*fmlookup A x11*) + *the* (*fmlookup A x12*) − *r*
∈ *More-Modules.ideal polynomial-bool* ⟹
r − *spec* ∈ *More-Modules.ideal polynomial-bool* ⟹
spec ∈ *pac-ideal* (*set-mset B*)› **for** *x12 r x11*
using ‹(*A*,*B*) ∈ *polys-rel*›
ideal.span-add[*OF ideal.span-add*[*OF ideal.span-neg ideal.span-neg*,
of ‹*the* (*fmlookup A x11*)› - ‹*the* (*fmlookup A x12*)›],
of ‹*set-mset B* ∪ *polynomial-bool*› ‹*the* (*fmlookup A x11*) + *the* (*fmlookup A x12*) − *r*›]
unfolding *polys-rel-def*
by (*subgoal-tac* ‹*r* ∈ *pac-ideal* (*set-mset B*)›)
(*auto dest*!: *multi-member-split simp*: *ran-m-def ideal.span-base*
intro: *diff-in-polynomial-bool-pac-idealI simp*: *H2'*)

have *H3'*: ‹*the* (*fmlookup A x12*) ∗ *q* − *r* ∈ *More-Modules.ideal polynomial-bool* ⟹
spec − *r* ∈ *More-Modules.ideal polynomial-bool* ⟹
r ∈ *pac-ideal* (*insert* (*the* (*fmlookup A x12*)) ((λ*x*. *the* (*fmlookup A x*)) ‘ *set-mset Aa*))›
for *Aa x12 r q*
by (*metis* (*no-types*, *lifting*) *ab-semigroup-mult-class.mult.commute diff-in-polynomial-bool-pac-idealI*
ideal.span-base pac-idealI3 set-image-mset set-mset-add-mset-insert union-single-eq-member)

have [*intro*]: ‹*spec* ∈ *pac-ideal* (*set-mset B*) ⟹ *spec* ∈ *pac-ideal* (*set-mset* A_0)› **and**
vars-B: ‹⋃ (*vars* ‘ *set-mset B*) ⊆ 𝒱›**and**
vars-B: ‹⋃ (*vars* ‘ *set-mset* (*ran-m A*)) ⊆ 𝒱›
using *rtranclp-PAC-Format-subset-ideal*[*OF* $A_0 B$ *vars-*A_0] *spec*$_0$ ‹(*A*, *B*) ∈ *polys-rel*›[*unfolded polys-rel-def*, *simplified*]
by (*smt in-mono mem-Collect-eq restricted-ideal-to-def*)+
have *spec-found*: ‹*PAC-Format*** ($\mathcal{V}_0$, A_0) (*V*, *add-mset r B*) ⟹
r − *spec* ∈ *ideal polynomial-bool* ⟹ *spec* ∈ *pac-ideal* (*set-mset* A_0)› **for** *V B r*
using *rtranclp-PAC-Format-subset-ideal*[*of* $\mathcal{V}_0$ A_0 *V* ‹*add-mset r B*›] *vars-*A_0 *spec*$_0$
by (*smt diff-in-polynomial-bool-pac-idealI2 in-mono mem-Collect-eq restricted-ideal-to-def*
rtranclp-PAC-Format-subset-ideal union-single-eq-member)

have *eq-successI*: ‹*st'* ≠ *FAILED* ⟹
st' ≠ *FOUND* ⟹ *st'* = *SUCCESS*› **for** *st'*
by (*cases st'*) *auto*
have *vars-diff-inv*: ‹*vars* (*Var x2* − *r*) = *vars* (*r* − *Var x2* :: *int mpoly*)› **for** *x2 r*
using *vars-uminus*[*of* ‹*Var x2* − *r*›]
by (*auto simp del*: *vars-uminus*)
have *vars-add-inv*: ‹*vars* (*Var x2* + *r*) = *vars* (*r* + *Var x2* :: *int mpoly*)› **for** *x2 r*
unfolding *add.commute*[*of* ‹*Var x2*› *r*] **..**
have *pre*: ‹*PAC-checker-step-inv spec a* 𝒱 *A*›
unfolding *PAC-checker-step-inv-def*
using *assms*
by (*smt UN-I in-dom-in-ran-m rtranclp-PAC-Format-subset-ideal subset-iff vars-B*)
have *G*[*intro*]: ‹ b^2 − *b* ∈ *ideal polynomial-bool*›
if ‹*a* − *b* ∈ *ideal polynomial-bool*› ‹a^2 − *a* ∈ *ideal polynomial-bool*›
for *a b*
proof −
have ‹(*a*−*b*) ∗ (*a*+*b*−*1*) ∈ *ideal polynomial-bool*›
using *ideal-mult-right-in that*(*1*) **by** *blast*
then have ‹−(*a*−*b*) ∗ (*a*+*b*−*1*) ∈ *ideal polynomial-bool*›
using *ideal.span-neg ideal-mult-right-in that*(*1*) **by** *blast*
then have ‹−(*a*−*b*) ∗ (*a*+*b*−*1*) + (a^2 − *a*) ∈ *ideal polynomial-bool*›

```
    using ideal.span-add that(2) by blast
  then show ‹?thesis›
    by (auto simp: algebra-simps power2-eq-square)
qed
have [iff]: ‹a ≠ FAILED› and
  [intro]: ‹a ≠ SUCCESS ⟹ a = FOUND› and
  [simp]: ‹merge-status a FOUND = FOUND›
  using assms(2) by (cases a; auto)+
note [[goals-limit=1]]
show ?thesis
  unfolding PAC-checker-step-def PAC-checker-specification-step-spec-def
    normalize-poly-spec-alt-def
    check-extension-precalc-def polys-rel-full-def check-linear-comb-def
  apply (cases st)
  apply clarsimp-all
  subgoal for x11 x12 x13
    apply (refine-vcg lhs-step-If)
    subgoal by (rule pre)
    subgoal for r eqa st'
      using assms vars-B PAC-Format-LC[OF assms(1), of 𝒱₀ A₀ ‹mset x11› r]
        spec-found[of 𝒱 r B] rtranclp-trans[of PAC-Format ‹(𝒱₀, A₀)› ‹(𝒱, B)› ‹(𝒱, add-mset r B)›]
      apply −
      apply (rule RETURN-SPEC-refine)
      apply (rule-tac x = ‹(merge-status a st',𝒱,add-mset r B)› in exI)
      apply (auto simp: polys-rel-update-remove ran-m-mapsto-upd-notin
        intro: PAC-Format-add-and-remove dest: rtranclp-PAC-Format-subset-ideal)
      done
    subgoal
      by (rule RETURN-SPEC-refine)
       (auto simp: Ex-status-iff dest: rtranclp-PAC-Format-subset-ideal)
    done
  subgoal for x31 x32 x34
    apply (refine-vcg lhs-step-If)
    subgoal by (rule pre)
    subgoal for r0 x r
      using assms vars-B apply −
      apply (rule RETURN-SPEC-refine)
      apply (rule-tac x = ‹(a,insert x32 𝒱, add-mset r B)› in exI)
      apply clarsimp-all
        apply (intro conjI)
      by (auto simp: intro!: polys-rel-update-remove PAC-Format-add-and-remove(5−)
        dest: rtranclp-PAC-Format-subset-ideal)
    subgoal
      by (rule RETURN-SPEC-refine)
       (auto simp: Ex-status-iff)
    done
  subgoal for x11
    unfolding check-del-def
    apply (refine-vcg lhs-step-If)
    subgoal by (rule pre)
    subgoal for eq
      using assms vars-B apply −
      apply (rule RETURN-SPEC-refine)
      apply (cases ‹x11 ∈# dom-m A›)
      subgoal
```

```
        apply (rule-tac x = ‹(a,𝒱, remove1-mset (the (fmlookup A x11)) B)› in exI)
        apply (auto simp: polys-rel-update-remove PAC-Format-add-and-remove
            is-failed-def is-success-def is-found-def
          dest!: eq-successI
          split: if-splits
          dest: rtranclp-PAC-Format-subset-ideal
          intro: PAC-Format-add-and-remove)
        done
      subgoal
        apply (rule-tac x = ‹(a,𝒱, B)› in exI)
        apply (auto simp: fmdrop-irrelevant
            is-failed-def is-success-def is-found-def
          dest!: eq-successI
          split: if-splits
          dest: rtranclp-PAC-Format-subset-ideal
          intro: PAC-Format-add-and-remove)
        done
      done
    subgoal
      by (rule RETURN-SPEC-refine)
        (auto simp: Ex-status-iff)
    done
  done
qed

definition PAC-checker
  :: ‹int-poly ⇒ fpac-step ⇒ status ⇒ (int-poly, nat, nat) pac-step list ⇒
    (status × fpac-step) nres›
where
  ‹PAC-checker spec A b st = do {
    (S, -) ← WHILE_T
       (λ((b :: status, A :: fpac-step), st). ¬is-failed b ∧ st ≠ [])
       (λ((bA), st). do {
          ASSERT(st ≠ []);
          S ← PAC-checker-step spec (bA) (hd st);
          RETURN (S, tl st)
        })
      ((b, A), st);
    RETURN S
  }›

lemma PAC-checker-PAC-checker-specification2:
  ‹(A, B) ∈ polys-rel-full ⟹
    ¬is-failed a ⟹
    (a = FOUND ⟹ spec ∈ pac-ideal (set-mset (snd B))) ⟹
    ⋃(vars ‘ set-mset (ran-m (snd A))) ⊆ fst B ⟹
    vars spec ⊆ fst B ⟹
  PAC-checker spec A a st ≤ ⇓ (status-rel ×_r polys-rel-full) (PAC-checker-specification2 spec B)›
  unfolding PAC-checker-def conc-fun-RES
  apply (subst RES-SPEC-eq)
  apply (refine-vcg WHILET-rule[where
      I = ‹λ((bB), st). bB ∈ (status-rel ×_r polys-rel-full)^-1 ‘‘
                    Collect (PAC-checker-specification-spec spec B)›
    and R = ‹measure (λ(-, st). Suc (length st))›])
  subgoal by auto
```

subgoal apply (*auto simp*: *PAC-checker-specification-spec-def*)
apply (*cases B*; *cases A*)
apply (*auto simp*:*polys-rel-def polys-rel-full-def Image-iff*)
done
subgoal by *auto*
subgoal
apply *auto*
apply (*rule*
PAC-checker-step-PAC-checker-specification2[*of* - - - - - - - ‹*fst B*›, *THEN order-trans*])
apply *assumption*
apply *assumption*
apply (*auto intro*: *PAC-checker-specification-spec-trans simp*: *conc-fun-RES*)
apply (*auto simp*: *PAC-checker-specification-spec-def polys-rel-full-def polys-rel-def*
dest: *PAC-Format-subset-ideal*
dest: *is-failed-is-success-completeD*; *fail*)+
by (*auto simp*: *Image-iff intro*: *PAC-checker-specification-spec-trans*
simp: *polys-rel-def polys-rel-full-def*)
subgoal
by *auto*
done

1.2 Full Checker

definition *full-checker*
:: ‹*int-poly* ⇒ (*nat*, *int-poly*) *fmap* ⇒ (*int-poly*, *nat*,*nat*) *pac-step list* ⇒ (*status* × -) *nres*›
where
‹*full-checker spec0 A pac* = *do* {
spec ← *normalize-poly-spec spec0*;
(*st*, $\mathcal{V}$, *A*) ← *remap-polys-change-all spec* {} *A*;
if is-failed st then
RETURN (*st*, $\mathcal{V}$, *A*)
else do {
$\mathcal{V}$ ← *SPEC*(λ$\mathcal{V}'$. $\mathcal{V}$ ∪ *vars spec0* ⊆ $\mathcal{V}'$);
PAC-checker spec ($\mathcal{V}$, *A*) *st pac*
}
}›

lemma *full-checker-spec*:
assumes ‹(*A*, *A'*) ∈ *polys-rel*›
shows
‹*full-checker spec A pac* ≤ ⇓{((*st*, *G*), (*st'*, *G'*)). (*st*, *st'*) ∈ *status-rel* ∧
(*st* ≠ *FAILED* ⟶ (*G*, *G'*) ∈ *polys-rel-full*)}
(*PAC-checker-specification spec* (*A'*))›
proof −
have *H*: ‹*set-mset b* ⊆ *pac-ideal* (*set-mset* (*ran-m A*)) ⟹
x ∈ *pac-ideal* (*set-mset b*) ⟹ *x* ∈ *pac-ideal* (*set-mset A'*)› **for** *b x*
using *assms* **apply** −
by (*drule pac-ideal-mono*) (*auto simp*: *polys-rel-def pac-ideal-idemp*)
have *1*: ‹*x* ∈ {(*st*, $\mathcal{V}'$, *A'*).
(¬ *is-failed st* ⟶ *pac-ideal* (*set-mset* (*ran-m x2*)) =
pac-ideal (*set-mset* (*ran-m A'*)) ∧
⋃ (*vars* ‘ *set-mset* (*ran-m ABC*)) ⊆ $\mathcal{V}'$ ∧
⋃ (*vars* ‘ *set-mset* (*ran-m A'*)) ⊆ $\mathcal{V}'$) ∧
(*st* = *FOUND* ⟶ *speca* ∈# *ran-m A'*)} ⟹
x = (*st*, *x'*) ⟹ *x'* = ($\mathcal{V}$, *Aa*) ⟹(($\mathcal{V}'$, *Aa*), $\mathcal{V}'$, *ran-m Aa*) ∈ *polys-rel-full*› **for** *Aa speca x2 st x*
$\mathcal{V}'$ $\mathcal{V}$ *x' ABC*

by (*auto simp*: *polys-rel-def polys-rel-full-def*)
have *H1*: ‹$\bigwedge$*a aa b xa x x1a x1 x2 speca*.
vars spec $\subseteq$ *x1b* $\Longrightarrow$
$\bigcup$ (*vars* ‘ *set-mset* (*ran-m A*)) $\subseteq$ *x1b* $\Longrightarrow$
$\bigcup$ (*vars* ‘ *set-mset* (*ran-m x2a*)) $\subseteq$ *x1b* $\Longrightarrow$
restricted-ideal-to$_I$ *x1b b* $\subseteq$ *restricted-ideal-to*$_I$ *x1b* (*ran-m x2a*) $\Longrightarrow$
xa $\in$ *restricted-ideal-to*$_I$ ($\bigcup$ (*vars* ‘ *set-mset* (*ran-m A*)) $\cup$ *vars spec*) *b* $\Longrightarrow$
xa $\in$ *restricted-ideal-to*$_I$ ($\bigcup$ (*vars* ‘ *set-mset* (*ran-m A*)) $\cup$ *vars spec*) (*ran-m x2a*)›
for *x1b b xa x2a*
by (*drule restricted-ideal-to-mono*[*of* - - - - ‹$\bigcup$ (*vars* ‘ *set-mset* (*ran-m A*)) $\cup$ *vars spec*›])
auto
have *H2*: ‹$\bigwedge$*a aa b speca x2 x1a x1b x2a*.
spec $-$ *speca* $\in$ *More-Modules.ideal polynomial-bool* $\Longrightarrow$
vars spec $\subseteq$ *x1b* $\Longrightarrow$
$\bigcup$ (*vars* ‘ *set-mset* (*ran-m A*)) $\subseteq$ *x1b* $\Longrightarrow$
$\bigcup$ (*vars* ‘ *set-mset* (*ran-m x2a*)) $\subseteq$ *x1b* $\Longrightarrow$
speca $\in$ *pac-ideal* (*set-mset* (*ran-m x2a*)) $\Longrightarrow$
restricted-ideal-to$_I$ *x1b b* $\subseteq$ *restricted-ideal-to*$_I$ *x1b* (*ran-m x2a*) $\Longrightarrow$
spec $\in$ *pac-ideal* (*set-mset* (*ran-m x2a*))›
by (*metis* (*no-types*, *lifting*) *group-eq-aux ideal.span-add ideal.span-base in-mono*
pac-ideal-alt-def sup.cobounded2)

show *?thesis*
supply[[*goals-limit*=*1*]]
unfolding *full-checker-def normalize-poly-spec-def*
PAC-checker-specification-def remap-polys-change-all-def
apply (*refine-vcg PAC-checker-PAC-checker-specification2*[*THEN order-trans*, *of* - -]
lhs-step-If)
subgoal by (*auto simp*: *is-failed-def RETURN-RES-refine-iff*)
apply (*rule 1*; *assumption*)
subgoal
using *fmap-ext assms* **by** (*auto simp*: *polys-rel-def ran-m-def*)
subgoal
by *auto*
subgoal
by *auto*
subgoal for *speca x1 x2 x x1a x2a x1b*
apply (*rule ref-two-step*[*OF conc-fun-R-mono*])
apply *auto*[]
using *assms*
by (*auto simp add*: *PAC-checker-specification-spec-def conc-fun-RES polys-rel-def H1 H2*
polys-rel-full-def
dest!: *rtranclp-PAC-Format-subset-ideal dest*: *is-failed-is-success-completeD*)
done
qed

lemma *full-checker-spec'*:
shows
‹(*uncurry2 full-checker*, *uncurry2* (λ*spec A* -. *PAC-checker-specification spec A*)) $\in$
(*Id* $\times_r$ *polys-rel*) $\times_r$ *Id* $\rightarrow_f$ ⟨{((*st*, *G*), (*st'*, *G'*)). (*st*, *st'*) $\in$ *status-rel* $\wedge$
(*st* $\neq$ *FAILED* $\longrightarrow$ (*G*, *G'*) $\in$ *polys-rel-full*)}⟩*nres-rel*›
using *full-checker-spec*
by (*auto intro*!: *frefI nres-relI*)

end

theory *EPAC-Checker*
  imports
    EPAC-Checker-Specification
    PAC-Checker.PAC-Map-Rel
    PAC-Checker.PAC-Polynomials-Operations
    PAC-Checker.PAC-Checker
    Show.Show
    Show.Show-Instances
begin

hide-const (**open**) *PAC-Checker-Specification.PAC-checker-step*
    PAC-Checker.PAC-checker-l PAC-Checker-Specification.PAC-checker
hide-fact (**open**) *PAC-Checker-Specification.PAC-checker-step-def*
  PAC-Checker.PAC-checker-l-def PAC-Checker-Specification.PAC-checker-def

lemma *vars-llist*[*simp*]:
  ‹*vars-llist* [] = {}›
  ‹*vars-llist* (*xs* @ *ys*) = *vars-llist xs* ∪ *vars-llist ys*›
  ‹*vars-llist* (*x* # *ys*) = *set* (*fst x*) ∪ *vars-llist ys*›
  by (*auto simp*: *vars-llist-def*)

2 Executable Checker

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

2.1 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).

Refinement relation **fun** *pac-step-rel-raw* :: ‹($'olbl \times 'lbl$) *set* ⇒ ($'a \times 'b$) *set* ⇒ ($'c \times 'd$) *set* ⇒ ($'a$, $'c$, $'olbl$) *pac-step* ⇒ ($'b$, $'d$, $'lbl$) *pac-step* ⇒ *bool*› **where**
‹*pac-step-rel-raw R1 R2 R3* (*CL p i r*) (*CL p' i' r'*) ⟷
  (*p*, *p'*) ∈ ⟨$R2 \times_r R1$⟩*list-rel* ∧ (*i*, *i'*) ∈ *R1* ∧
  (*r*, *r'*) ∈ *R2*› |
‹*pac-step-rel-raw R1 R2 R3* (*Del p1*) (*Del p1'*) ⟷
  (*p1*, *p1'*) ∈ *R1*› |
‹*pac-step-rel-raw R1 R2 R3* (*Extension i x p1*) (*Extension j x' p1'*) ⟷
  (*i*, *j*) ∈ *R1* ∧ (*x*, *x'*) ∈ *R3* ∧ (*p1*, *p1'*) ∈ *R2*› |
‹*pac-step-rel-raw R1 R2 R3* - - ⟷ *False*›

fun *pac-step-rel-assn* :: ‹($'olbl$ ⇒ $'lbl$ ⇒ *assn*) ⇒ ($'a$ ⇒ $'b$ ⇒ *assn*) ⇒ ($'c$ ⇒ $'d$ ⇒ *assn*) ⇒ ($'a$, $'c$, $'olbl$) *pac-step* ⇒ ($'b$, $'d$, $'lbl$) *pac-step* ⇒ *assn*› **where**
‹*pac-step-rel-assn R1 R2 R3* (*CL p i r*) (*CL p' i' r'*) =
  list-assn ($R2 \times_a R1$) *p p'* ∗ *R1 i i'* ∗ *R2 r r'*› |
‹*pac-step-rel-assn R1 R2 R3* (*Del p1*) (*Del p1'*) =
  R1 p1 p1'› |
‹*pac-step-rel-assn R1 R2 R3* (*Extension i x p1*) (*Extension i' x' p1'*) =
  R1 i i' ∗ *R3 x x'* ∗ *R2 p1 p1'*› |
‹*pac-step-rel-assn R1 R2* - - - = *false*›

lemma *pac-step-rel-assn-alt-def*:
⟨*pac-step-rel-assn R1 R2 R3 x y* = (
case (*x*, *y*) *of*
  (*CL p i r*, *CL p′ i′ r′*) ⇒
    list-assn (*R2* $\times_a$ *R1*) *p p′* ∗ *R1 i i′* ∗ *R2 r r′*
  | (*Del p1*, *Del p1′*) ⇒ *R1 p1 p1′*
  | (*Extension i x p1*, *Extension i′ x′ p1′*) ⇒ *R1 i i′* ∗ *R3 x x′* ∗ *R2 p1 p1′*
  | - ⇒ *false*)⟩
  by (*auto split*: *pac-step.splits*)

Addition checking

Linear Combination **definition** *check-linear-combi-l-pre-err* :: ⟨*nat* ⇒ *bool* ⇒ *bool* ⇒ *bool* ⇒ *string nres*⟩ **where**
⟨*check-linear-combi-l-pre-err r* - - - = *SPEC* (λ-. *True*)⟩

definition *check-linear-combi-l-dom-err* :: ⟨*llist-polynomial* ⇒ *nat* ⇒ *string nres*⟩ **where**
⟨*check-linear-combi-l-dom-err p r* = *SPEC* (λ-. *True*)⟩

definition *check-linear-combi-l-mult-err* :: ⟨*llist-polynomial* ⇒ *llist-polynomial* ⇒ *string nres*⟩ **where**
⟨*check-linear-combi-l-mult-err pq r* = *SPEC* (λ-. *True*)⟩
definition *linear-combi-l-pre* **where**
⟨*linear-combi-l-pre i A* $\mathcal{V}$ *xs* ⟷
(∀ *i*∈#*dom-m A*. *vars-llist* (*the* (*fmlookup A i*)) ⊆ $\mathcal{V}$)⟩

definition *linear-combi-l* **where**
⟨*linear-combi-l i A* $\mathcal{V}$ *xs* = *do* {
  ASSERT(*linear-combi-l-pre i A* $\mathcal{V}$ *xs*);
  $WHILE_T$
   (λ(*p*, *xs*, *err*). *xs* ≠ [] ∧ ¬*is-cfailed err*)
   (λ(*p*, *xs*, -). *do* {
     ASSERT(*xs* ≠ []);
     ASSERT(*vars-llist p* ⊆ $\mathcal{V}$);
     let (q_0 :: *llist-polynomial*, *i*) = *hd xs*;
     if (*i* ∉# *dom-m A* ∨ ¬(*vars-llist* q_0 ⊆ $\mathcal{V}$))
     then do {
       err ← *check-linear-combi-l-dom-err* q_0 *i*;
       RETURN (*p*, *xs*, *error-msg i err*)
     } *else do* {
       ASSERT(*fmlookup A i* ≠ *None*);
       let r = *the* (*fmlookup A i*);
       ASSERT(*vars-llist r* ⊆ $\mathcal{V}$);
       if q_0 = [([], *1*)]
       then do {
         pq ← *add-poly-l* (*p*, *r*);
         RETURN (*pq*, *tl xs*, *CSUCCESS*)
       }
       else do {
         q ← *full-normalize-poly* (q_0);
         ASSERT(*vars-llist q* ⊆ $\mathcal{V}$);
         pq ← *mult-poly-full q r*;
         ASSERT(*vars-llist pq* ⊆ $\mathcal{V}$);
         pq ← *add-poly-l* (*p*, *pq*);
         RETURN (*pq*, *tl xs*, *CSUCCESS*)

```
      }
    }
  })
  ([], xs, CSUCCESS)
}›
```

definition *check-linear-combi-l* **where**

```
‹check-linear-combi-l spec A 𝒱 i xs r = do{
b← RES(UNIV::bool set);
if b ∨ i ∈# dom-m A ∨ xs = [] ∨ ¬(vars-llist r ⊆ 𝒱)
then do {
  err ← check-linear-combi-l-pre-err i (i ∈# dom-m A) (xs = []) (¬(vars-llist r ⊆ 𝒱));
  RETURN (error-msg i err)
}
else do {
  (p, -, err) ← linear-combi-l i A 𝒱 xs;
  if (is-cfailed err)
  then do {
    RETURN err
  }
  else do {
    b ← weak-equality-l p r;
    b′ ← weak-equality-l r spec;
    if b then (if b′ then RETURN CFOUND else RETURN CSUCCESS) else do {
      c ← check-linear-combi-l-mult-err p r;
      RETURN (error-msg i c)
    }
  }
}}›
```

Deletion checking

definition *check-extension-l-side-cond-err*
:: ‹*string* ⇒ *llist-polynomial* ⇒ *llist-polynomial* ⇒ *string nres*›
where
‹*check-extension-l-side-cond-err v p′ q* = *SPEC* (λ-. *True*)›

definition (**in** −)*check-extension-l2*
:: ‹- ⇒ - ⇒ *string set* ⇒ *nat* ⇒ *string* ⇒ *llist-polynomial* ⇒ (*string code-status*) *nres*›
where

```
‹check-extension-l2 spec A 𝒱 i v p′ = do {
  b ← SPEC(λb. b ⟶ i ∉# dom-m A ∧ v ∉ 𝒱);
  if ¬b
  then do {
    c ← check-extension-l-dom-err i;
    RETURN (error-msg i c)
  } else do {
      let p′ = p′;
      let b = vars-llist p′ ⊆ 𝒱;
      if ¬b
      then do {
        c ← check-extension-l-new-var-multiple-err v p′;
        RETURN (error-msg i c)
      }
      else do {
         ASSERT(vars-llist p′ ⊆ 𝒱);
         p2 ← mult-poly-full p′ p′;
```

```
      ASSERT(vars-llist p2 ⊆ 𝒱);
      let p' = map (λ(a,b). (a, −b)) p';
      ASSERT(vars-llist p' ⊆ 𝒱);
      q ← add-poly-l (p2, p');
      ASSERT(vars-llist q ⊆ 𝒱);
      eq ← weak-equality-l q [];
      if eq then do {
        RETURN (CSUCCESS)
      } else do {
       c ← check-extension-l-side-cond-err v p' q;
       RETURN (error-msg i c)
     }
    }
  }
        }›
```

Extension checking

Step checking **definition** *PAC-checker-l-step-inv* **where**

```
‹PAC-checker-l-step-inv spec st' 𝒱 A ⟷
(∀ i∈#dom-m A. vars-llist (the (fmlookup A i)) ⊆ 𝒱)›
```

definition *PAC-checker-l-step* :: ‹- ⇒ *string code-status* × *string set* × - ⇒ (*llist-polynomial*, *string*, *nat*) *pac-step* ⇒ -› **where**

```
‹PAC-checker-l-step = (λspec (st', 𝒱, A) st. do {
  ASSERT(¬is-cfailed st');
  ASSERT(PAC-checker-l-step-inv spec st' 𝒱 A);
  case st of
   CL - - - ⇒
     do {
      ASSERT (PAC-checker-l-step-inv spec st' 𝒱 A);
       r ← full-normalize-poly (pac-res st);
      eq ← check-linear-combi-l spec A 𝒱 (new-id st) (pac-srcs 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 {
      ASSERT (PAC-checker-l-step-inv spec st' 𝒱 A);
      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)
  }
  | Extension - - - ⇒
     do {
      ASSERT (PAC-checker-l-step-inv spec st' 𝒱 A);
       r ← full-normalize-poly (pac-res st);
      eq ← check-extension-l2 spec A 𝒱 (new-id st) (new-var st) r;
      if ¬is-cfailed eq
      then do {
```

ASSERT(*new-var st* ∉ *vars-llist r* ∧ *vars-llist r* ⊆ 𝒱);
r' ← *add-poly-l* ([([*new-var st*], −*1*)], *r*);
RETURN (*st'*,
insert (*new-var st*) 𝒱, *fmupd* (*new-id st*) *r' A*)}
else RETURN (*eq*, 𝒱, *A*)
}}
)›

lemma *pac-step-rel-raw-def*:
‹⟨*K*, *V*, *R*⟩ *pac-step-rel-raw* = *pac-step-rel-raw K V R*›
by (*auto intro*!: *ext simp*: *relAPP-def*)

2.2 Correctness

We now enter the locale to reason about polynomials directly.

context *poly-embed*
begin

lemma (**in** −) *vars-llist-merge-coeffsD*:
‹*x* ∈ *vars-llist* (*merge-coeffs pa*) ⟹ *x* ∈ *vars-llist pa*›
apply (*induction pa rule*:*merge-coeffs.induct*)
apply (*auto split*: *if-splits*)
done
lemma (**in** −) *add-nset-list-rel-add-mset-iff*:
‹(*pa*, *add-mset* (*aa*) (*ys*)) ∈ ⟨*R*⟩*list-rel O* {(*c*, *a*). *a* = *mset c*} ⟷
(∃ pa_1 pa_2 *x*. *pa* = pa_1 @ *x* # pa_2 ∧ (pa_1 @ pa_2, *ys*) ∈ ⟨*R*⟩*list-rel O* {(*c*, *a*). *a* = *mset c*} ∧
(*x*, *aa*) ∈ *R*)›
apply (*rule iffI*)
subgoal
apply *clarify*
apply (*subgoal-tac* ‹*aa* ∈ *set y*›)
apply (*auto dest*!: *split-list simp*: *list-rel-split-right-iff list-rel-append1 list-rel-split-left-iff list-rel-append2*)
apply (*rule-tac x*=*cs* **in** *exI*)
apply (*rule-tac x*=*xs* **in** *exI*)
apply (*rule-tac x*=*x* **in** *exI*)
apply *simp*
apply (*rule-tac b* = ‹*ysa*@*zs*› **in** *relcompI*)
apply (*auto dest*!: *split-list simp*: *list-rel-split-right-iff list-rel-append1 list-rel-split-left-iff list-rel-append2*)
by (*metis add-mset-remove-trivial mset-remove1 multi-self-add-other-not-self remove1-idem*)
subgoal
apply (*auto dest*!: *split-list simp*: *list-rel-split-right-iff list-rel-append1 list-rel-split-left-iff list-rel-append2*)
apply (*rule-tac b* = ‹*cs*@*aa*#*ds*› **in** *relcompI*)
apply (*auto dest*!: *split-list simp*: *list-rel-split-right-iff list-rel-append1 list-rel-split-left-iff list-rel-append2*)
done
done

lemma (**in** −) *sorted-poly-rel-vars-llist2*:
‹(*pa*, *r*) ∈ *sorted-poly-rel* ⟹ (*vars-llist pa*) = ⋃ (*set-mset* ‘ *fst* ‘ *set-mset r*)›
apply (*auto split*: *if-splits simp*: *sorted-poly-list-rel-wrt-def list-mset-rel-def br-def list-rel-append1 list-rel-append2 list-rel-split-right-iff term-poly-list-rel-set-mset vars-llist-def image-Un term-poly-list-rel-def*

add-nset-list-rel-add-mset-iff dest!: *split-list*)
apply (*auto simp*: *list-rel-split-left-iff*)
done

lemma (**in** −)*normalize-poly-p-vars*: ‹*normalize-poly-p p q* $\Longrightarrow$ $\bigcup$ (*set-mset* ‘ *fst* ‘ *set-mset q*) $\subseteq$ $\bigcup$ (*set-mset* ‘ *fst* ‘ *set-mset p*)›
by (*induction rule*: *normalize-poly-p.induct*)
auto

lemma (**in** −)*rtranclp-normalize-poly-p-vars*: ‹*normalize-poly-p*** *p q* $\Longrightarrow$ $\bigcup$ (*set-mset* ‘ *fst* ‘ *set-mset q*) $\subseteq$ $\bigcup$ (*set-mset* ‘ *fst* ‘ *set-mset p*)›
by (*induction rule*: *rtranclp-induct*)
(*force dest*!: *normalize-poly-p-vars*)+

lemma *normalize-poly-normalize-poly-p2*:
assumes ‹(*p*, *p*′) ∈ *unsorted-poly-rel*›
shows ‹*normalize-poly p* ≤ ⇓{(*xs*,*ys*). (*xs*,*ys*)∈*sorted-poly-rel* ∧ *vars-llist xs* ⊆ *vars-llist p*} (*SPEC* (λ*r*. *normalize-poly-p*** *p*′ *r*))›
proof −
have *1*: ‹*sort-poly-spec p* ≤ *SPEC* (λ*p*′. *vars-llist p*′ = *vars-llist p*)›
unfolding *sort-poly-spec-def vars-llist-def*
by (*auto dest*: *mset-eq-setD*)
have [*refine*]: ‹*sort-poly-spec p* ≤ ⇓{(*xs*,*ys*). (*xs*,*ys*)∈*sorted-repeat-poly-list-rel* (*rel2p* (*Id* ∪ *term-order-rel*)) ∧
vars-llist xs ⊆ *vars-llist p*} (*RETURN p*′)›
using *sort-poly-spec-id*[*OF assms*] **apply** −
apply (*rule order-trans*)
apply (*rule SPEC-rule-conjI*[*OF 1*])
unfolding *RETURN-def*
apply (*subst* (*asm*) *conc-fun-RES*)
apply (*subst* (*asm*) *RES-SPEC-eq*)
apply *assumption*
apply (*auto simp*: *conc-fun-RES*)
done
have *1*: ‹*SPEC* (λ*r*. *normalize-poly-p*** *p*′ *r*) = *do* {
p′′ ← *RETURN p*′;
ASSERT(*p*′′ = *p*′);
SPEC (λ*r*. *normalize-poly-p*** *p*′′ *r*)
}›
by *auto*
show *?thesis*
unfolding *normalize-poly-def*
apply (*subst 1*)
apply (*refine-rcg*)
subgoal for *pa p*′
by (*force intro*!: *RES-refine simp*: *RETURN-def dest*: *vars-llist-merge-coeffsD sorted-poly-rel-vars-llist2 merge-coeffs-is-normalize-poly-p*
subsetD vars-llist-merge-coeffsD)
done
qed

lemma (**in** −)*vars-llist-mult-poly-raw*: ‹*vars-llist* (*mult-poly-raw p q*) ⊆ *vars-llist p* ∪ *vars-llist q*›
proof −
have [*simp*]: ‹*foldl* (λ*b x*. *map* (*mult-monomials x*) *qs* @ *b*) *b ps* = *foldl* (λ*b x*. *map* (*mult-monomials x*) *qs* @ *b*) [] *ps* @ *b*›
if ‹*NO-MATCH* [] *b*› **for** *qs ps b*

by (*induction ps arbitrary*: *b*)
(*simp*, *metis* (*no-types*, *lifting*) *append-assoc foldl-Cons self-append-conv*)
have [*simp*]: ‹$x \in$ *set* (*mult-monoms a aa*) $\longleftrightarrow$ $x \in$ *set a* $\vee$ $x \in$ *set aa*› **for** *x a aa*
by (*induction a aa rule*: *mult-monoms.induct*)
(*auto split*: *if-splits*)
have *0*: ‹*vars-llist* (*map* (*mult-monomials* (*a*, *ba*)) *q*) $\subseteq$ *vars-llist q* $\cup$ *set a*› **for** *a ba q*
unfolding *mult-monomials-def*
by (*induction q*) *auto*

have ‹*vars-llist* (*foldl* (λb *x*. *map* (*mult-monomials x*) *q* @ *b*) [] *p*) $\subseteq$ *vars-llist p* $\cup$ *vars-llist q* $\cup$ *vars-llist b*› **for** *b*
by (*induction p*) (*use 0* **in** *force*)+
then show *?thesis*
unfolding *mult-poly-raw-def*
by *auto*
qed

lemma *mult-poly-full-mult-poly-p′2*:
assumes ‹(*p*, *p′*) $\in$ *sorted-poly-rel*› ‹(*q*, *q′*) $\in$ *sorted-poly-rel*›
shows ‹*mult-poly-full p q* $\le \Downarrow$ {(*xs*,*ys*). (*xs*,*ys*)$\in$*sorted-poly-rel* $\wedge$ *vars-llist xs* $\subseteq$ *vars-llist p* $\cup$ *vars-llist q*} (*mult-poly-p′ p′ q′*)›
unfolding *mult-poly-full-def mult-poly-p′-def*
apply (*refine-rcg full-normalize-poly-normalize-poly-p*
normalize-poly-normalize-poly-p2[*THEN order-trans*])
apply (*subst RETURN-RES-refine-iff*)
apply (*subst Bex-def*)
apply (*subst mem-Collect-eq*)
apply (*subst conj-commute*)
apply (*rule mult-poly-raw-mult-poly-p*[*OF assms*(*1*,*2*)])
apply *assumption*
subgoal
using *vars-llist-mult-poly-raw*[*of p q*]
unfolding *conc-fun-RES*
by *auto*
done

lemma *mult-poly-full-spec2*:
assumes
‹(*p*, *p″*) $\in$ *sorted-poly-rel O mset-poly-rel*› **and**
‹(*q*, *q″*) $\in$ *sorted-poly-rel O mset-poly-rel*›
shows
‹*mult-poly-full p q* $\le \Downarrow$ {(*xs*,*ys*). (*xs*,*ys*)$\in$*sorted-poly-rel O mset-poly-rel* $\wedge$ *vars-llist xs* $\subseteq$ *vars-llist p* $\cup$ *vars-llist q*}
(*SPEC* (λs. $s - p'' * q'' \in$ *ideal polynomial-bool*))›
proof −
have *1*: ‹{(*xs*, *ys*). (*xs*, *ys*) $\in$ *sorted-poly-rel O mset-poly-rel* $\wedge$ *vars-llist xs* $\subseteq$ *vars-llist p* $\cup$ *vars-llist q*} =
{(*xs*, *ys*). (*xs*, *ys*) $\in$ *sorted-poly-rel* $\wedge$ *vars-llist xs* $\subseteq$ *vars-llist p* $\cup$ *vars-llist q*} *O*{(*xs*, *ys*). (*xs*, *ys*) $\in$ *mset-poly-rel*}›
by *blast*
obtain *p′ q′* **where**
pq: ‹(*p*, *p′*) $\in$ *sorted-poly-rel*›
‹(*p′*, *p″*) $\in$ *mset-poly-rel*›
‹(*q*, *q′*) $\in$ *sorted-poly-rel*›
‹(*q′*, *q″*) $\in$ *mset-poly-rel*›

```
    using assms by auto
  show ?thesis
    apply (rule mult-poly-full-mult-poly-p'2[THEN order-trans, OF pq(1,3)])
    apply (subst 1)
    apply (subst conc-fun-chain[symmetric])
    apply (rule ref-two-step')
    unfolding mult-poly-p'-def
    apply refine-vcg
    by (use pq assms in ‹auto simp: mult-poly-p'-def mset-poly-rel-def
      dest!: rtranclp-normalize-poly-p-poly-of-mset rtranclp-mult-poly-p-mult-ideal-final
      intro!: RES-refine›)
qed

lemma mult-poly-full-mult-poly-spec:
  assumes ‹(p, p') ∈ sorted-poly-rel O mset-poly-rel› ‹(q, q') ∈ sorted-poly-rel O mset-poly-rel›
  shows ‹mult-poly-full p q ≤ ⇓ {(xs,ys). (xs,ys)∈sorted-poly-rel O mset-poly-rel ∧ vars-llist xs ⊆ vars-llist
p ∪ vars-llist q} (mult-poly-spec p' q')›
  apply (rule mult-poly-full-spec2[OF assms, THEN order-trans])
  apply (rule ref-two-step')
  by (auto simp: mult-poly-spec-def dest: ideal.span-neg)


lemma vars-llist-merge-coeff0: ‹vars-llist (merge-coeffs0 paa) ⊆ vars-llist paa›
  by (induction paa rule: merge-coeffs0.induct)
    auto

lemma sort-poly-spec-id'2:
  assumes ‹(p, p') ∈ unsorted-poly-rel-with0›
  shows ‹sort-poly-spec p ≤ ⇓ {(xs, ys). (xs, ys) ∈ sorted-repeat-poly-rel-with0 ∧
     vars-llist xs ⊆ vars-llist p} (RETURN p')›
proof -
  obtain y where
    py: ‹(p, y) ∈ ⟨term-poly-list-rel ×r int-rel⟩list-rel› and
    p'-y: ‹p' = mset y›
    using assms
    unfolding fully-unsorted-poly-list-rel-def poly-list-rel-def sorted-poly-list-rel-wrt-def
    by (auto simp: list-mset-rel-def br-def)
  then have [simp]: ‹length y = length p›
    by (auto simp: list-rel-def list-all2-conv-all-nth)
  have H: ‹(x, p')
        ∈ ⟨term-poly-list-rel ×r int-rel⟩list-rel O list-mset-rel›
     if px: ‹mset p = mset x› and ‹sorted-wrt (rel2p (Id ∪ lexord var-order-rel)) (map fst x)›
     for x :: ‹llist-polynomial›
  proof -
    obtain f where
      f: ‹bij-betw f {..<length x} {..<length p}› and
      [simp]: ‹⋀i. i<length x ⟹ x ! i = p ! (f i)›
      using px apply - apply (subst (asm)(2) eq-commute) unfolding mset-eq-perm
      by (auto dest!: permutation-Ex-bij)
    let ?y = ‹map (λi. y ! f i) [0 ..< length x]›
    have ‹i < length y ⟹ (p ! f i, y ! f i) ∈ term-poly-list-rel ×r int-rel› for i
      using list-all2-nthD[of - p y
         ‹f i›, OF py[unfolded list-rel-def mem-Collect-eq prod.case]]
         mset-eq-length[OF px] f
      by (auto simp: list-rel-def list-all2-conv-all-nth bij-betw-def)
```

```
    then have ‹(x, ?y) ∈ ⟨term-poly-list-rel ×r int-rel⟩list-rel› and
      xy: ‹length x = length y›
      using py list-all2-nthD[of ‹rel2p (term-poly-list-rel ×r int-rel)› p y
        ‹f i› for i, simplified] mset-eq-length[OF px]
      by (auto simp: list-rel-def list-all2-conv-all-nth)
    moreover {
      have f: ‹mset-set {0..<length x} = f ‘# mset-set {0..<length x}›
        using f mset-eq-length[OF px]
        by (auto simp: bij-betw-def lessThan-atLeast0 image-mset-mset-set)
      have ‹mset y = {#y ! f x. x ∈# mset-set {0..<length x}#}›
        by (subst drop-0[symmetric], subst mset-drop-upto, subst xy[symmetric], subst f)
          auto
      then have ‹(?y, p′) ∈ list-mset-rel›
        by (auto simp: list-mset-rel-def br-def p′-y)
    }
    ultimately show ?thesis
      by (auto intro!: relcompI[of - ?y])
  qed
  show ?thesis
    unfolding sort-poly-spec-def poly-list-rel-def sorted-repeat-poly-list-rel-with0-wrt-def
    by refine-rcg (auto intro: H simp: vars-llist-def dest: mset-eq-setD)
qed

lemma sort-all-coeffs-unsorted-poly-rel-with02:
  assumes ‹(p, p′) ∈ fully-unsorted-poly-rel›
  shows ‹sort-all-coeffs p ≤ ⇓ {(xs, ys). (xs, ys) ∈ unsorted-poly-rel-with0 ∧ vars-llist xs ⊆ vars-llist p}
(RETURN p′)›
proof −
  have H: ‹(map (λ(a, y). (mset a, y)) (rev p)) =
          map (λ(a, y). (mset a, y)) s ⟷
          (map (λ(a, y). (mset a, y)) p) =
          map (λ(a, y). (mset a, y)) (rev s)› for s
    by (auto simp flip: rev-map simp: eq-commute[of ‹rev (map - -)› ‹map - -›])
  have 1: ‹⋀s y. (p, y) ∈ ⟨unsorted-term-poly-list-rel ×r int-rel⟩list-rel ⟹
           p′ = mset y ⟹
           map (λ(a, y). (mset a, y)) (rev p) = map (λ(a, y). (mset a, y)) s ⟹
           ∀x∈set s. sorted-wrt var-order (fst x) ⟹
           (s, map (λ(a, y). (mset a, y)) s)
           ∈ ⟨term-poly-list-rel ×r int-rel⟩list-rel›
    by (auto 4 4 simp: rel2p-def
        dest!: list-rel-unsorted-term-poly-list-relD
        dest: shuffle-terms-distinct-iff[THEN iffD1]
        intro!: map-mset-unsorted-term-poly-list-rel
        sorted-wrt-mono-rel[of - ‹rel2p (var-order-rel)› ‹rel2p (Id ∪ var-order-rel)›])
  have 2: ‹⋀s y. (p, y) ∈ ⟨unsorted-term-poly-list-rel ×r int-rel⟩list-rel ⟹
           p′ = mset y ⟹
           map (λ(a, y). (mset a, y)) (rev p) = map (λ(a, y). (mset a, y)) s ⟹
           ∀x∈set s. sorted-wrt var-order (fst x) ⟹
           mset y = {#case x of (a, x) ⇒ (mset a, x). x ∈# mset s#}›
    by (metis (no-types, lifting) list-rel-unsorted-term-poly-list-relD mset-map mset-rev)
  have vars-llits-alt-def:
    ‹x ∈ vars-llist p ⟷ x ∈ ⋃(set-mset ‘ fst ‘ set (map (λ(a, y). (mset a, y)) (rev p)))› for p x
    by (force simp: vars-llist-def)

  have [intro]: ‹map (λ(a, y). (mset a, y)) (rev p) = map (λ(a, y). (mset a, y)) s ⟹
```

x ∈ *vars-llist s* ⟹ *x* ∈ *vars-llist p*› **for** *s x*
unfolding *vars-llits-alt-def*
by (*auto simp*: *vars-llist-def image-image dest*!: *split-list*)
show *?thesis*
by (*rule sort-all-coeffs*[*THEN order-trans*])
(*use assms* **in** ‹*auto simp*: *shuffle-coefficients-def poly-list-rel-def*
RETURN-def fully-unsorted-poly-list-rel-def list-mset-rel-def
br-def dest: *list-rel-unsorted-term-poly-list-relD*
intro!: *RES-refine 1 2*
intro!: *relcompI*[*of* - ‹*map* (λ(*a*, *y*). (*mset a*, *y*)) (*rev p*)›]›)
qed

lemma *full-normalize-poly-normalize-poly-p2*:
assumes ‹(*p*, *p*′) ∈ *fully-unsorted-poly-rel*›
shows ‹*full-normalize-poly p* ≤ ⇓ {(*xs*, *ys*). (*xs*, *ys*) ∈ *sorted-poly-rel* ∧ *vars-llist xs* ⊆ *vars-llist p*}
(*SPEC* (λ*r*. *normalize-poly-p*** *p*′ *r*))›
(**is** ‹*?A* ≤ ⇓ *?R ?B*›)
proof −
have *1*: ‹*?B* = *do* {
p′ ← *RETURN p*′;
p′ ← *RETURN p*′;
SPEC (λ*r*. *normalize-poly-p*** *p*′ *r*)
}›
by *auto*
have [*refine0*]: ‹*sort-all-coeffs p* ≤ *SPEC*(λ*q*. (*q*, *p*′) ∈ {(*xs*, *ys*). (*xs*, *ys*) ∈ *unsorted-poly-rel-with0* ∧ *vars-llist xs* ⊆ *vars-llist p*})›
by (*rule sort-all-coeffs-unsorted-poly-rel-with02*[*OF assms*, *THEN order-trans*])
(*auto simp*: *conc-fun-RES RETURN-def*)
have [*refine0*]: ‹*sort-poly-spec p* ≤ *SPEC* (λ*c*. (*c*, *p*′) ∈
{(*xs*, *ys*). (*xs*, *ys*) ∈ *sorted-repeat-poly-rel-with0* ∧ *vars-llist xs* ⊆ *vars-llist p*})›
if ‹(*p*, *p*′) ∈ *unsorted-poly-rel-with0*›
for *p p*′
by (*rule sort-poly-spec-id*′*2*[*THEN order-trans*, *OF that*])
(*auto simp*: *conc-fun-RES RETURN-def*)
show *?thesis*
apply (*subst 1*)
unfolding *full-normalize-poly-def*
apply (*refine-rcg*)
by (*use* **in** ‹*auto intro*!: *RES-refine*
dest!: *merge-coeffs0-is-normalize-poly-p*
dest!: *set-mp*[*OF vars-llist-merge-coeff0*]
simp: *RETURN-def*›)
qed

lemma *add-poly-full-spec*:
assumes
‹(*p*, *p*′′) ∈ *sorted-poly-rel O mset-poly-rel*› **and**
‹(*q*, *q*′′) ∈ *sorted-poly-rel O mset-poly-rel*›
shows
‹*add-poly-l* (*p*, *q*) ≤ ⇓(*sorted-poly-rel O mset-poly-rel*)
(*SPEC* (λ*s*. *s* − (*p*′′ + *q*′′)∈ *ideal polynomial-bool*))›
proof −
obtain *p*′ *q*′ **where**
pq: ‹(*p*, *p*′) ∈ *sorted-poly-rel*›
‹(*p*′, *p*′′) ∈ *mset-poly-rel*›

```
    ‹(q, q′) ∈ sorted-poly-rel›
    ‹(q′, q′′) ∈ mset-poly-rel›
    using assms by auto
  show ?thesis
    apply (rule add-poly-l-add-poly-p′[THEN order-trans, OF pq(1,3)])
    apply (subst conc-fun-chain[symmetric])
    apply (rule ref-two-step′)
    by (use pq assms in ‹clarsimp simp: add-poly-p′-def mset-poly-rel-def ideal.span-zero
      dest!: rtranclp-add-poly-p-polynomial-of-mset-full
      intro!: RES-refine›)
qed
lemma (in −)add-poly-l-simps:
  ‹add-poly-l  (p, q) =
       (case (p,q) of
         (p, []) ⇒ RETURN p
       | ([], q) ⇒ RETURN q
       | ((xs, n) # p, (ys, m) # q) ⇒
           (if xs = ys then if n + m = 0 then add-poly-l (p, q) else
              do {
                pq ← add-poly-l (p, q);
                RETURN ((xs, n + m) # pq)
            }
           else if (xs, ys) ∈ term-order-rel
             then do {
                pq ← add-poly-l (p, (ys, m) # q);
                RETURN ((xs, n) # pq)
           }
           else do {
                pq ← add-poly-l ((xs, n) # p, q);
                RETURN ((ys, m) # pq)
           }))›
    apply (subst add-poly-l-def)
    apply (subst RECT-unfold, refine-mono)
    apply (subst add-poly-l-def[symmetric, abs-def])+
    apply auto
    done
lemma nat-less-induct-useful:
  assumes ‹P 0›‹(⋀m. (∀n < Suc m. P n) ⟹ P (Suc m))›
  shows ‹P m›
    using assms
    apply(induction m rule: nat-less-induct)
  apply (case-tac n)
  apply auto
  done
lemma add-poly-l-vars: ‹add-poly-l (p, q) ≤ SPEC(λxa. vars-llist xa ⊆ vars-llist p ∪ vars-llist q)›
  apply (induction length p + length q arbitrary: p q rule: nat-less-induct-useful)
  subgoal
    apply (subst add-poly-l-simps)
    apply (auto split: list.splits)
    done
  subgoal premises p for n p q
    using p(1)[rule-format, of n ‹tl p› q]
    using p(1) [rule-format, of n p ‹tl q›]  p(1)[rule-format, of ‹n−1› ‹tl p› ‹tl q›]
    using p(2−)
    apply (subst add-poly-l-simps)
```

```
    apply (case-tac p)
    subgoal by (auto split: list.splits)
    subgoal
      apply (simp split: prod.splits list.splits if-splits)
      apply (intro conjI impI allI)
      apply (auto intro: order-trans intro!: Refine-Basic.bind-rule)
      apply (rule order-trans, assumption, auto)+
      done
    done
  done
lemma pw-le-SPEC-merge: ‹f ≤ ⇓R g ⟹ f ≤ RES Φ ⟹ f ≤⇓{(x,y). (x,y)∈R ∧ x ∈ Φ} g›
   by (simp add: pw-conc-inres pw-conc-nofail pw-le-iff)
lemma add-poly-l-add-poly-p'2:
  assumes ‹(p, p') ∈ sorted-poly-rel› ‹(q, q') ∈ sorted-poly-rel›
  shows ‹add-poly-l (p, q) ≤ ⇓ {(xs,ys). (xs,ys) ∈ sorted-poly-rel ∧ vars-llist xs ⊆ vars-llist p ∪ vars-llist
q} (add-poly-p' p' q')›
  unfolding add-poly-p'-def
  apply (rule pw-le-SPEC-merge[THEN order-trans])
  apply (rule add-poly-l-spec[THEN fref-to-Down-curry-right, of - p' q'])
  using assms apply auto[2]
  apply (rule add-poly-l-vars)
  apply (auto simp: conc-fun-RES)
  done

lemma add-poly-full-spec2:
  assumes
    ‹(p, p'') ∈ sorted-poly-rel O mset-poly-rel› and
    ‹(q, q'') ∈ sorted-poly-rel O mset-poly-rel›
  shows
    ‹add-poly-l (p, q) ≤ ⇓ {(xs,ys). (xs,ys) ∈ sorted-poly-rel  O mset-poly-rel ∧ vars-llist xs ⊆ vars-llist
p ∪ vars-llist q}
    (SPEC (λs.  s − (p'' + q'' )∈ ideal polynomial-bool))›
proof −
  obtain p' q' where
    pq: ‹(p, p') ∈ sorted-poly-rel›
    ‹(p', p'') ∈ mset-poly-rel›
    ‹(q, q') ∈ sorted-poly-rel›
    ‹(q', q'') ∈ mset-poly-rel›
    using assms by auto
  have 1: ‹{(xs, ys). (xs, ys) ∈ sorted-poly-rel O mset-poly-rel  ∧ vars-llist xs ⊆ vars-llist p ∪ vars-llist
q} =
    {(xs, ys). (xs, ys) ∈ sorted-poly-rel ∧ vars-llist xs ⊆ vars-llist p ∪ vars-llist q}  O mset-poly-rel›
    by blast
  show ?thesis
    apply (rule add-poly-l-add-poly-p'2[THEN order-trans, OF pq(1,3)])
    apply (subst 1, subst conc-fun-chain[symmetric])
    apply (rule ref-two-step')
    by (use pq assms in ‹clarsimp simp: add-poly-p'-def mset-poly-rel-def ideal.span-zero
      dest!: rtranclp-add-poly-p-polynomial-of-mset-full
      intro!: RES-refine›)
qed

lemma add-poly-full-spec3:
  assumes
    ‹(p, p'') ∈ sorted-poly-rel O mset-poly-rel› and
```

⟨(q, q′′) ∈ *sorted-poly-rel O mset-poly-rel*⟩
shows
⟨*add-poly-l* (p, q) ≤ ⇓ {(xs,ys). (xs,ys) ∈ *sorted-poly-rel O mset-poly-rel* ∧ *vars-llist xs* ⊆ *vars-llist p* ∪ *vars-llist q*}
(*add-poly-spec p′′ q′′*)⟩
apply (*rule add-poly-full-spec2*[*OF assms*, *THEN order-trans*])
apply (*rule ref-two-step′*)
apply (*auto simp*: *add-poly-spec-def dest*: *ideal.span-neg*)
done

lemma *full-normalize-poly-full-spec2*:
assumes
⟨(p, p′′) ∈ *fully-unsorted-poly-rel O mset-poly-rel*⟩
shows
⟨*full-normalize-poly p* ≤ ⇓{(xs, ys). (xs, ys) ∈ *sorted-poly-rel O mset-poly-rel* ∧ *vars-llist xs* ⊆ *vars-llist p*}
(*SPEC* (λs. s − (p′′)∈ *ideal polynomial-bool* ∧ *vars s* ⊆ *vars p′′*))⟩
proof −
obtain p′ **where**
pq: ⟨(p, p′) ∈ *fully-unsorted-poly-rel*⟩
⟨(p′, p′′) ∈ *mset-poly-rel*⟩
using *assms* **by** *auto*
have *1*: ⟨⇓ {(xs, ys). (xs, ys) ∈ *sorted-poly-rel O mset-poly-rel* ∧ *vars-llist xs* ⊆ *vars-llist p*} =
⇓ ({(xs, ys). (xs, ys) ∈ *sorted-poly-rel* ∧ *vars-llist xs* ⊆ *vars-llist p*} *O mset-poly-rel*)⟩
by (*rule cong*[*of* ⟨λu. ⇓u⟩]) *auto*
show *?thesis*
apply (*rule full-normalize-poly-normalize-poly-p2*[*THEN order-trans*, *OF pq*(*1*)])
apply (*subst 1*)
apply (*subst conc-fun-chain*[*symmetric*])
apply (*rule ref-two-step′*)
by (*use pq assms* **in** ⟨*clarsimp simp*: *add-poly-p′-def mset-poly-rel-def ideal.span-zero*
ideal.span-zero rtranclp-normalize-poly-p-poly-of-mset
dest!: *rtranclp-add-poly-p-polynomial-of-mset-full*
intro!: *RES-refine*⟩)
qed
lemma (**in** −) *add-poly-l-simps-empty*[*simp*]: ⟨*add-poly-l* ([], a) = *RETURN a*⟩
by (*subst add-poly-l-simps*, *cases a*) *auto*

definition *term-rel* :: ⟨-⟩ **where**
⟨*term-rel* = *sorted-poly-rel O mset-poly-rel*⟩
definition *raw-term-rel* **where**
⟨*raw-term-rel* = *fully-unsorted-poly-rel O mset-poly-rel*⟩

fun (**in** −)*insort-wrt* :: ⟨(′a ⇒ ′b) ⇒ (′b ⇒ ′b ⇒ bool) ⇒ ′a ⇒ ′a list ⇒ ′a list⟩ **where**
⟨*insort-wrt* - - a [] = [a]⟩ |
⟨*insort-wrt f P a* (x # xs) =
(*if P* (f a) (f x) *then* a # x # xs *else* x # *insort-wrt f P a xs*)⟩

lemma (**in** −)*set-insort-wrt* [*simp*]: ⟨*set* (*insort-wrt P f a xs*) = *insert a* (*set xs*)⟩
by (*induction P f a xs rule*: *insort-wrt.induct*) *auto*

lemma (**in** −)*sorted-insort-wrt*:
⟨*transp P* ⟹ *total* (*p2rel P*) ⟹ *sorted-wrt* (λa b. P (f a) (f b)) xs ⟹ *reflp-on P* (f ‘ *set* (a # xs)) ⟹

sorted-wrt (λ*a b*. *P* (*f a*) (*f b*)) (*insort-wrt f P a xs*)›
apply (*induction f P a xs rule*: *insort-wrt.induct*)
subgoal by *auto*
subgoal for *f P a x xs*
apply (*cases* ‹*x*=*a*›)
apply (*auto simp*: *Relation.total-on-def p2rel-def reflp-on-def dest*: *transpD sympD reflpD elim*: *reflpE*)+
apply (*force simp*: *Relation.total-on-def p2rel-def reflp-on-def dest*: *transpD sympD reflpD elim*: *reflpE*)+
done
done

lemma (**in** −)*sorted-insort-wrt3*:
‹*transp P* ⟹ *total* (*p2rel P*) ⟹ *sorted-wrt* (λ*a b*. *P* (*f a*) (*f b*)) *xs* ⟹ *f a*∉*f* ‘ *set xs* ⟹
sorted-wrt (λ*a b*. *P* (*f a*) (*f b*)) (*insort-wrt f P a xs*)›
apply (*induction f P a xs rule*: *insort-wrt.induct*)
subgoal by *auto*
subgoal for *f P a x xs*
apply (*cases* ‹*x*=*a*›)
apply (*auto simp*: *Relation.total-on-def p2rel-def reflp-on-def dest*: *transpD sympD reflpD elim*: *reflpE*)
done
done
lemma (**in** −)*sorted-insort-wrt4*:
‹*transp P* ⟹ *total* (*p2rel P*) ⟹ *f a*∉*f* ‘ *set xs* ⟹ *sorted-wrt* (λ*a b*. *P* (*f a*) (*f b*)) *xs* ⟹ *f'*=(λ*a b*. *P* (*f a*) (*f b*)) ⟹
sorted-wrt f' (*insort-wrt f P a xs*)›
using *sorted-insort-wrt3*[*of P f xs a*] **by** *auto*

When *a* is empty, constants are added up.

lemma *add-poly-p-insort*:
‹*fst a* ≠ [] ⟹ *vars-llist* [*a*] ∩ *vars-llist b* = {} ⟹ *add-poly-l* ([*a*],*b*) = *RETURN* (*insort-wrt fst term-order a b*)›
apply (*induction b*)
subgoal
by (*subst add-poly-l-simps*) *auto*
subgoal for *y ys*
apply (*cases a*, *cases y*)
apply (*subst add-poly-l-simps*)
apply (*auto simp*: *rel2p-def Int-Un-distrib*)
done
done

lemma (**in** −) *map-insort-wrt*: ‹*map f* (*insort-wrt f P x xs*) = *insort-wrt id P* (*f x*) (*map f xs*)›
by (*induction xs*)
auto

lemma (**in**−) *distinct-insort-wrt*[*simp*]: ‹*distinct* (*insort-wrt f P x xs*) ⟷ *distinct* (*x* # *xs*)›
by (*induction xs*) *auto*
lemma (**in** −) *mset-insort-wrt*[*simp*]: ‹*mset* (*insort-wrt f P x xs*) = *add-mset x* (*mset xs*)›
by (*induction xs*)
auto
lemma (**in** −) *transp-term-order-rel*: ‹*transp* (λ*x y*. (*fst x*, *fst y*) ∈ *term-order-rel*)›
apply (*auto simp*: *transp-def*)
by (*smt lexord-partial-trans lexord-trans trans-less-than-char var-order-rel-def*)

lemma (**in** −) *transp-term-order*: ‹*transp term-order*›
  using *transp-term-order-rel*
  by (*auto simp*: *transp-def rel2p-def*)

lemma *total-term-order-rel*: ‹*total* (*term-order-rel*)›
  apply *standard*
  using *total-on-lexord-less-than-char-linear*[*unfolded var-order-rel-def*[*symmetric*]] **by** (*auto simp*: *p2rel-def intro*!:)

lemma *monomom-rel-mapI*: ‹*sorted-wrt* (λ*x y*. (*fst x*, *fst y*) ∈ *term-order-rel*) *r* ⟹
  distinct (*map fst r*) ⟹
  (∀ *x*∈*set r*. *distinct* (*fst x*) ∧ *sorted-wrt var-order* (*fst x*)) ⟹
  (*r*, *map* (λ(*a*, *y*). (*mset a*, *y*)) *r*) ∈ ⟨*term-poly-list-rel* $\times_r$ *int-rel*⟩*list-rel*›
  apply (*induction r*)
  subgoal
    by *auto*
  subgoal for *x xs*
    apply (*cases x*)
    apply (*auto simp*: *term-poly-list-rel-def rel2p-def*)
    done
  done

lemma *add-poly-l-single-new-var*:
  assumes ‹(*r*, *ra*) ∈ *sorted-poly-rel O mset-poly-rel*› **and**
    ‹*v* ∉ *vars-llist r*› **and**
   v: ‹(*v*, *v*′) ∈ *var-rel*›
  shows
    ‹*add-poly-l* ([([*v*], −*1*)], *r*)
    ≤ ⇓ {(*a*,*b*). (*a*,*b*)∈*sorted-poly-rel O mset-poly-rel* ∧ *vars-llist a* ⊆ *insert v* (*vars-llist r*)}
    (*SPEC*
      (λ*r0*. *r0* = *ra* − *Var v*′ ∧
    vars r0 = *vars ra* ∪ {*v*′}))›
proof −
  have [*simp*]: ‹([], *ra*) ∈ *term-rel* ⟹ ([([*v*], − *1*)], *ra* − *Var v*′) ∈ *term-rel*› **for** *ra*
    using *v*
    apply (*auto intro*!: *RETURN-RES-refine relcompI*[*of* - ‹*mset* [(*mset* [*v*], −*1*)]›]
      simp: *mset-poly-rel-def var-rel-def br-def Const-1-eq-1 term-rel-def*)
    apply (*auto simp*: *sorted-poly-list-rel-wrt-def list-mset-rel-def br-def*
      term-poly-list-rel-def
      intro!: *relcompI*[*of* - ‹[(*mset* [*v*], −*1*)]›])
    done
  have [*iff*]: ‹*v*′ ∉ *vars ra*›
  proof (*rule ccontr*)
    assume *H*: ‹¬*?thesis*›
    then have ‹φ *v* ∈ φ ‘ *vars-llist r*›
      using *assms sorted-poly-rel-vars-llist*[*OF assms*(*1*)]
      by (*auto simp*: *var-rel-def br-def*)
    then have ‹*v* ∈ *vars-llist r*›
      using φ*-inj* **by** (*auto simp*: *image-iff inj-def*)
    then show ‹*False*›
      using *assms*(*2*) **by** *fast*
  qed
  have [*simp*]: ‹([], *ra*) ∈ *term-rel* ⟹ *vars* (*ra* − *Var v*′) = *vars* (*ra*) ∪ {*v*′}› **for** *ra*
    by (*auto simp*: *term-rel-def mset-poly-rel-def*)

have [*simp*]: ‹$v' \notin$ *vars ra* $\Longrightarrow$ *vars* (*ra* − *Var* v') = *vars ra* ∪ {v'}›
  by (*auto simp add*: *vars-subst-in-left-only-diff-iff*)
have [*iff*]: ‹(\[*v*\], *b*) ∉ *set r* › **for** *b*
  using *assms*
  by (*auto simp*: *vars-llist-def*)
have
  ‹*add-poly-l* (\[(\[*v*\], −*1*)\], *r*)
  ≤ ⇓ (*sorted-poly-rel O mset-poly-rel*)
  (*SPEC*
    (λ*r0*. *r0* = *ra* − *Var* v' ∧
  vars r0 = *vars ra* ∪ {v'}))›
  using *v sorted-poly-rel-vars-llist*\[*OF assms*(*1*)\]
  apply −
  apply (*subst add-poly-p-insort*)
  apply (*use assms* **in** *auto*)
  apply (*rule RETURN-RES-refine*)
  apply *auto*
  apply (*rule-tac b*=‹*add-mset* ({#*v*#}, −*1*) (*y*)› **in** *relcompI*)
  apply (*auto simp*: *rel2p-def mset-poly-rel-def Const-1-eq-1 var-rel-def br-def*)
  apply (*auto simp*: *sorted-poly-list-rel-wrt-def sorted-wrt-map*)
  apply (*rule-tac b* = ‹*map* (λ(*a*,*b*). (*mset a*, *b*)) ((*insort-wrt fst term-order* (\[*v*\], − *1*) *r*))› **in** *relcompI*)
  apply (*auto simp*: *list-mset-rel-def br-def map-insort-wrt*)
    prefer *2*
  apply (*auto dest*!: *term-poly-list-rel-list-relD*)\[\]
    prefer *2*
  apply (*auto intro*!: *sorted-insort-wrt4 monomom-rel-mapI simp*: *rel2p-def transp-term-order total-term-order-rel transp-term-order-rel map-insort-wrt*)
  apply (*auto dest*!: *split-list simp*: *list-rel-append1 list-rel-split-right-iff term-poly-list-rel-def*)
  done
then show *?thesis*
  using *add-poly-l-vars*\[*of* ‹\[(\[*v*\], − *1*)\]› *r*\]
  unfolding *conc-fun-RES*
  apply (*subst* (*asm*) *RES-SPEC-eq*)
  apply (*rule order-trans*)
  apply (*rule SPEC-rule-conjI*)
  apply *assumption*
  apply *auto*
  done
qed

lemma *empty-sorted-poly-rel*\[*simp*,*intro*\]: ‹ (\[\], *0*) ∈ *sorted-poly-rel O mset-poly-rel*›
  by (*auto intro*!: *relcompI*\[*of* ‹\[\]›\] *simp*: *mset-poly-rel-def*)

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

lemma *single-valued-monomials*: ‹*single-valued* (⟨*term-poly-list-rel* $\times_r$ *int-rel*⟩*list-rel*)›
  by (*intro single-valued-relcomp list-rel-sv*)
  (*auto simp*: *mset-poly-rel-def sorted-poly-list-rel-wrt-def list-mset-rel-def br-def single-valued-def term-poly-list-rel-def*)
lemma *single-valued-term*: ‹*single-valued* (*sorted-poly-rel O mset-poly-rel*) ›
  using *single-valued-monomials* **apply** −
  by (*rule single-valued-relcomp*)

(*auto simp*: *mset-poly-rel-def sorted-poly-list-rel-wrt-def list-mset-rel-def br-def single-valued-def*)

lemma *single-valued-poly*:
‹(*ysa*, *cs*) ∈ ⟨*sorted-poly-rel O mset-poly-rel* $\times_r$ *nat-rel*⟩*list-rel* ⟹
(*ysa*, *csa*) ∈ ⟨*sorted-poly-rel O mset-poly-rel* $\times_r$ *nat-rel*⟩*list-rel* ⟹
cs = *csa*›
using *list-rel-sv*[*of* ‹*sorted-poly-rel O mset-poly-rel* $\times_r$ *nat-rel*›, *OF prod-rel-sv*[*OF single-valued-term*]]
by (*auto simp*: *single-valued-def*)

lemma *check-linear-combi-l-check-linear-comb*:
assumes ‹(*A*, *B*) ∈ *fmap-polys-rel*› **and** ‹(*r*, *r'*) ∈ *sorted-poly-rel O mset-poly-rel*›
‹(*i*, *i'*) ∈ *nat-rel*›
‹($\mathcal{V}'$, $\mathcal{V}$) ∈ ⟨*var-rel*⟩*set-rel*› **and**
xs: ‹(*xs*, *xs'*) ∈ ⟨(*fully-unsorted-poly-rel O mset-poly-rel*) $\times_r$ *nat-rel*⟩*list-rel*› **and**
A: ‹⋀*i*. *i* ∈# *dom-m A* ⟹ *vars-llist* (*the* (*fmlookup A i*)) ⊆ $\mathcal{V}'$›
shows
‹*check-linear-combi-l spec A* $\mathcal{V}'$ *i xs r* ≤ ⇓ {(*st*, *b*). (¬*is-cfailed st* ⟷ *b*) ∧
(*is-cfound st* ⟶ *spec* = *r*) ∧ (*b* ⟶ *vars-llist r* ⊆ $\mathcal{V}'$ ∧ *i* ∉# *dom-m A*)} (*check-linear-comb B* $\mathcal{V}$ *xs' i' r'*)›
proof −
have $\mathcal{V}$: ‹$\mathcal{V}$ = φ $\mathcal{V}'$›
using *assms*(*4*) **unfolding** *set-rel-def var-rel-def*
by (*auto simp*: *br-def*)

define *f* **where**
‹*f* = (λ*ys*::((*char list list* × *int*) *list* × *nat*) *list*.
(∀ *x* ∈ *set* (*take* (*length ys*) *xs'*). *snd x* ∈# *dom-m B* ∧ *vars* (*fst x*) ⊆ $\mathcal{V}$))›
let *?I* = ‹λ(*p*, *xs''*, *err*). ¬*is-cfailed err* ⟶
(∃ *r ys*. (*p*, *r*) ∈ *sorted-poly-rel O mset-poly-rel* ∧ *f ys* ∧ *vars-llist p* ⊆ $\mathcal{V}'$ ∧
(∑ (*p*,*n*) ∈# *mset* (*take* (*length ys*) *xs'*). *the* (*fmlookup B n*) ∗ *p*) − *r* ∈ *ideal polynomial-bool* ∧ *xs* = *ys* @ *xs''* ∧
(*xs''*, *drop* (*length ys*) *xs'*) ∈ ⟨(*fully-unsorted-poly-rel O mset-poly-rel*) $\times_r$ *nat-rel*⟩*list-rel*)›

have [*simp*]: ‹*length xs* = *length xs'*›
using *xs* **by** (*auto simp*: *list-rel-imp-same-length*)

have [*simp*]: ‹*drop* (*length ysa*) *xs'* = *cs* @ (*b*) # *ysb* ⟹ *length ysa* < *length xs'*› **for** *ysa cs b ysb*
by (*rule ccontr*) *auto*

have *Hf2*: ‹(∑ (*p*, *n*)←*cs*. *the* (*fmlookup B n*) ∗ *p*) + *the* (*fmlookup B bb*) ∗ *ad* − *xf* ∈ *More-Modules.ideal polynomial-bool*›
if *1*: ‹(∑ (*p*, *n*)←*cs*. *the* (*fmlookup B n*) ∗ *p*) − *r* ∈ *More-Modules.ideal polynomial-bool*› **and**
2: ‹*xd* − *xb* ∗ *the* (*fmlookup B bb*) ∈ *More-Modules.ideal polynomial-bool*› **and**
3: ‹*xb* − *ad* ∈ *More-Modules.ideal polynomial-bool*› **and**
4: ‹*xf* − (*r* + *xd*) ∈ *More-Modules.ideal polynomial-bool*›
for *a ba bb r ys cs ysa ad ysc x y xa xb xc xd xe xf*
proof −
have *2*: ‹*xd* − *ad* ∗ *the* (*fmlookup B bb*) ∈ *More-Modules.ideal polynomial-bool*›
using *2 3*
by (*smt diff-add-eq group-eq-aux ideal.scale-left-diff-distrib ideal.span-add-eq ideal-mult-right-in*)
note *two* = *ideal.span-neg*[*OF 2*]
note *4* = *ideal.span-neg*[*OF 4*]

note *5* = *ideal.span-add*[*OF 1 two*, *simplified*]
note *6* = *ideal.span-add*[*OF 4 5*]
show *?thesis*
using *6* **by** (*auto simp*: *algebra-simps*)
qed
have *Hf2'*: ‹(∑ (*p*, *n*)←*cs*. *the* (*fmlookup B n*) ∗ *p*) + *the* (*fmlookup B bb*) − *xf* ∈ *More-Modules.ideal polynomial-bool*›
if *1*: ‹(∑ (*p*, *n*)←*cs*. *the* (*fmlookup B n*) ∗ *p*) − *r* ∈ *More-Modules.ideal polynomial-bool*› **and**
2: ‹*xd* − *the* (*fmlookup B bb*) ∈ *More-Modules.ideal polynomial-bool*› **and**
4: ‹*xf* − (*r* + *xd*) ∈ *More-Modules.ideal polynomial-bool*›
for *a ba bb r ys cs ysa ad ysc x y xa xb xc xd xe xf*
using *Hf2*[*of cs r xd 1 bb 1 xf*] *that* **by** (*auto simp*: *ideal.span-zero*)

have [*dest*!]: ‹([([], *1*)], *ad*) ∈ *raw-term-rel* ⟹ *ad* = *1*› **for** *ad*
by (*auto simp*: *raw-term-rel-def fully-unsorted-poly-list-rel-def list-mset-rel-def Const-1-eq-1 br-def list-rel-split-right-iff unsorted-term-poly-list-rel-def mset-poly-rel-def*)

have *H*[*simp*]: ‹*length ys* < *length xs* ⟹
i < *length xs'* − *length ys* ⟷ (*i* < *length xs'* − *Suc* (*length ys*) ∨ *i* = *length xs'* − *length ys* − *1*)›
for *ys i*
by *auto*
have *lin*: ‹*linear-combi-l i' A 𝒱' xs* ≤ ⇓ {((*p*, *xs*, *err*), (*b*, *p'*)). (¬*b* ⟶ *is-cfailed err*) ∧
(*b* ⟶(*p*, *p'*) ∈ *sorted-poly-rel O mset-poly-rel*)}
(*SPEC*(λ(*b*, *r*). *b* ⟶ ((∀ *i* ∈ *set xs'*. *snd i* ∈# *dom-m B* ∧ *vars* (*fst i*) ⊆ 𝒱) ∧
(∑ (*p*,*n*) ∈# *mset xs'*. *the* (*fmlookup B n*) ∗ *p*) − *r* ∈ *ideal polynomial-bool*)))›
using *assms*(*1*) *xs*
unfolding *linear-combi-l-def conc-fun-RES check-linear-combi-l-dom-err-def term-rel-def*[*symmetric*] *raw-term-rel-def*[*symmetric*] *error-msg-def in-dom-m-lookup-iff*[*symmetric*] **apply** −
apply (*rule ASSERT-leI*)
subgoal using *assms* **unfolding** *linear-combi-l-pre-def* **by** *blast*
apply (*subst* (*2*) *RES-SPEC-eq*)
apply (*rule WHILET-rule*[**where** *R* = ‹*measure* (λ(-, *xs*, *p*). *if is-cfailed p then 0 else Suc* (*length xs*))›
and *I* = ‹*?I*›])
subgoal by *auto*
subgoal using *xs* **by** (*auto 5 5 intro*!: *exI*[*of - 0*] *intro*: *exI*[*of -xs*] *exI*[*of -* ‹[]›] *ideal.span-zero simp*: *f-def*)
subgoal for *s*
unfolding *term-rel-def*[*symmetric*]
apply (*refine-vcg full-normalize-poly-full-spec2*[*THEN order-trans*, *unfolded term-rel-def*[*symmetric*] *raw-term-rel-def*[*symmetric*]]])
subgoal
by *clarsimp*
subgoal using *xs* **by** *auto*
subgoal
by (*clarsimp simp*: *list-rel-split-right-iff list-rel-append1 neq-Nil-conv list-rel-imp-same-length*)
subgoal
by (*clarsimp simp*: *list-rel-split-right-iff list-rel-append1 neq-Nil-conv list-rel-imp-same-length*)
apply ((*use assms*(*6*) **in** ‹*solves auto*›)+)[*2*]
subgoal for *a b aa ba ab bb*
apply (*cases aa*; *cases b*)
apply (*simp only*: *prod.simps*; *clarify*)
apply (*simp only*: *prod.simps*; *clarify*)
subgoal for *ac bc list aaa baa r ys*
using *param-nth*[*of* ‹*length ys*› *xs'* ‹*length ys*› *xs* ‹*raw-term-rel* $\times_r$ *nat-rel*›]

apply (*cases* ‹*xs*′ ! *length ys*›)
apply (*auto intro*!: *add-poly-full-spec2*[*THEN order-trans*, *unfolded term-rel-def*[*symmetric*] *raw-term-rel-def*[*symmetric*]] *simp*: *conc-fun-RES*)
apply (*rule-tac x*=*ysa* **in** *exI*)
apply *auto*
apply (*auto simp*: *f-def take-Suc-conv-app-nth list-rel-imp-same-length*[*symmetric*] *single-valued-poly*)
apply (*auto dest*!: *sorted-poly-rel-vars-llist*[*unfolded term-rel-def*[*symmetric*]] *fully-unsorted-poly-rel-vars-subset-vars-llist*[*unfolded raw-term-rel-def*[*symmetric*]] *simp*: $\mathcal{V}$ *ideal.span-zero list-rel-append1 list-rel-split-right-iff list-rel-imp-same-length* *intro*!: *Hf2*′)
done
done
apply (*clarsimp simp*: *list-rel-split-right-iff list-rel-append1 neq-Nil-conv list-rel-imp-same-length*)
apply (*rule-tac P* = ‹(*x*, *fst* (*hd* (*drop* (*length xs*′ − *length* (*fst* (*snd s*))) *xs*′))) ∈ *raw-term-rel*› **in** *TrueE*)
apply (*auto simp*: *list-rel-imp-same-length*)[*2*]
apply (*clarsimp simp*: *list-rel-split-right-iff list-rel-append1 neq-Nil-conv list-rel-imp-same-length*)
apply (*auto simp*: *conc-fun-RES*)
apply *refine-vcg*
subgoal for *a ba bb r ys cs ysa ad ysc x y xa xb*
by *auto*
apply (*rule mult-poly-full-spec2*[*THEN order-trans*, *unfolded term-rel-def*[*symmetric*]])
apply *assumption*
apply *auto*
unfolding *conc-fun-RES*
apply *auto*
apply *refine-vcg*
subgoal using *A* **by** *simp blast*
apply (*rule add-poly-full-spec2*[*THEN order-trans*, *unfolded term-rel-def*[*symmetric*]])
apply *assumption*
apply (*auto simp*:)
apply (*subst conc-fun-RES*)
apply *clarsimp-all*
apply (*auto simp*: *f-def take-Suc-conv-app-nth list-rel-imp-same-length single-valued-poly*)
apply (*rule-tac x*=*yse* **in** *exI*)
apply (*auto simp*: *f-def take-Suc-conv-app-nth list-rel-imp-same-length*[*symmetric*] *single-valued-poly*)
apply (*auto dest*!: *sorted-poly-rel-vars-llist*[*unfolded term-rel-def*[*symmetric*]] *fully-unsorted-poly-rel-vars-subset-vars-llist*[*unfolded raw-term-rel-def*[*symmetric*]] *simp*: $\mathcal{V}$ *intro*: *Hf2*)[]
apply (*auto intro*: *Hf2*)
apply *force*
done
subgoal for *s*
unfolding *term-rel-def*[*symmetric*] *f-def*
apply *simp*
apply (*case-tac* ‹*is-cfailed* (*snd* (*snd s*))›; *cases s*)
apply *simp-all*
apply (*rule-tac x*=*False* **in** *exI*)
apply *clarsimp-all*
apply (*rule-tac x*=*True* **in** *exI*)
apply *clarsimp-all*
apply *auto*
done

done
have [*iff*]: ‹ $xs = [] \longleftrightarrow xs' = []$›
using *list-rel-imp-same-length*[*OF assms*(*5*)]
by (*metis length-0-conv*)
show *?thesis*
using *sorted-poly-rel-vars-llist*[*OF assms*(*2*)] *list-rel-imp-same-length*[*OF assms*(*5*)]
fmap-rel-nat-rel-dom-m[*OF assms*(*1*)] *assms*(*3*) *assms*(*2*)
unfolding *check-linear-combi-l-def check-linear-comb-def check-linear-combi-l-mult-err-def*
weak-equality-l-def conc-fun-RES term-rel-def[*symmetric*] *check-linear-combi-l-pre-err-def*
error-msg-def **apply** −
apply *simp*
apply (*refine-vcg lin*[*THEN order-trans*, *unfolded term-rel-def*[*symmetric*]])
apply (*clarsimp simp add*: *conc-fun-RES bind-RES-RETURN-eq split*: *if-splits*)
apply (*clarsimp simp add*: *conc-fun-RES bind-RES-RETURN-eq split*: *if-splits*)
apply (*clarsimp simp add*: *conc-fun-RES bind-RES-RETURN-eq split*: *if-splits*)
apply (*auto split*: *if-splits simp*: *bind-RES-RETURN-eq*)
apply (*rule lin*[*THEN order-trans*, *unfolded term-rel-def*[*symmetric*]])
apply (*clarsimp simp add*: *conc-fun-RES bind-RES-RETURN-eq split*: *if-splits*)
apply (*auto 5 3 split*: *if-splits simp*: *bind-RES-RETURN-eq* $\mathcal{V}$)
apply (*frule single-valuedD*[*OF single-valued-term*[*unfolded term-rel-def*[*symmetric*]]])
apply *assumption*
apply (*auto simp*: *conc-fun-RES*)
apply (*drule single-valuedD*[*OF single-valued-term*[*unfolded term-rel-def*[*symmetric*]]])
apply *assumption*
apply (*auto simp*: *conc-fun-RES*)
apply (*rule lin*[*THEN order-trans*, *unfolded term-rel-def*[*symmetric*]])
apply (*clarsimp simp add*: *conc-fun-RES bind-RES-RETURN-eq split*: *if-splits*)
apply (*auto 5 3 split*: *if-splits simp*: *bind-RES-RETURN-eq* $\mathcal{V}$)
apply (*drule single-valuedD*[*OF single-valued-term*[*unfolded term-rel-def*[*symmetric*]]])
apply *assumption*
apply (*auto simp*: *conc-fun-RES bind-RES-RETURN-eq*)
done
qed

definition *remap-polys-with-err* :: ‹*int mpoly* ⇒ *int mpoly* ⇒ *nat set* ⇒ (*nat*, *int-poly*) *fmap* ⇒ (*status* × *fpac-step*) *nres*› **where**
‹*remap-polys-with-err spec spec0* = ($\lambda\mathcal{V}$ *A*. *do*{
dom ← *SPEC*(λ*dom*. *set-mset* (*dom-m A*) ⊆ *dom* ∧ *finite dom*);
$\mathcal{V}$ ← *SPEC*($\lambda\mathcal{V}'$. $\mathcal{V}$ ∪ *vars spec0* ⊆ $\mathcal{V}'$);
failed ← *SPEC*(λ-::*bool*. *True*);
if failed
then do {
SPEC (λ(*mem*, -, -). *mem* = *FAILED*)
}
else do {
(*b*, *N*) ← $FOREACH_C$ *dom* (λ(*b*, $\mathcal{V}$, *A*′). ¬*is-failed b*)
(λ*i* (*b*, $\mathcal{V}$, *A*′).
if i ∈# *dom-m A*
then do {
ASSERT(¬*is-failed b*);
err ← *RES* {*FAILED*,*SUCCESS*};
if is-failed err then SPEC(λ(*err*′, $\mathcal{V}$, *A*′). *err* = *err*′)
else do {
p ← *SPEC*(λ*p*. *the* (*fmlookup A i*) − *p* ∈ *ideal polynomial-bool* ∧ *vars p* ⊆ *vars* (*the* (*fmlookup*

A i)));
    eq ← *SPEC*(λ*eq*. *eq* ≠ *FAILED* ∧ (*eq* = *FOUND* ⟶ *p* = *spec*));
    𝒱 ← *SPEC*(λ𝒱′. 𝒱 ∪ *vars* (*the* (*fmlookup A i*)) ⊆ 𝒱′);
    RETURN(*merge-status eq err*, 𝒱, *fmupd i p A*′)
    }
   }
  else RETURN (*b*, 𝒱, *A*′))
 (*SUCCESS*, 𝒱, *fmempty*);
 RETURN (*b*, *N*)
 }
})›

lemma *remap-polys-with-err-spec*:
‹*remap-polys-with-err spec spec0* 𝒱 *A* ≤ ⇓{(*a*,(*err*, 𝒱′, *A*)). *a* = (*err*, 𝒱′, *A*) ∧ (¬*is-failed err* ⟶ *vars spec0* ⊆ 𝒱′)} (*remap-polys-polynomial-bool spec* 𝒱 *A*)›
proof −
 have [*dest*]: ‹*set-mset* (*dom-m x2a*) = *set-mset* (*dom-m A*) ⟹ *dom-m A* = *dom-m x2a*› **for** *x2a*
  by (*simp add*: *distinct-mset-dom distinct-set-mset-eq-iff*)
 define *I* **where**
  [*simp*]: ‹*I* = (λ*dom* (*b*, 𝒱′, *A*′). ¬*is-failed b* ⟶
   (*set-mset* (*dom-m A*′) = *set-mset* (*dom-m A*) − *dom* ∧
   (∀ *i* ∈ *set-mset* (*dom-m A*) − *dom*. *the* (*fmlookup A i*) − *the* (*fmlookup A*′ *i*) ∈ *ideal polynomial-bool*) ∧
   ⋃(*vars* ‘ *set-mset* (*ran-m* (*fmrestrict-set* (*set-mset* (*dom-m A*′)) *A*))) ⊆ 𝒱′ ∧
   ⋃(*vars* ‘ *set-mset* (*ran-m A*′)) ⊆ 𝒱′) ∧
   𝒱 ∪ *vars spec0* ⊆ 𝒱′ ∧
  (*b* = *FOUND* ⟶ *spec* ∈# *ran-m A*′))›
 show *?thesis*
  unfolding *remap-polys-with-err-def remap-polys-polynomial-bool-def conc-fun-RES*
  apply (*rewrite at* ‹- ≤ ⌑› *RES-SPEC-eq*)
  apply (*refine-vcg FOREACHc-rule*[**where** *I* = *I*])
  subgoal by *auto*
  subgoal by *auto*
  subgoal by *auto*
  subgoal by *auto*
  subgoal by *auto*
  subgoal for *x xa xb it σ a b aa ba xc xd xe xf*
   supply[[*goals-limit=1*]]
   by (*auto simp add*: *ran-m-mapsto-upd-notin dom-m-fmrestrict-set*′ *subset-eq*)
  subgoal
   supply[[*goals-limit=1*]]
   by (*auto simp add*: *ran-m-mapsto-upd-notin dom-m-fmrestrict-set*′ *subset-eq*)
  subgoal
   by (*auto simp add*: *ran-m-mapsto-upd-notin dom-m-fmrestrict-set*′ *subset-eq fmlookup-restrict-set-id*′)
  subgoal for *x xa xb it σ a b*
   by (*cases a*)
    (*auto simp add*: *ran-m-mapsto-upd-notin dom-m-fmrestrict-set*′ *subset-eq fmlookup-restrict-set-id*′)
  done
qed

definition (**in** −) *remap-polys-l-with-err-pre*
 :: ‹*llist-polynomial* ⇒ *llist-polynomial* ⇒ *string set* ⇒ (*nat*, *llist-polynomial*) *fmap* ⇒ *bool*›
where

⟨*remap-polys-l-with-err-pre spec spec0* $\mathcal{V}$ *A* ⟷ *vars-llist spec* ⊆ *vars-llist spec0*⟩

definition (**in** −) *remap-polys-l-with-err* :: ⟨*llist-polynomial* ⇒ *llist-polynomial* ⇒ *string set* ⇒ (*nat*, *llist-polynomial*) *fmap* ⇒
(- *code-status* × *string set* × (*nat*, *llist-polynomial*) *fmap*) *nres*⟩ **where**
⟨*remap-polys-l-with-err spec spec0* = (λ$\mathcal{V}$ *A*. *do*{
ASSERT(*remap-polys-l-with-err-pre spec spec0* $\mathcal{V}$ *A*);
dom ← *SPEC*(λ*dom*. *set-mset* (*dom-m A*) ⊆ *dom* ∧ *finite dom*);
$\mathcal{V}$ ← *RETURN*($\mathcal{V}$ ∪ *vars-llist spec0*);
failed ← *SPEC*(λ-::*bool*. *True*);
if failed
then do {
c ← *remap-polys-l-dom-err*;
SPEC (λ(*mem*, -, -). *mem* = *error-msg* (*0* :: *nat*) *c*)
}
else do {
(*err*, $\mathcal{V}$, *A*) ← $FOREACH_C$ *dom* (λ(*err*, $\mathcal{V}$, *A′*). ¬*is-cfailed err*)
(λ*i* (*err*, $\mathcal{V}$, *A′*).
if i ∈# *dom-m A*
then do {
err′ ← *SPEC*(λ*err*. *err* ≠ *CFOUND*);
if is-cfailed err′ then RETURN((*err′*, $\mathcal{V}$, *A′*))
else do {
p ← *full-normalize-poly* (*the* (*fmlookup A i*));
eq ← *weak-equality-l p spec*;
$\mathcal{V}$ ← *RETURN*($\mathcal{V}$ ∪ *vars-llist* (*the* (*fmlookup A i*)));
RETURN((*if eq then CFOUND else CSUCCESS*), $\mathcal{V}$, *fmupd i p A′*)
}
} *else RETURN* (*err*, $\mathcal{V}$, *A′*))
(*CSUCCESS*, $\mathcal{V}$, *fmempty*);
RETURN (*err*, $\mathcal{V}$, *A*)
}})⟩

lemma *sorted-poly-rel-extend-vars*:
⟨(*A*, *B*) ∈ *sorted-poly-rel O mset-poly-rel* ⟹
(*x1c*, *x1a*) ∈ ⟨*var-rel*⟩*set-rel* ⟹
RETURN (*x1c* ∪ *vars-llist A*)
≤ ⇓ (⟨*var-rel*⟩*set-rel*)
(*SPEC* ((⊆) (*x1a* ∪ *vars* (*B*))))⟩
using *sorted-poly-rel-vars-llist*[*of A B*]
apply (*subst RETURN-RES-refine-iff*)
apply *clarsimp*
apply (*rule exI*[*of* - ⟨*x1a* ∪ φ ‘ *vars-llist A*⟩])
apply (*auto simp*: *set-rel-def var-rel-def br-def*
dest: *fully-unsorted-poly-rel-vars-subset-vars-llist*)
done

lemma *remap-polys-l-remap-polys*:
assumes
AB: ⟨(*A*, *B*) ∈ ⟨*nat-rel*, *fully-unsorted-poly-rel O mset-poly-rel*⟩*fmap-rel*⟩ **and**
spec: ⟨(*spec*, *spec′*) ∈ *sorted-poly-rel O mset-poly-rel*⟩ **and**
V: ⟨($\mathcal{V}$, $\mathcal{V}′$) ∈ ⟨*var-rel*⟩*set-rel*⟩ **and**
⟨(*spec0*, *spec0′*) ∈ *fully-unsorted-poly-rel O mset-poly-rel*⟩
⟨*remap-polys-l-with-err-pre spec spec0* $\mathcal{V}$ *A*⟩
shows ⟨*remap-polys-l-with-err spec spec0* $\mathcal{V}$ *A* ≤

```
    ⇓{(a,b). ¬is-cfailed (fst a) ⟶ (a,b)∈ code-status-status-rel ×r ⟨var-rel⟩set-rel ×r fmap-polys-rel}
    (remap-polys-with-err spec' spec0' 𝒱' B)›
  (is ‹- ≤ ⇓ ?R -›)
proof -
  have 1: ‹inj-on id (dom :: nat set)› for dom
    by auto
  have H: ‹x ∈# dom-m A ⟹
     (⋀p. (the (fmlookup A x), p) ∈ fully-unsorted-poly-rel ⟹
       (p, the (fmlookup B x)) ∈ mset-poly-rel ⟹ thesis) ⟹
     thesis› for x thesis
     using fmap-rel-nat-the-fmlookup[OF AB, of x x] fmap-rel-nat-rel-dom-m[OF AB] by auto
  have full-normalize-poly: ‹full-normalize-poly (the (fmlookup A x))
       ≤ ⇓ (sorted-poly-rel O mset-poly-rel)
          (SPEC
            (λp. the (fmlookup B x') - p ∈ More-Modules.ideal polynomial-bool ∧
                 vars p ⊆ vars (the (fmlookup B x'))))›
      if x-dom: ‹x ∈# dom-m A› and ‹(x, x') ∈ Id› for x x'
      apply (rule H[OF x-dom])
      subgoal for p
      apply (rule full-normalize-poly-normalize-poly-p[THEN order-trans])
      apply assumption
      subgoal
        using that(2) apply -
        unfolding conc-fun-chain[symmetric]
        by (rule ref-two-step', rule RES-refine)
         (auto simp: rtranclp-normalize-poly-p-poly-of-mset
          mset-poly-rel-def ideal.span-zero)
      done
      done

  have H': ‹(p, pa) ∈ sorted-poly-rel O mset-poly-rel ⟹
    weak-equality-l p spec ≤ ⇓{(b,enn). b = (enn=FOUND)}
    (SPEC (λeqa. eqa ≠ FAILED ∧ (eqa = FOUND ⟶ pa = spec')))› for p pa
    using spec by (auto simp: weak-equality-l-def weak-equality-spec-def RETURN-def
        list-mset-rel-def br-def mset-poly-rel-def intro!: RES-refine
      dest: list-rel-term-poly-list-rel-same-rightD sorted-poly-list-relD)
  have [refine]: ‹SPEC (λerr. err ≠ CFOUND) ≤ ⇓code-status-status-rel (RES {FAILED, SUCCESS})›
    by (auto simp: code-status-status-rel-def intro!: RES-refine)
      (case-tac s, auto)
  have [intro!]: ‹∃ a. (aa, a) ∈ ⟨var-rel⟩set-rel› for aa
    by (auto simp: set-rel-def var-rel-def br-def)

  have emp: ‹(𝒱, 𝒱') ∈ ⟨var-rel⟩set-rel ⟹
   ((CSUCCESS, 𝒱, fmempty), SUCCESS, 𝒱', fmempty) ∈ code-status-status-rel ×r ⟨var-rel⟩set-rel ×r
fmap-polys-rel› for 𝒱 𝒱'
    by auto
  show ?thesis
    using assms
    unfolding remap-polys-l-with-err-def remap-polys-l-dom-err-def
      remap-polys-with-err-def prod.case
    apply (refine-rcg full-normalize-poly fmap-rel-fmupd-fmap-rel)
    subgoal
      by auto
    apply (rule fully-unsorted-poly-rel-extend-vars)
    subgoal
```

using *assms* **by** *auto*
subgoal
by *auto*
subgoal
by *auto*
subgoal
by (*auto simp*: *error-msg-def intro*!: *RES-refine*)
apply (*rule 1*)
subgoal by *auto*
apply (*rule emp*)
subgoal
using *V* **by** *auto*
subgoal by (*auto simp*: *code-status-status-rel-def*)
subgoal by *auto*
subgoal by *auto*
subgoal by (*auto simp*: *code-status-status-rel-def RETURN-def intro*!: *RES-refine*)
subgoal by *auto*
apply (*rule H′*)
subgoal by *auto*
apply (*rule fully-unsorted-poly-rel-extend-vars*)
subgoal by (*auto intro*!: *fmap-rel-nat-the-fmlookup*)
subgoal by (*auto intro*!: *fmap-rel-fmupd-fmap-rel*)
subgoal for *dom doma failed faileda x it* σ *x′ it′* σ' *x1 x2 x1a x2a x1b x2b x1c x2c p pa err′ err - - eqa eqaa* $\mathcal{V}''$ $\mathcal{V}'''$
by (*cases eqaa*)
(*auto intro*!: *fmap-rel-fmupd-fmap-rel*)
subgoal by (*auto simp*: *code-status-status-rel-def is-cfailed-def*)
subgoal by (*auto simp*: *code-status-status-rel-def*)
done
qed

end

export-code *add-poly-l′* **in** *SML* **module-name** *test*

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
}›

lemma (**in** −) *keys-mult-monomial2*:
‹*keys* (*monomial* (*n*::*int*) (*k*::′*a* $\Rightarrow_0$ *nat*) ∗ *a*) = (*if n* = *0 then* {} *else* ((+) *k*) ‘ *keys* (*a*))›
proof −
have [*simp*]: ‹($\sum$ *aa*. (*if k* = *aa then n else 0*) ∗
($\sum$ *q*. *lookup* (*a*) *q when k* + *xa* = *aa* + *q*)) =

```
    (∑ aa. (if k = aa then n * (∑ q. lookup (a) q when k + xa = aa + q) else 0))›
   for xa
   by (smt Sum-any.cong mult-not-zero)

 show ?thesis
  apply (auto simp: vars-def times-mpoly.rep-eq Const.rep-eq times-poly-mapping.rep-eq
    Const₀-def elim!: in-keys-timesE split: if-splits)
  apply (auto simp: lookup-monomial-If prod-fun-def
    keys-def times-poly-mapping.rep-eq)
  done
qed

lemma keys-Const₀-mult-left:
 ‹keys (Const₀ (b::int) * aa) = (if b = 0 then {} else keys aa)› for aa :: ‹('a :: {cancel-semigroup-add,monoid-add}
⇒₀ nat) ⇒₀ -›
 by (auto elim!: in-keys-timesE simp: keys-mult-monomial keys-single Const₀-def keys-mult-monomial2)

hide-fact (open) poly-embed.PAC-checker-l-PAC-checker
context poly-embed
begin

definition fmap-polys-rel2 where
 ‹fmap-polys-rel2 err 𝒱 ≡ {(xs, ys). ¬ is-cfailed err ⟶ ((xs, ys) ∈ fmap-polys-rel ∧ (∀ i∈#dom-m xs.
vars-llist (the (fmlookup xs i)) ⊆ 𝒱))}›

lemma check-del-l-check-del:
 ‹(A, B) ∈ fmap-polys-rel ⟹ (x3, x3a) ∈ Id ⟹ check-del-l spec A (pac-src1 (Del x3))
 ≤ ⇓ {(st, b). (¬is-cfailed st ⟷ b) ∧ (b ⟶ st = CSUCCESS)} (check-del B (pac-src1 (Del x3a)))›
 unfolding check-del-l-def check-del-def
 by (refine-vcg lhs-step-If RETURN-SPEC-refine)
   (auto simp: fmap-rel-nat-rel-dom-m bind-RES-RETURN-eq)


lemma check-extension-alt-def:
 ‹check-extension-precalc 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);
        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 −
 have [intro]: ‹ab ∉ 𝒱 ⟹
```

```
      vars ba ⊆ 𝒱 ⟹
      MPoly-Type.coeff (ba + Var ab) (monomial (Suc 0) ab) = 1› for ab ba
    by (subst coeff-add[symmetric], subst not-in-vars-coeff0)
      (auto simp flip: coeff-add monom.abs-eq
        simp: not-in-vars-coeff0 MPoly-Type.coeff-def
          Var.abs-eq Var₀-def lookup-single-eq monom.rep-eq)
  have [simp]: ‹MPoly-Type.coeff p (monomial (Suc 0) ab) = −1›
    if ‹vars (p + Var ab) ⊆ 𝒱›
      ‹ab ∉ 𝒱›
    for ab
   proof −
    define q where ‹q ≡ p + Var ab›
    then have p: ‹p = q − Var ab›
      by auto
    show ?thesis
      unfolding p
      apply (subst coeff-minus[symmetric], subst not-in-vars-coeff0)
      using that unfolding q-def[symmetric]
      by (auto simp flip: coeff-minus simp: not-in-vars-coeff0
          Var.abs-eq Var₀-def simp flip: monom.abs-eq
          simp: not-in-vars-coeff0 MPoly-Type.coeff-def
          Var.abs-eq Var₀-def lookup-single-eq monom.rep-eq)
  qed
  have [simp]: ‹vars (p − Var ab) = vars (Var ab − p)› for ab
    using vars-uminus[of ‹p − Var ab›]
    by simp
  show ?thesis
    unfolding check-extension-def
    apply (auto 5 5 simp: check-extension-precalc-def weak-equality-def
      mult-poly-spec-def field-simps
      add-poly-spec-def power2-eq-square cong: if-cong
      intro!: intro-spec-refine[where R=Id, simplified]
      split: option.splits dest: ideal.span-add)
   done
qed

lemma check-extension-l2-check-extension:
  assumes ‹(A, B) ∈ fmap-polys-rel› and ‹(r, r') ∈ sorted-poly-rel O mset-poly-rel› and
    ‹(i, i') ∈ nat-rel› ‹(𝒱, 𝒱') ∈ ⟨var-rel⟩set-rel› ‹(x, x') ∈ var-rel›
  shows
    ‹check-extension-l2 spec A 𝒱 i x r ≤
      ⇓{((st), (b)).
        (¬is-cfailed st ⟷ b) ∧
    (is-cfound st ⟶ spec = r) ∧
    (b ⟶ vars-llist r ⊆ 𝒱 ∧ x ∉ 𝒱)} (check-extension-precalc B 𝒱' i' x' r')›
proof −
  have ‹x' = φ x›
    using assms(5) by (auto simp: var-rel-def br-def)


  have [simp]: ‹(l, l') ∈ ⟨term-poly-list-rel ×ᵣ int-rel⟩list-rel ⟹
      (map (λ(a, b). (a, − b)) l, map (λ(a, b). (a, − b)) l')
      ∈ ⟨term-poly-list-rel ×ᵣ int-rel⟩list-rel› for l l'
     by (induction l l'  rule: list-rel-induct)
       (auto simp: list-mset-rel-def br-def)
```

```
have [intro!]:
  ‹(x2c, za) ∈ ⟨term-poly-list-rel ×r int-rel⟩list-rel O list-mset-rel ⟹
   (map (λ(a, b). (a, − b)) x2c,
      {#case x of (a, b) ⇒ (a, − b). x ∈# za#})
    ∈ ⟨term-poly-list-rel ×r int-rel⟩list-rel O list-mset-rel› for x2c za
   apply (auto)
   subgoal for y
     apply (induction x2c y  rule: list-rel-induct)
     apply (auto simp: list-mset-rel-def br-def)
     apply (rename-tac a b aa l l', rule-tac b = ‹(aa, − b) # map (λ(a, b). (a, − b)) l'› in relcompI)
     by auto
   done
have [simp]: ‹(λx. fst (case x of (a, b) ⇒ (a, − b))) = fst›
  by auto

have uminus: ‹(x2c, x2a) ∈ sorted-poly-rel O mset-poly-rel ⟹
     (map (λ(a, b). (a, − b)) x2c,
      − x2a)
     ∈ sorted-poly-rel O mset-poly-rel› for x2c x2a x1c x1a
   apply (clarsimp simp: sorted-poly-list-rel-wrt-def
    mset-poly-rel-def)
  apply (rule-tac b = ‹(λ(a, b). (a, − b)) `# za› in relcompI)
  by (auto simp: sorted-poly-list-rel-wrt-def
    mset-poly-rel-def comp-def polynomial-of-mset-uminus)
 have [simp]: ‹([], 0) ∈ sorted-poly-rel O mset-poly-rel›
   by (auto simp: sorted-poly-list-rel-wrt-def
    mset-poly-rel-def list-mset-rel-def br-def
     intro!: relcompI[of - ‹{#}›])
 have [simp]: ‹vars-llist (map (λ(a, b). (a, − b)) xs) = vars-llist xs› for xs
   by (auto simp: vars-llist-def)

 show ?thesis
   unfolding check-extension-l2-def
     check-extension-l-dom-err-def
     check-extension-l-no-new-var-err-def
     check-extension-l-new-var-multiple-err-def
     check-extension-l-side-cond-err-def
    apply (rule order-trans)
    defer
    apply (rule ref-two-step')
    apply (rule check-extension-alt-def)
    apply (refine-vcg add-poly-full-spec3 mult-poly-full-mult-poly-spec)
    subgoal using assms(1,3,4,5)
      by (auto simp: var-rel-set-rel-iff)
    subgoal using assms(1,3,4,5)
      by (auto simp: var-rel-set-rel-iff)
    subgoal by auto
    subgoal by auto
    apply (rule assms)
    subgoal using sorted-poly-rel-vars-llist[of ‹r› ‹r'›] assms
      by (force simp: set-rel-def var-rel-def br-def
        dest!: sorted-poly-rel-vars-llist)
    subgoal using assms by auto
    subgoal using assms by auto
```

subgoal using *assms* **by** *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
apply (*rule uminus*)
subgoal using *assms* **by** *auto*
subgoal by *auto*
subgoal using *assms* **by** *auto*
subgoal using *assms* **by** *auto*
subgoal using *assms* **by** *auto*
subgoal using *assms* **by** (*auto simp*: *in-set-conv-decomp-first*[*of - r*] *remove1-append*)
subgoal using *assms* **by** *auto*
done
qed

lemma *PAC-checker-l-step-PAC-checker-step*:
assumes
‹(*Ast*, *Bst*) ∈{((*err*, 𝒱, *A*), (*err'*, 𝒱', *A'*)). ((*err*, 𝒱, *A*), (*err'*, 𝒱', *A'*)) ∈ (*code-status-status-rel* $\times_r$ ⟨*var-rel*⟩*set-rel* $\times_r$ *fmap-polys-rel2 err* 𝒱)}› **and**
‹(*st*, *st'*) ∈ *epac-step-rel*› **and**
spec: ‹(*spec*, *spec'*) ∈ *sorted-poly-rel O mset-poly-rel*› **and**
fail: ‹¬*is-cfailed* (*fst Ast*)›
shows
‹*PAC-checker-l-step spec Ast st* ≤
⇓{((*err*, 𝒱, *A*), (*err'*, 𝒱', *A'*)). ((*err*, 𝒱, *A*), (*err'*, 𝒱', *A'*)) ∈ (*code-status-status-rel* $\times_r$ ⟨*var-rel*⟩*set-rel* $\times_r$ *fmap-polys-rel2 err* 𝒱)}
(*PAC-checker-step spec' Bst st'*)›
proof −
obtain *A* 𝒱 *cst B* 𝒱' *cst'* **where**
Ast: ‹*Ast* = (*cst*, 𝒱, *A*)› **and**
Bst: ‹*Bst* = (*cst'*, 𝒱', *B*)› **and**
𝒱[*intro*]: ‹(𝒱, 𝒱') ∈ ⟨*var-rel*⟩*set-rel*› **and**
AB: ‹¬*is-cfailed cst* ⟹ (*A*, *B*) ∈ *fmap-polys-rel*›
‹(*cst*, *cst'*) ∈ *code-status-status-rel*›
using *assms*(*1*) **unfolding** *fmap-polys-rel2-def*
by (*cases Ast*; *cases Bst*; *auto*)
have [*intro*]: ‹*xc* ∈ 𝒱 ⟹ φ *xc* ∈ 𝒱'› **for** *xc*
using 𝒱 **by** (*auto simp*: *set-rel-def var-rel-def br-def*)
have 𝒱': ‹𝒱' = φ ‘ 𝒱›
using 𝒱
by (*auto simp*: *set-rel-def var-rel-def br-def*)
have [*refine*]: ‹(*r*, *ra*) ∈ *sorted-poly-rel O mset-poly-rel* ⟹
(*eqa*, *eqaa*)
∈ {(*st*, *b*). (¬ *is-cfailed st* ⟷ *b*) ∧ (*is-cfound st* ⟶ *spec* = *r*) ∧ (*b* ⟶ *vars-llist r* ⊆ 𝒱 ∧ *new-id step* ∉# *dom-m A*)} ⟹
RETURN eqa
≤ ⇓ *code-status-status-rel*
(*SPEC*
(λ*st'*. (¬ *is-failed st'* ∧
is-found st' ⟶
ra − *spec'* ∈ *More-Modules.ideal polynomial-bool*)))›
for *r ra eqa eqaa step*
using *spec*

by (*cases eqa*)
  (*auto intro*!: *RETURN-RES-refine dest*!: *sorted-poly-list-relD*
    simp: *mset-poly-rel-def ideal.span-zero*)
have [*simp*]: ‹(*eqa*, *st'a*) ∈ *code-status-status-rel* ⟹
    (*merge-cstatus cst eqa*, *merge-status cst' st'a*)
    ∈ *code-status-status-rel*› **for** *eqa st'a*
  using *AB*
  by (*cases eqa*; *cases st'a*)
    (*auto simp*: *code-status-status-rel-def*)
have [*simp*]: ‹(*merge-cstatus cst CSUCCESS*, *cst'*) ∈ *code-status-status-rel*›
  using *AB*
  by (*cases st*)
    (*auto simp*: *code-status-status-rel-def*)
have [*simp*]: ‹(*x32*, *x32a*) ∈ *var-rel* ⟹
    ([([*x32*], −*1*::*int*)], −*Var x32a*) ∈ *fully-unsorted-poly-rel O mset-poly-rel*› **for** *x32 x32a*
  by (*auto simp*: *mset-poly-rel-def fully-unsorted-poly-list-rel-def list-mset-rel-def br-def*
      unsorted-term-poly-list-rel-def var-rel-def Const-1-eq-1
    intro!: *relcompI*[*of* - ‹{#({#*x32*#}, −*1* :: *int*)#}›]
      relcompI[*of* - ‹[({#*x32*#}, −*1*)]›])
have *H3*: ‹*p* − *Var a* = (−*Var a*) + *p*› **for** *p* :: ‹*int mpoly*› **and** *a*
  by *auto*
have [*iff*]: ‹*x3a* ∈# *remove1-mset x3a* (*dom-m B*) ⟷ *False*› **for** *x3a B*
  using *distinct-mset-dom*[*of B*]
  by (*cases* ‹*x3a* ∈# *dom-m B*›) (*auto dest*!: *multi-member-split*)
show *?thesis*
  using *assms*(*2*)
  unfolding *PAC-checker-l-step-def PAC-checker-step-def Ast Bst prod.case*
  apply (*cases st*; *cases st'*; *simp only*: *p2rel-def pac-step.case*
    pac-step-rel-raw-def mem-Collect-eq prod.case pac-step-rel-raw.simps)
  subgoal
    apply (*refine-rcg normalize-poly-normalize-poly-spec check-linear-combi-l-check-linear-comb*
      full-normalize-poly-diff-ideal)
    subgoal using *fail* **unfolding** *Ast* **by** *auto*
    subgoal using *assms*(*1*) *fail* $\mathcal{V}'$
      unfolding *PAC-checker-l-step-inv-def* **by** (*auto simp*: *fmap-polys-rel2-def Ast Bst*
        dest!: *multi-member-split*)
    subgoal using *AB* **by** *auto*
    subgoal using *AB* **by** *auto*
    subgoal by *auto*
    subgoal by *auto*
    subgoal by *auto*
    subgoal using *AB* **unfolding** *PAC-checker-step-inv-def fmap-rel-alt-def PAC-checker-l-step-inv-def*
      by (*auto simp*: *all-conj-distrib dest*!: *multi-member-split sorted-poly-rel-vars-llist2*)
    apply *assumption*+
    subgoal
      by (*auto simp*: *code-status-status-rel-def*)
    subgoal
      using *AB*
      by (*auto intro*!: *fmap-rel-fmupd-fmap-rel fmap-rel-fmdrop-fmap-rel AB simp*: *fmap-polys-rel2-def*
PAC-checker-l-step-inv-def subset-iff)
    subgoal using *AB*
      by (*auto intro*!: *fmap-rel-fmupd-fmap-rel fmap-rel-fmdrop-fmap-rel AB simp*: *fmap-polys-rel2-def*
PAC-checker-l-step-inv-def subset-iff)
    done
  subgoal

apply (*refine-rcg full-normalize-poly-diff-ideal add-poly-l-single-new-var check-extension-l2-check-extension*)
subgoal using *fail* **unfolding** *Ast* **by** *auto*
subgoal using *assms*(*1*) *fail* $\mathcal{V}'$
unfolding *PAC-checker-l-step-inv-def* **by** (*auto simp*: *fmap-polys-rel2-def Ast Bst dest*!: *multi-member-split*)
subgoal using *AB* **by** (*auto intro*!: *fully-unsorted-poly-rel-diff*[*of* - ‹− *Var* - :: *int mpoly*›, *unfolded H3*[*symmetric*]] *simp*: *comp-def case-prod-beta*)
subgoal using *AB* **by** (*auto intro*!: *fully-unsorted-poly-rel-diff*[*of* - ‹− *Var* - :: *int mpoly*›, *unfolded H3*[*symmetric*]] *simp*: *comp-def case-prod-beta*)
subgoal using *AB* **by** *auto*
subgoal using *AB* **by** *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *simp*
subgoal by *simp*
subgoal by *simp*
subgoal using *AB* $\mathcal{V}$
by (*auto simp*: *fmap-polys-rel2-def PAC-checker-l-step-inv-def intro*!: *fmap-rel-fmupd-fmap-rel insert-var-rel-set-rel dest*!: *in-diffD*)
subgoal
by (*auto simp*: *code-status-status-rel-def AB fmap-polys-rel2-def code-status.is-cfailed-def*)
done
subgoal
apply (*refine-rcg normalize-poly-normalize-poly-spec check-del-l-check-del check-addition-l-check-add full-normalize-poly-diff-ideal*[*unfolded normalize-poly-spec-def*[*symmetric*]])
subgoal using *fail* **unfolding** *Ast* **by** *auto*
subgoal using *assms*(*1*) *fail* $\mathcal{V}'$
unfolding *PAC-checker-l-step-inv-def* **by** (*auto simp*: *fmap-polys-rel2-def Ast Bst dest*!: *multi-member-split*)
subgoal using *AB* **by** *auto*
subgoal using *AB* **by** *auto*
subgoal using *AB*
by (*auto intro*!: *fmap-rel-fmupd-fmap-rel fmap-rel-fmdrop-fmap-rel code-status-status-rel-def simp*: *fmap-polys-rel2-def PAC-checker-l-step-inv-def dest*: *in-diffD*)
subgoal
using *AB*
by (*auto intro*!: *fmap-rel-fmupd-fmap-rel fmap-rel-fmdrop-fmap-rel simp*: *fmap-polys-rel2-def*)
done
done
qed

lemma *PAC-checker-l-PAC-checker*:
assumes
‹$(A, B) \in \{((\mathcal{V}, A), (\mathcal{V}', A')).\ ((\mathcal{V}, A), (\mathcal{V}', A')) \in (\langle var\text{-}rel\rangle set\text{-}rel \times_r fmap\text{-}polys\text{-}rel2\ b\ \mathcal{V})\}$›
(**is** ‹- ∈ *?A*›) **and**
‹$(st, st') \in \langle epac\text{-}step\text{-}rel\rangle list\text{-}rel$› **and**
‹$(spec, spec') \in sorted\text{-}poly\text{-}rel\ O\ mset\text{-}poly\text{-}rel$› **and**

‹$(b,\ b') \in$ *code-status-status-rel*›
shows
‹*PAC-checker-l spec A b st* $\leq$
$\Downarrow$ {((*err*, $\mathcal{V}$, *A*), (*err*′, $\mathcal{V}'$, *A*′)). ((*err*, $\mathcal{V}$, *A*), (*err*′, $\mathcal{V}'$, *A*′)) ∈ (*code-status-status-rel* $\times_r$ ⟨*var-rel*⟩*set-rel* $\times_r$ *fmap-polys-rel2 err* $\mathcal{V}$)} (*PAC-checker spec*′ *B b*′ *st*′)›
proof −
have [*refine0*]: ‹(((*b*, *A*), *st*), (*b*′, *B*), *st*′) ∈
({((*err*, $\mathcal{V}$, *A*), (*err*′, $\mathcal{V}'$, *A*′)). ((*err*, $\mathcal{V}$, *A*), (*err*′, $\mathcal{V}'$, *A*′)) ∈ (*code-status-status-rel* $\times_r$ ⟨*var-rel*⟩*set-rel* $\times_r$ *fmap-polys-rel2 err* $\mathcal{V}$)} $\times_r$
⟨*epac-step-rel*⟩*list-rel*)›
using *assms* **by** (*auto simp*: *code-status-status-rel-def*)
show *?thesis*
using *assms*
unfolding *PAC-checker-l-def PAC-checker-def*
apply (*refine-rcg PAC-checker-l-step-PAC-checker-step*)
subgoal by (*auto simp*: *code-status-status-rel-discrim-iff*
WHILEIT-refine[**where** *R* = ‹(*?A* $\times_r$ ⟨*pac-step-rel*⟩*list-rel*)›])
subgoal by *auto*
subgoal by (*auto simp*: *neq-Nil-conv*)
subgoal by (*auto simp*: *neq-Nil-conv intro*!: *param-nth*)
subgoal by (*auto simp*: *neq-Nil-conv*)
subgoal by (*auto simp*: *neq-Nil-conv fmap-polys-rel2-def*)
subgoal by (*auto simp*: *neq-Nil-conv fmap-polys-rel2-def*)
done
qed

lemma *sorted-poly-rel-extend-vars2*:
‹(*A*, *B*) ∈ *sorted-poly-rel O mset-poly-rel* ⟹
(*x1c*, *x1a*) ∈ ⟨*var-rel*⟩*set-rel* ⟹
RETURN (*x1c* ∪ *vars-llist A*)
≤ ⇓ {(*a*,*b*). (*a*,*b*) ∈ ⟨*var-rel*⟩*set-rel* ∧ *a* = *x1c* ∪ *vars-llist A*}
(*SPEC* ((⊆) (*x1a* ∪ *vars* (*B*))))›
using *sorted-poly-rel-vars-llist*[*of A B*]
apply (*subst RETURN-RES-refine-iff*)
apply *clarsimp*
apply (*rule exI*[*of* - ‹*x1a* ∪ φ ‘ *vars-llist A*›])
apply (*auto simp*: *set-rel-def var-rel-def br-def*
dest: *fully-unsorted-poly-rel-vars-subset-vars-llist*)
done

lemma *fully-unsorted-poly-rel-extend-vars2*:
‹(*A*, *B*) ∈ *fully-unsorted-poly-rel O mset-poly-rel* ⟹
(*x1c*, *x1a*) ∈ ⟨*var-rel*⟩*set-rel* ⟹
RETURN (*x1c* ∪ *vars-llist A*)
≤ ⇓ {(*a*,*b*). (*a*,*b*) ∈ ⟨*var-rel*⟩*set-rel* ∧ *a* = *x1c* ∪ *vars-llist A*}
(*SPEC* ((⊆) (*x1a* ∪ *vars* (*B*))))›
using *fully-unsorted-poly-rel-vars-subset-vars-llist*[*of A B*]
apply (*subst RETURN-RES-refine-iff*)
apply *clarsimp*
apply (*rule exI*[*of* - ‹*x1a* ∪ φ ‘ *vars-llist A*›])
apply (*auto simp*: *set-rel-def var-rel-def br-def*
dest: *fully-unsorted-poly-rel-vars-subset-vars-llist*)
done

```
lemma remap-polys-l-with-err-remap-polys-with-err:
  assumes
    AB: ‹(A, B) ∈ ⟨nat-rel, fully-unsorted-poly-rel O mset-poly-rel⟩fmap-rel› and
    spec: ‹(spec, spec′) ∈ sorted-poly-rel O mset-poly-rel› and
    V: ‹(𝒱, 𝒱′) ∈ ⟨var-rel⟩set-rel› and
    spec0: ‹(spec0, spec0′) ∈ fully-unsorted-poly-rel O mset-poly-rel› and
    pre: ‹remap-polys-l-with-err-pre spec spec0 𝒱 A›
  shows ‹remap-polys-l-with-err spec spec0 𝒱 A ≤
    ⇓{((err, 𝒱, A), (err′, 𝒱′, A′)). (err, err′) ∈ code-status-status-rel ∧
        (¬is-cfailed err ⟶ ((err, 𝒱, A), (err′, 𝒱′, A′)) ∈ (code-status-status-rel ×r ⟨var-rel⟩set-rel ×r
fmap-polys-rel2 err 𝒱))}
        (remap-polys-with-err spec′ spec0′ 𝒱′ B)›
  (is ‹- ≤ ⇓ ?R -›)
proof −
  have 1: ‹inj-on id (dom :: nat set)› for dom
    by auto
  have H: ‹x ∈# dom-m A ⟹
      (⋀p. (the (fmlookup A x), p) ∈ fully-unsorted-poly-rel ⟹
        (p, the (fmlookup B x)) ∈ mset-poly-rel ⟹ thesis) ⟹
      thesis› for x thesis
      using fmap-rel-nat-the-fmlookup[OF AB, of x x] fmap-rel-nat-rel-dom-m[OF AB] by auto

  have full-normalize-poly: ‹full-normalize-poly (the (fmlookup A x))
        ≤ ⇓ {(xs, ys). (xs, ys) ∈ sorted-poly-rel O mset-poly-rel ∧ vars-llist xs ⊆ vars-llist (the (fmlookup
A x))}
          (SPEC
            (λp. the (fmlookup B x′) − p ∈ More-Modules.ideal polynomial-bool ∧
                vars p ⊆ vars (the (fmlookup B x′))))› (is ‹- ≤ ⇓?A -›)
      if x-dom: ‹x ∈# dom-m A› and ‹(x, x′) ∈ Id› for x x′
      apply (rule H[OF x-dom])
      subgoal for p
      apply (rule full-normalize-poly-normalize-poly-p2[THEN order-trans])
      apply assumption
      subgoal
        using that(2) apply −
        unfolding conc-fun-chain[symmetric]
        by
         (auto simp: rtranclp-normalize-poly-p-poly-of-mset conc-fun-RES
            mset-poly-rel-def ideal.span-zero
            intro!: exI[of - ‹polynomial-of-mset -›])
      done
      done

  have H′: ‹(p, pa) ∈ sorted-poly-rel O mset-poly-rel ⟹
    weak-equality-l p spec ≤ ⇓{(b,enn). b = (enn=FOUND)}
    (SPEC (λeqa. eqa ≠ FAILED ∧ (eqa = FOUND ⟶ pa = spec′)))› for p pa
    using spec by (auto simp: weak-equality-l-def weak-equality-spec-def RETURN-def
        list-mset-rel-def br-def mset-poly-rel-def intro!: RES-refine
      dest: list-rel-term-poly-list-rel-same-rightD sorted-poly-list-relD)
 have [refine]: ‹SPEC (λerr. err ≠ CFOUND) ≤ ⇓code-status-status-rel (RES {FAILED, SUCCESS})›
   by (auto simp: code-status-status-rel-def intro!: RES-refine)
     (case-tac s, auto)
  have [intro!]: ‹∃ a. (aa, a) ∈ ⟨var-rel⟩set-rel› for aa
    by (auto simp: set-rel-def var-rel-def br-def)
```

have *emp*: ‹($\mathcal{V}$, $\mathcal{V}'$) ∈ ⟨*var-rel*⟩*set-rel* ⟹
((*CSUCCESS*, $\mathcal{V}$, *fmempty*), *SUCCESS*, $\mathcal{V}'$, *fmempty*) ∈
{((*err*, $\mathcal{V}$, *A*), (*f′*, $\mathcal{V}'$, *A′*)). ((*err*, $\mathcal{V}$, *A*), (*f′*, $\mathcal{V}'$, *A′*)) ∈
(*code-status-status-rel* $\times_r$ ⟨*var-rel*⟩*set-rel* $\times_r$ *fmap-polys-rel2 err* $\mathcal{V}$)}›
for $\mathcal{V}$ $\mathcal{V}'$
by (*auto simp*: *fmap-polys-rel2-def*)
have *XXX*: ‹($\mathcal{V}''$, $\mathcal{V}'''$) ∈ ⟨*var-rel*⟩*set-rel* ⟹ *x* ∈ $\mathcal{V}''$ ⟹ *φ x* ∈ $\mathcal{V}'''$› **for** *x* $\mathcal{V}''$ $\mathcal{V}'''$
by (*auto simp*: *br-def set-rel-def var-rel-def*)

show *?thesis*
using *assms*
unfolding *remap-polys-l-with-err-def remap-polys-l-dom-err-def*
remap-polys-with-err-def prod.case term-rel-def[*symmetric*]
apply (*refine-rcg full-normalize-poly fmap-rel-fmupd-fmap-rel*)

subgoal
by *auto*
apply (*rule fully-unsorted-poly-rel-extend-vars2*[*unfolded term-rel-def*[*symmetric*]])
subgoal using *spec0* **by** *auto*
subgoal by *auto*
subgoal by *auto*
subgoal
by (*auto simp*: *error-msg-def fmap-polys-rel2-def intro*!: *RES-refine*)
apply (*rule 1*)
subgoal by *auto*
apply (*rule emp*)
subgoal
using *V* **by** *auto*
subgoal by (*auto simp*: *code-status-status-rel-def*)
subgoal by *auto*
subgoal by *auto*
subgoal by (*auto simp*: *code-status-status-rel-def RETURN-def fmap-polys-rel2-def intro*!: *RES-refine*)
subgoal by *auto*
apply (*rule H′*)
subgoal by *auto*
apply (*rule fully-unsorted-poly-rel-extend-vars2*[*unfolded term-rel-def*[*symmetric*]])
subgoal by (*auto intro*!: *fmap-rel-nat-the-fmlookup*)
subgoal by (*auto intro*!: *fmap-rel-fmupd-fmap-rel*)
subgoal for *dom doma failed faileda x it σ x′ it′ σ′ x1 x2 x1a x2a x1b x2b x1c x2c p pa err′ err - -*
eqa eqaa $\mathcal{V}''$ $\mathcal{V}'''$
unfolding *term-rel-def*[*symmetric*]
by (*cases eqaa*)
(*auto simp*: *fmap-rel-fmupd-fmap-rel*[**where** *R* = ‹*sorted-poly-rel O mset-poly-rel*›, *unfolded term-rel-def*[*symmetric*]]
simp: *fmap-polys-rel2-def code-status-status-rel-def term-rel-def*[*symmetric*]
dest: *in-diffD*)
subgoal by (*auto simp*: *code-status-status-rel-def is-cfailed-def*)
subgoal by (*auto simp*: *code-status-status-rel-def*)
done
qed

definition (**in** −) *full-checker-l*
:: ‹*llist-polynomial* ⇒ (*nat*, *llist-polynomial*) *fmap* ⇒ (-, *string*, *nat*) *pac-step list* ⇒
(*string code-status* × -) *nres*›
where

⟨*full-checker-l spec A st* = *do* {
  spec′ ← *full-normalize-poly spec*;
  (*b*, $\mathcal{V}$, *A*) ← *remap-polys-l-with-err spec′ spec* {} *A*;
  if is-cfailed b
  then RETURN (*b*, $\mathcal{V}$, *A*)
  else do {
    let $\mathcal{V}$ = $\mathcal{V}$;
    PAC-checker-l spec′ ($\mathcal{V}$, *A*) *b st*
  }
}⟩

lemma (**in** −)*RES-RES-RETURN-RES3*: ⟨*RES A* ≫= (λ(*a*,*b*,*c*). *RES* (*f a b c*)) = *RES* (⋃((λ(*a*,*b*,*c*). *f a b c*) ' *A*))⟩ **for** *A f*
  by (*auto simp*: *pw-eq-iff refine-pw-simps*)

definition *vars-rel2* :: ⟨-⟩ **where**
  ⟨*vars-rel2 err* = {(*A*,*B*). ¬*is-cfailed err* ⟶ (*A*,*B*)∈ ⟨*var-rel*⟩*set-rel*} ⟩

lemma *full-normalize-poly-normalize-poly-spec-vars2*: ⟨(*p3*, *p1*) ∈ *fully-unsorted-poly-rel O mset-poly-rel* ⟹
  full-normalize-poly p3
  ≤ ⇓ ({(*xs*, *ys*). (*xs*, *ys*) ∈ *sorted-poly-rel* ∧ *vars-llist xs* ⊆ *vars-llist p3*} *O*
  mset-poly-rel)
  (*normalize-poly-spec p1*)
  ⟩
  using *full-normalize-poly-normalize-poly-p2*[*unfolded normalize-poly-spec-alt-def*[*symmetric*], *THEN ref-two-step*[*OF* - *normalize-poly-p-normalize-poly-spec*], *unfolded conc-fun-chain*]
  by *auto*

lemma *full-checker-l-full-checker*:
 assumes
   ⟨(*A*, *B*) ∈ *unsorted-fmap-polys-rel*⟩ **and**
   st: ⟨(*st*, *st′*) ∈ ⟨*epac-step-rel*⟩*list-rel*⟩ **and**
   spec: ⟨(*spec*, *spec′*) ∈ *fully-unsorted-poly-rel O mset-poly-rel*⟩
  shows
   ⟨*full-checker-l spec A st* ≤ ⇓ {((*err*, $\mathcal{V}$, *A*), *err′*, $\mathcal{V}'$, *A′*).
       ((*err*, $\mathcal{V}$, *A*), *err′*, $\mathcal{V}'$, *A′*) ∈ *code-status-status-rel* $\times_r$ *vars-rel2 err* $\times_r$ *fmap-polys-rel2 err* $\mathcal{V}$}
(*full-checker spec′ B st′*)⟩
proof −
  have *aa*: ⟨{((*err*, $\mathcal{V}$, *A*), *err′*, $\mathcal{V}'$, *A′*). (*err*,*err′*) ∈ *code-status-status-rel* ∧
   (¬ *is-cfailed err* ⟶
   ((*err*, $\mathcal{V}$, *A*), *err′*, $\mathcal{V}'$, *A′*) ∈ *code-status-status-rel* $\times_r$ ⟨*var-rel*⟩*set-rel* $\times_r$ *fmap-polys-rel2 err* $\mathcal{V}$) ∧
   (¬*is-failed err′* ⟶ *vars spec′* ⊆ $\mathcal{V}'$)} =
   {((*err*, $\mathcal{V}$, *A*), *err′*, $\mathcal{V}'$, *A′*). (*err*,*err′*) ∈ *code-status-status-rel* ∧ (¬ *is-cfailed err* ⟶
   ((*err*, $\mathcal{V}$, *A*), *err′*, $\mathcal{V}'$, *A′*) ∈ *code-status-status-rel* $\times_r$ ⟨*var-rel*⟩*set-rel* $\times_r$ *fmap-polys-rel2 err* $\mathcal{V}$)} *O*
   {((*err*, $\mathcal{V}$, *A*), *err′*, $\mathcal{V}'$, *A′*). ((*err*, $\mathcal{V}$, *A*), *err′*, $\mathcal{V}'$, *A′*) ∈ *Id* ∧ (¬*is-failed err′* ⟶ *vars spec′* ⊆ $\mathcal{V}'$)}⟩
for *spec′*
    by *auto*
  have [*refine*]:
   ⟨(*spec*, *spec′*) ∈ *sorted-poly-rel O mset-poly-rel* ⟹
   (*spec0*, *spec0′*) ∈ *fully-unsorted-poly-rel O mset-poly-rel* ⟹
   ($\mathcal{V}$, $\mathcal{V}'$) ∈ ⟨*var-rel*⟩*set-rel* ⟹ *remap-polys-l-with-err-pre spec spec0* $\mathcal{V}$ *A* ⟹
  remap-polys-l-with-err spec spec0 $\mathcal{V}$ *A* ≤ ⇓ {((*err*, $\mathcal{V}$, *A*), *err′*, $\mathcal{V}'$, *A′*). (*err*,*err′*) ∈ *code-status-status-rel* ∧ (¬ *is-cfailed err* ⟶
   ((*err*, $\mathcal{V}$, *A*), *err′*, $\mathcal{V}'$, *A′*) ∈ *code-status-status-rel* $\times_r$ ⟨*var-rel*⟩*set-rel* $\times_r$ *fmap-polys-rel2 err* $\mathcal{V}$) ∧

$(\neg$*is-failed err'* $\longrightarrow$ *vars spec0'* $\subseteq \mathcal{V}')\}$
(*remap-polys-change-all spec'* $\mathcal{V}'$ *B*)› (**is** ‹- $\Longrightarrow$ - $\Longrightarrow$ - $\Longrightarrow$ - $\Longrightarrow$ - $\leq \Downarrow$ *?A* -›)
for *spec spec'* $\mathcal{V}$ $\mathcal{V}'$ *spec0 spec0'*
apply (*rule remap-polys-l-with-err-remap-polys-with-err*[*THEN order-trans*, *OF assms*(*1*)])
apply *assumption+*
apply (*subst aa*, *subst conc-fun-chain*[*symmetric*])
apply (*rule ref-two-step*[*OF order.refl*])
apply (*rule remap-polys-with-err-spec*[*THEN order-trans*])
apply (*rule conc-fun-R-mono*[*THEN order-trans*, *of* - ‹$\{((err, \mathcal{V}, A), err', \mathcal{V}', A')$.
$((err, \mathcal{V}, A), err', \mathcal{V}', A') \in Id \wedge (\neg$ *is-failed err'* $\longrightarrow$ *vars spec0'* $\subseteq \mathcal{V}')\}$›])
apply (*solves auto*)
apply (*subst ref-two-step'*)
apply (*rule remap-polys-polynomial-bool-remap-polys-change-all*)
apply (*solves* ‹*rule TrueI*›)
done

have *unfold-code-status*: ‹$(\exists (a).\ P\ a) \longleftrightarrow (\exists a.\ P\ (CFAILED\ a)) \vee P\ CFOUND \vee P\ CSUCCESS$›
for *P*
by (*auto*, *case-tac a*, *auto*)
have *unfold-status*: ‹$(\exists (a).\ P\ a) \longleftrightarrow (\ P\ (FAILED)) \vee P\ FOUND \vee P\ SUCCESS$› **for** *P*
by (*auto*, *case-tac a*, *auto*)
have *var-rel-set-rel-alt-def*: ‹$(A, B) \in \langle var\text{-}rel\rangle set\text{-}rel \longleftrightarrow B = \varphi$ ‘ A› **for** *A B*
by (*auto simp*: *var-rel-def set-rel-def br-def*)
have [*refine*]: ‹$(x1c, x1a) \in \langle var\text{-}rel\rangle set\text{-}rel \Longrightarrow$
SPEC $(\lambda(err, \mathcal{V}').\ (err = CSUCCESS \vee is\text{-}cfailed\ err) \wedge (err = CSUCCESS \longrightarrow \mathcal{V}' = x1c \cup$ *vars-llist spec*))
$\leq \Downarrow$ (*code-status-status-rel* $\times_r$ $\langle var\text{-}rel\rangle set\text{-}rel$)
(*SPEC* $(\lambda(err, \mathcal{V}').\ (err = SUCCESS \vee is\text{-}failed\ err) \wedge (err = SUCCESS \longrightarrow x1a \cup vars\ spec' \subseteq \mathcal{V}')))$› **for** *x1c x1a*
using *fully-unsorted-poly-rel-vars-subset-vars-llist*[*OF spec*]
by (*force simp*: *code-status-status-rel-def is-failed-def unfold-code-status unfold-status is-cfailed-def var-rel-set-rel-alt-def*
intro!: *RES-refine*
intro:)
have [*refine*]: ‹$(b,b') \in \{((\mathcal{V}, A), \mathcal{V}', A').\ ((\mathcal{V}, A), \mathcal{V}', A') \in \langle var\text{-}rel\rangle set\text{-}rel \times_r fmap\text{-}polys\text{-}rel2\ c\ \mathcal{V}\}$ $\Longrightarrow$
$(spec, spec') \in$ *sorted-poly-rel O mset-poly-rel* $\Longrightarrow$
$(c, c') \in$ *code-status-status-rel* $\Longrightarrow$
PAC-checker-l spec b c st
$\leq \Downarrow \{((err, \mathcal{V}, A), err', \mathcal{V}', A')$.
$((err, \mathcal{V}, A), err', \mathcal{V}', A') \in$ *code-status-status-rel* $\times_r$ *vars-rel2 err* $\times_r$ *fmap-polys-rel2 err* $\mathcal{V}\}$
(*PAC-checker spec' b' c' st'*)› **for** *spec b c spec' b' c'*
using *assms* **apply** −
apply (*rule order-trans*)
apply (*rule ref-two-step*[*OF PAC-checker-l-PAC-checker*])
apply *assumption+*
apply (*rule order-refl*)
apply (*rule conc-fun-R-mono*)
apply (*auto simp*: *vars-rel2-def*)
done
have *still-in*: ‹$(spec'a, spec) \in \{(xs, ys).\ (xs, ys) \in sorted\text{-}poly\text{-}rel \wedge vars\text{-}llist\ xs \subseteq spec0\}$ *O mset-poly-rel* $\Longrightarrow$
$(x, x') \in$ *?A spec'* $\Longrightarrow$
$x2 = (x1a, x2a) \Longrightarrow$
$x' = (x1, x2) \Longrightarrow$

```
    x2b = (x1c, x2c) ⟹
    x = (x1b, x2b) ⟹
    ¬ is-cfailed x1b ⟹
    ¬ is-failed x1 ⟹
    RETURN x1c
    ≤ ⇓ (⟨var-rel⟩set-rel) (SPEC ((⊆) (x1a ∪ vars spec′)))›
    for spec′a spec x x′ x1 x2 x1a x2a x1b x2b x1c x2c spec0
    apply (auto intro!: RETURN-RES-refine exI[of - x1a])
    done


  show ?thesis
    unfolding full-checker-def full-checker-l-def
    apply (refine-rcg remap-polys-l-remap-polys full-normalize-poly-normalize-poly-spec-vars2
      assms)
    subgoal by auto
    subgoal by auto
    subgoal unfolding remap-polys-l-with-err-pre-def by auto
    subgoal by (auto simp: is-cfailed-def is-failed-def)
    subgoal by (auto simp: vars-rel2-def fmap-polys-rel2-def)
    apply (rule still-in; assumption)
    subgoal by auto
    subgoal by auto
    subgoal by (auto simp: fmap-polys-rel2-def vars-rel2-def)
    done
qed

lemma full-checker-l-full-checker′:
  ‹(uncurry2 full-checker-l, uncurry2 full-checker) ∈
  ((fully-unsorted-poly-rel O mset-poly-rel) ×r unsorted-fmap-polys-rel) ×r ⟨epac-step-rel⟩list-rel →f
    ⟨{((err, 𝒱, A), err′, 𝒱′, A′).
    ((err, 𝒱, A), err′, 𝒱′, A′)
    ∈ code-status-status-rel ×r  vars-rel2 err ×r {(xs, ys).
      (¬ is-cfailed err ⟶ (xs, ys) ∈ ⟨nat-rel, sorted-poly-rel O mset-poly-rel⟩fmap-rel ∧
        (∀ i∈#dom-m xs. vars-llist (the (fmlookup xs i)) ⊆ 𝒱))}}⟩nres-rel›
  apply (intro frefI nres-relI)
  using full-checker-l-full-checker unfolding fmap-polys-rel2-def by force

end

end

theory EPAC-Checker-Init
  imports EPAC-Checker PAC-Checker.WB-Sort PAC-Checker.PAC-Checker-Relation
begin
```

3 Initial Normalisation of Polynomials

3.1 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.

end

theory *EPAC-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.

```
local-setup ‹
  let
    val version =
      trim-line (#1 (Isabelle-System.bash-output (cd $ISAFOL/ && git rev−parse −−short HEAD ||
echo unknown)))
  in
    Local-Theory.define
      ((binding ‹version›, NoSyn),
        ((binding ‹version-def›, []), HOLogic.mk-literal version)) #> #2
  end
›
```

declare *version-def* [*code*]

end
theory *EPAC-Steps-Refine*
  imports *EPAC-Checker*
begin

lemma *is-CL-import*[*sepref-fr-rules*]:
  assumes ‹*CONSTRAINT is-pure K*› ‹*CONSTRAINT is-pure V*› ‹*CONSTRAINT is-pure R*›
  shows
    ‹$(return\ o\ pac\text{-}res,\ RETURN\ o\ pac\text{-}res) \in [\lambda x.\ is\text{-}Extension\ x \lor is\text{-}CL\ x]_a\ (pac\text{-}step\text{-}rel\text{-}assn\ K\ V\ R)^k \to V$›
    ‹$(return\ o\ pac\text{-}src1,\ RETURN\ o\ pac\text{-}src1) \in [\lambda x.\ is\text{-}Del\ x]_a\ (pac\text{-}step\text{-}rel\text{-}assn\ K\ V\ R)^k \to K$›
    ‹$(return\ o\ new\text{-}id,\ RETURN\ o\ new\text{-}id) \in [\lambda x.\ is\text{-}Extension\ x \lor is\text{-}CL\ x]_a\ (pac\text{-}step\text{-}rel\text{-}assn\ K\ V\ R)^k \to K$›
    ‹$(return\ o\ is\text{-}CL,\ RETURN\ o\ is\text{-}CL) \in (pac\text{-}step\text{-}rel\text{-}assn\ K\ V\ R)^k \to_a bool\text{-}assn$›
    ‹$(return\ o\ is\text{-}Del,\ RETURN\ o\ is\text{-}Del) \in (pac\text{-}step\text{-}rel\text{-}assn\ K\ V\ R)^k \to_a bool\text{-}assn$›
    ‹$(return\ o\ new\text{-}var,\ RETURN\ o\ new\text{-}var) \in [\lambda x.\ is\text{-}Extension\ x]_a\ (pac\text{-}step\text{-}rel\text{-}assn\ K\ V\ R)^k \to R$›
    ‹$(return\ o\ is\text{-}Extension,\ RETURN\ o\ is\text{-}Extension) \in (pac\text{-}step\text{-}rel\text{-}assn\ K\ V\ R)^k \to_a bool\text{-}assn$›
  using *assms*
  by (*sepref-to-hoare*; *sep-auto simp*: *pac-step-rel-assn-alt-def is-pure-conv ent-true-drop pure-app-eq*
    split: *pac-step.splits*; *fail*)+

lemma *is-CL-import2*[*sepref-fr-rules*]:
  assumes ‹*CONSTRAINT is-pure K*› ‹*CONSTRAINT is-pure V*›
  shows
    ‹$(return\ o\ pac\text{-}srcs,\ RETURN\ o\ pac\text{-}srcs) \in [\lambda x.\ is\text{-}CL\ x]_a\ (pac\text{-}step\text{-}rel\text{-}assn\ K\ V\ R)^k \to list\text{-}assn\ (V \times_a K)$›
  using *assms*
  by (*sepref-to-hoare*; *sep-auto simp*: *pac-step-rel-assn-alt-def is-pure-conv ent-true-drop pure-app-eq*
    assms[*simplified*] *list-assn-pure-conv*
    split: *pac-step.splits*)

lemma *is-Mult-lastI*:
  ‹$\neg\ is\text{-}CL\ b \Longrightarrow \neg is\text{-}Extension\ b \Longrightarrow is\text{-}Del\ b$›

```
  by (cases b) auto

end

theory EPAC-Checker-Synthesis
  imports EPAC-Checker EPAC-Version
    EPAC-Checker-Init
    EPAC-Steps-Refine
    PAC-Checker.More-Loops
    PAC-Checker.WB-Sort PAC-Checker.PAC-Checker-Relation
    PAC-Checker.PAC-Checker-Synthesis
begin
hide-fact (open) PAC-Checker.PAC-checker-l-def
hide-const (open) PAC-Checker.PAC-checker-l
```

4 Code Synthesis of the Complete Checker

```
definition check-linear-combi-l-pre-err-impl :: ‹uint64 ⇒ bool ⇒ bool ⇒ bool ⇒ string› where
  ‹check-linear-combi-l-pre-err-impl i adom emptyl ivars =
  ''Precondition for '%' failed '' @ show (nat-of-uint64 i) @
  ''(already in domain: '' @ show adom @
  ''; empty CL'' @ show emptyl @
  ''; new vars: '' @ show ivars @ '')''›

abbreviation comp4 (infixl oooo 55) where f oooo g ≡ λx. f ooo (g x)

lemma [sepref-fr-rules]:
  ‹(uncurry3 (return oooo check-linear-combi-l-pre-err-impl),
    uncurry3 check-linear-combi-l-pre-err) ∈ uint64-nat-assn^k *a bool-assn^k *a bool-assn^k *a bool-assn^k
→a raw-string-assn›
  unfolding list-assn-pure-conv check-linear-combi-l-pre-err-impl-def
    check-linear-combi-l-pre-err-def
  apply sepref-to-hoare
  apply sep-auto
  done

definition check-linear-combi-l-dom-err-impl :: ‹ - ⇒ uint64 ⇒ string› where
  ‹check-linear-combi-l-dom-err-impl xs i =
  ''Invalid polynomial '' @ show (nat-of-uint64 i)›

lemma [sepref-fr-rules]:
  ‹(uncurry (return oo (check-linear-combi-l-dom-err-impl)),
  uncurry (check-linear-combi-l-dom-err)) ∈ poly-assn^k *a uint64-nat-assn^k →a raw-string-assn›
  unfolding list-assn-pure-conv check-linear-combi-l-dom-err-impl-def
    check-linear-combi-l-dom-err-def
  apply sepref-to-hoare
  apply sep-auto
  done

definition check-linear-combi-l-mult-err-impl :: ‹ - ⇒ - ⇒ string› where
  ‹check-linear-combi-l-mult-err-impl xs ys =
  ''Invalid calculation, found'' @ show xs @ '' instead of '' @ show ys›

lemma [sepref-fr-rules]:
  ‹(uncurry (return oo check-linear-combi-l-mult-err-impl),
```

uncurry check-linear-combi-l-mult-err) ∈ *poly-assn*^{k} $*_a$ *poly-assn*^{k} $\rightarrow_a$ *raw-string-assn*›
unfolding *list-assn-pure-conv check-linear-combi-l-mult-err-impl-def*
check-linear-combi-l-mult-err-def
apply *sepref-to-hoare*
apply *sep-auto*
done

sepref-definition *linear-combi-l-impl*
is ‹*uncurry3 linear-combi-l*›
:: ‹*uint64-nat-assn*^{k} $*_a$ *polys-assn*^{k} $*_a$ *vars-assn*^{k} $*_a$ (*list-assn* (*poly-assn* $\times_a$ *uint64-nat-assn*))^k $\rightarrow_a$ *poly-assn* $\times_a$ (*list-assn* (*poly-assn* $\times_a$ *uint64-nat-assn*)) $\times_a$ *status-assn raw-string-assn*›
supply [[*goals-limit=1*]]
unfolding *linear-combi-l-def check-linear-combi-l-def conv-to-is-Nil*
term-order-rel′-def[*symmetric*]
term-order-rel′-alt-def
in-dom-m-lookup-iff
fmlookup′-def[*symmetric*]
vars-llist-alt-def is-Nil-def
unfolding
HOL-list.fold-custom-empty
apply (*rewrite* **in** ‹(*op-HOL-list-empty*, -)› *annotate-assn*[**where** *A*=‹*poly-assn*›])
by *sepref*

definition *has-failed* :: ‹*bool nres*› **where**
‹*has-failed* = *RES UNIV*›

lemma [*sepref-fr-rules*]:
‹(*uncurry0* (*return False*), *uncurry0 has-failed*)∈*unit-assn*^{k} $\rightarrow_a$ *bool-assn*›
by *sepref-to-hoare*
(*sep-auto simp*: *has-failed-def*)

declare *linear-combi-l-impl.refine*[*sepref-fr-rules*]
sepref-register *check-linear-combi-l-pre-err*
sepref-definition *check-linear-combi-l-impl*
is ‹*uncurry5 check-linear-combi-l*›
:: ‹*poly-assn*^{k} $*_a$ *polys-assn*^{k} $*_a$ *vars-assn*^{k} $*_a$ *uint64-nat-assn*^{k} $*_a$
(*list-assn* (*poly-assn* $\times_a$ *uint64-nat-assn*))^k $*_a$ *poly-assn*^{k} $\rightarrow_a$ *status-assn raw-string-assn*›
supply [[*goals-limit=1*]]
unfolding *check-mult-l-def check-linear-combi-l-def conv-to-is-Nil*
term-order-rel′-def[*symmetric*]
term-order-rel′-alt-def
in-dom-m-lookup-iff
fmlookup′-def[*symmetric*]
vars-llist-alt-def is-Nil-def
has-failed-def[*symmetric*]
by *sepref*

declare *check-linear-combi-l-impl.refine*[*sepref-fr-rules*]

sepref-register *is-cfailed is-Del*

definition *PAC-checker-l-step′* :: - **where**
‹*PAC-checker-l-step′ a b c d* = *PAC-checker-l-step a* (*b*, *c*, *d*)›

lemma *PAC-checker-l-step-alt-def*:

‹*PAC-checker-l-step a bcd e* = (*let* (*b*,*c*,*d*) = *bcd in PAC-checker-l-step′ a b c d e*)›
unfolding *PAC-checker-l-step′-def* **by** *auto*

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*
find-theorems *is-CL RETURN*

sepref-register *check-linear-combi-l check-extension-l2*
term *check-extension-l2*

definition *check-extension-l2-cond* :: ‹*nat* ⇒ -› **where**
‹*check-extension-l2-cond i A* $\mathcal{V}$ *v* = *SPEC* (λ*b*. *b* ⟶ *fmlookup′ i A* = *None* ∧ *v* ∉ $\mathcal{V}$)›

definition *check-extension-l2-cond2* :: ‹*nat* ⇒ -› **where**
‹*check-extension-l2-cond2 i A* $\mathcal{V}$ *v* = *RETURN* (*fmlookup′ i A* = *None* ∧ *v* ∉ $\mathcal{V}$)›

sepref-definition *check-extension-l2-cond2-impl*
is ‹*uncurry3 check-extension-l2-cond2*›
:: ‹*uint64-nat-assn*^{k} $*_a$ *polys-assn*^{k} $*_a$ *vars-assn*^{k} $*_a$ *string-assn*^{k} $\rightarrow_a$ *bool-assn*›
supply [[*goals-limit=1*]]
unfolding *check-extension-l2-cond2-def*
in-dom-m-lookup-iff
fmlookup′-def[*symmetric*]
not-not is-None-def
by *sepref*

lemma *check-extension-l2-cond2-check-extension-l2-cond*:
‹(*uncurry3 check-extension-l2-cond2*, *uncurry3 check-extension-l2-cond*) ∈
(((*nat-rel* $\times_r$ *Id*) $\times_r$ *Id*) $\times_r$ *Id*) $\rightarrow_f$ ⟨*bool-rel*⟩*nres-rel*›
by (*auto intro*!: *RES-refine nres-relI frefI*
simp: *check-extension-l2-cond-def check-extension-l2-cond2-def*)

lemmas [*sepref-fr-rules*] =
check-extension-l2-cond2-impl.refine[*FCOMP check-extension-l2-cond2-check-extension-l2-cond*]

definition *check-extension-l-side-cond-err-impl* :: ‹- ⇒ -› **where**
‹*check-extension-l-side-cond-err-impl v r s* =
″*Error while checking side conditions of extensions polynow*, *var is* ″ @ *show v* @
″*side condition p*∗*p* − *p* = ″ @ *show s* @ ″ *and should be 0*″›
term *check-extension-l-side-cond-err*
lemma [*sepref-fr-rules*]:
‹(*uncurry2* (*return ooo* (*check-extension-l-side-cond-err-impl*)),
uncurry2 (*check-extension-l-side-cond-err*)) ∈ *string-assn*^{k} $*_a$ *poly-assn*^{k} $*_a$ *poly-assn*^{k} $\rightarrow_a$ *raw-string-assn*›
unfolding *check-extension-l-side-cond-err-impl-def check-extension-l-side-cond-err-def*
list-assn-pure-conv
apply *sepref-to-hoare*
apply *sep-auto*
done

definition *check-extension-l-new-var-multiple-err-impl* :: ‹- ⇒ -› **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 [*sepref-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} $\rightarrow_a$ *raw-string-assn*⟩
unfolding *check-extension-l-new-var-multiple-err-impl-def*
check-extension-l-new-var-multiple-err-def
list-assn-pure-conv
apply *sepref-to-hoare*
apply *sep-auto*
done

sepref-definition *check-extension-l-impl*
is ⟨*uncurry5 check-extension-l2*⟩
:: ⟨*poly-assn*^{k} $*_a$ *polys-assn*^{k} $*_a$ *vars-assn*^{k} $*_a$ *uint64-nat-assn*^{k} $*_a$
string-assn^k $*_a$ *poly-assn*^{k} $\rightarrow_a$ *status-assn raw-string-assn*⟩
supply [[*goals-limit=1*]]
unfolding *check-extension-l2-def*
in-dom-m-lookup-iff
fmlookup'-def[*symmetric*]
not-not is-None-def
uminus-poly-def[*symmetric*]
HOL-list.fold-custom-empty
check-extension-l2-cond-def[*symmetric*]
vars-llist-alt-def
by *sepref*

lemmas [*sepref-fr-rules*] =
check-extension-l-impl.refine

lemma *is-Mult-lastI*:
⟨¬ *is-CL b* ⟹ ¬*is-Extension b* ⟹ *is-Del b*⟩
by (*cases b*) *auto*

sepref-definition *check-step-impl*
is ⟨*uncurry4 PAC-checker-l-step'*⟩
:: ⟨*poly-assn*^{k} $*_a$ (*status-assn raw-string-assn*)^d $*_a$ *vars-assn*^{d} $*_a$ *polys-assn*^{d} $*_a$ (*pac-step-rel-assn* (*uint64-nat-assn*) *poly-assn* (*string-assn* :: *string* ⇒ -))^d $\rightarrow_a$
status-assn raw-string-assn $\times_a$ *vars-assn* $\times_a$ *polys-assn*⟩
supply [[*goals-limit=1*]] *is-Mult-lastI*[*intro*] *single-valued-uint64-nat-rel*[*simp*]
unfolding *PAC-checker-l-step-def PAC-checker-l-step'-def*
pac-step.case-eq-if Let-def
is-success-alt-def[*symmetric*]
uminus-poly-def[*symmetric*]
HOL-list.fold-custom-empty
by *sepref*

declare *check-step-impl.refine*[*sepref-fr-rules*]

sepref-register *PAC-checker-l-step PAC-checker-l-step' fully-normalize-poly-impl*

definition *PAC-checker-l'* **where**

‹PAC-checker-l′ p 𝒱 A status steps = PAC-checker-l p (𝒱, A) status steps›

lemma *PAC-checker-l-alt-def*:
‹PAC-checker-l p 𝒱A status steps =
(let (𝒱, A) = 𝒱A in PAC-checker-l′ p 𝒱 A status steps)›
unfolding *PAC-checker-l′-def* **by** *auto*

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)›
apply (*intro ext*)
apply (*case-tac x*; *case-tac xa*)
apply *simp-all*
apply (*simp-all add*: *relAPP-def p2rel-def pure-def*)
unfolding *pure-def*[*symmetric*] *list-assn-pure-conv*
apply (*auto simp*: *pure-def relAPP-def*)
done

lemma *safe-epac-step-rel-assn*[*safe-constraint-rules*]:
‹CONSTRAINT is-pure K ⟹ CONSTRAINT is-pure V ⟹ CONSTRAINT is-pure R ⟹
CONSTRAINT is-pure (EPAC-Checker.pac-step-rel-assn K V R)›
by (*auto simp*: *step-rewrite-pure*(*1*)[*symmetric*] *is-pure-conv*)

sepref-definition *PAC-checker-l-impl*
is ‹uncurry4 PAC-checker-l′›
:: ‹$poly\text{-}assn^k *_a vars\text{-}assn^d *_a polys\text{-}assn^d *_a (status\text{-}assn\ raw\text{-}string\text{-}assn)^d *_a$
$(list\text{-}assn\ (pac\text{-}step\text{-}rel\text{-}assn\ (uint64\text{-}nat\text{-}assn)\ poly\text{-}assn\ string\text{-}assn))^k \rightarrow_a$
$status\text{-}assn\ raw\text{-}string\text{-}assn \times_a vars\text{-}assn \times_a polys\text{-}assn$›
supply [[*goals-limit=1*]] *is-Mult-lastI*[*intro*]
unfolding *PAC-checker-l-def is-success-alt-def*[*symmetric*] *PAC-checker-l-step-alt-def*
nres-bind-let-law[*symmetric*] *PAC-checker-l′-def*
conv-to-is-Nil is-Nil-def
apply (*subst nres-bind-let-law*)
by *sepref*

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)) ∈ $unit\text{-}assn^k \rightarrow_a raw\text{-}string\text{-}assn$›
unfolding *remap-polys-l-dom-err-def*
remap-polys-l-dom-err-def
list-assn-pure-conv
by *sepref-to-hoare sep-auto*

MLton is not able to optimise the calls to pow.

lemma *pow-2-64*: ‹(*2*::*nat*) ^ *64* = *18446744073709551616*›
  by *auto*

sepref-register *upper-bound-on-dom op-fmap-empty*

definition *full-checker-l2*
  :: ‹*llist-polynomial* ⇒ (*nat*, *llist-polynomial*) *fmap* ⇒ (-, *string*, *nat*) *pac-step list* ⇒
    (*string code-status* × -) *nres*›
where
  ‹*full-checker-l2 spec A st* = *do* {
    spec′ ← *full-normalize-poly spec*;
    (*b*, 𝒱, *A*) ← *remap-polys-l spec* {} *A*;
    if is-cfailed b
    then RETURN (*b*, 𝒱, *A*)
    else do {
      PAC-checker-l spec′ (𝒱, *A*) *b st*
    }
  }›

sepref-register *remap-polys-l*
find-theorems *full-checker-l2*
sepref-definition *full-checker-l-impl*
  is ‹*uncurry2 full-checker-l2*›
  :: ‹$\textit{poly-assn}^k *_a \textit{polys-assn-input}^d *_a (\textit{list-assn}\ (\textit{pac-step-rel-assn}\ (\textit{uint64-nat-assn})\ \textit{poly-assn string-assn}))^k \rightarrow_a$
    $\textit{status-assn raw-string-assn} \times_a \textit{vars-assn} \times_a \textit{polys-assn}$›
  supply [[*goals-limit=1*]] *is-Mult-lastI*[*intro*]
  unfolding *full-checker-l-def hs.fold-custom-empty*
    union-vars-poly-alt-def[*symmetric*]
    PAC-checker-l-alt-def
    full-checker-l2-def
  by *sepref*

sepref-definition *PAC-empty-impl*
  is ‹*uncurry0* (*RETURN fmempty*)›
  :: ‹$\textit{unit-assn}^k \rightarrow_a \textit{polys-assn-input}$›
  unfolding *op-iam-fmap-empty-def*[*symmetric*] *pat-fmap-empty*
  by *sepref*

sepref-definition *empty-vars-impl*
  is ‹*uncurry0* (*RETURN* {})›
  :: ‹$\textit{unit-assn}^k \rightarrow_a \textit{vars-assn}$›
  unfolding *hs.fold-custom-empty*
  by *sepref*

end
theory *EPAC-Perfectly-Shared*
  imports *EPAC-Checker-Specification*
    PAC-Checker.PAC-Checker
    EPAC-Checker
begin

We now introduce sharing of variables to make a more efficient representation possible.

5 Perfectly sharing of elements

5.1 Definition

type-synonym (*'nat*, *'string*) *shared-vars* = ‹*'string multiset* × (*'nat*, *'string*) *fmap* × (*'string*, *'nat*) *fmap*›

definition *perfectly-shared-vars*
:: ‹*'string multiset* ⇒ (*'nat*, *'string*) *shared-vars* ⇒ *bool*›
where
‹*perfectly-shared-vars* $\mathcal{V}$ = (λ($\mathcal{D}$, *V*, *V'*).
set-mset (*dom-m V'*) = *set-mset* $\mathcal{V}$ ∧ $\mathcal{D}$ = $\mathcal{V}$ ∧
(∀ *i* ∈#*dom-m V*. *fmlookup V'* (*the* (*fmlookup V i*)) = *Some i*) ∧
(∀ *str* ∈#*dom-m V'*. *fmlookup V* (*the* (*fmlookup V' str*)) = *Some str*) ∧
(∀ *i j*. *i*∈#*dom-m V* ⟶ *j*∈#*dom-m V* ⟶ (*fmlookup V i* = *fmlookup V j* ⟷ *i* = *j*)))›

abbreviation *fmlookup-direct* :: ‹(*'a*, *'b*) *fmap* ⇒ *'a* ⇒ *'b*› (**infix** ∝ *70*) **where**
‹*fmlookup-direct A b* ≡ *the* (*fmlookup A b*)›

lemma *perfectly-shared-vars-simps*:
assumes ‹*perfectly-shared-vars* $\mathcal{V}$ (*VV'*)›
shows ‹*str* ∈# $\mathcal{V}$ ⟷ *str* ∈# *dom-m* (*snd* (*snd VV'*))›
using *assms*
unfolding *perfectly-shared-vars-def*
apply *auto*
done

lemma *perfectly-shared-add-new-var*:
fixes *V* :: ‹(*'nat*, *'string*) *fmap*› **and**
v :: ‹*'string*›
assumes ‹*perfectly-shared-vars* $\mathcal{V}$ (*D*, *V*, *V'*)› **and**
‹*v* ∉# $\mathcal{V}$› **and**
k-notin[*simp*]: ‹*k* ∉# *dom-m V*›
shows ‹*perfectly-shared-vars* (*add-mset v* $\mathcal{V}$) (*add-mset v D*, *fmupd k v V*, *fmupd v k V'*)›
proof −
have
DV[*simp*]: ‹*D* = $\mathcal{V}$› **and**
V'$\mathcal{V}$: ‹*set-mset* (*dom-m V'*) = *set-mset* $\mathcal{V}$› **and**
map: ‹⋀*i*. *i* ∈#*dom-m V* ⟹ *fmlookup V'* (*the* (*fmlookup V i*)) = *Some i*› **and**
map-str: ‹⋀*str*. *str* ∈#*dom-m V'* ⟹ *fmlookup V* (*the* (*fmlookup V' str*)) = *Some str*› **and**
perfect: ‹⋀*i j*. *i*∈#*dom-m V* ⟹ *j*∈#*dom-m V* ⟹ *fmlookup V i* = *fmlookup V j* ⟷ *i* = *j*›
using *assms* **unfolding** *perfectly-shared-vars-def*
by *auto*
have *v-notin*[*simp*]: ‹*v* ∉# *dom-m V'*›
using *V'*$\mathcal{V}$ *assms*(*2*) **by** *blast*
show *?thesis*
unfolding *perfectly-shared-vars-def prod.simps*
proof (*intro conjI allI ballI impI*)
show ‹*add-mset v D* = *add-mset v* $\mathcal{V}$›
using *DV* **by** *auto*
show ‹*set-mset* (*dom-m* (*fmupd v k V'*)) = *set-mset* (*add-mset v* $\mathcal{V}$)›
using *V'*$\mathcal{V}$ *in-remove1-mset-neq* **by** *fastforce*

show ‹*fmlookup* (*fmupd v k V'*) (*fmupd k v V* ∝ *i*) = *Some i*›
if ‹*i* ∈# *dom-m* (*fmupd k v V*)›
for *i*

using *map*[*of i*] *that v-notin*
by (*auto dest*!: *indom-mI simp del*: *v-notin*)

show ‹*fmlookup* (*fmupd k v V*) (*fmupd v k V′* ∝ *str*) = *Some str*›
if ‹*str* ∈# *dom-m* (*fmupd v k V′*)›
for *str*
using *map-str*[*of str*] *that k-notin*
by (*auto dest*!: *indom-mI simp del*: *k-notin*)
show ‹(*fmlookup* (*fmupd k v V*) *i* = *fmlookup* (*fmupd k v V*) *j*) = (*i* = *j*)›
if ‹*i* ∈# *dom-m* (*fmupd k v V*)› **and**
‹*j* ∈# *dom-m* (*fmupd k v V*)›
for *i j*
using *perfect*[*of i j*] *that*
using *indom-mI*[*of V i*] *map*[*of i*] *indom-mI*[*of V j*] *map*[*of j*] *indom-mI*[*of V′ v*]
apply (*auto simp*: *eq-commute*[*of* ‹*Some -*› ‹*fmlookup V -*›])
done
qed
qed

lemma *perfectly-shared-vars-remove-update*:
assumes ‹*perfectly-shared-vars* (*add-mset v* 𝒱) (*D*, *V*, *V′*)› **and**
‹*v* ∉# 𝒱›
shows ‹*perfectly-shared-vars* 𝒱 (*remove1-mset v D*, *fmdrop* (*V′* ∝ *v*) *V*, *fmdrop v V′*)›
using *assms*
unfolding *perfectly-shared-vars-def*
by (*fastforce simp*: *distinct-mset-dom distinct-mset-remove1-All*)

6 Refinement

datatype *memory-allocation* =
Allocated | *alloc-failed*: *Mem-Out*

type-synonym (*′nat*, *′string*) *vars* = ‹*′string multiset*›

definition *perfectly-shared-var-rel* :: ‹(*′nat*,*′string*) *shared-vars* ⇒ (*′nat* × *′string*) *set*› **where**
‹*perfectly-shared-var-rel* = (λ(𝒟, 𝒱, 𝒱′). *br* (λ*i*. 𝒱 ∝ *i*) (λ*i*. *i* ∈# *dom-m* 𝒱))›

definition *perfectly-shared-vars-rel* :: ‹((*′nat*,*′string*) *shared-vars* × (*′nat*, *′string*) *vars*) *set*›
where
‹*perfectly-shared-vars-rel* = {(𝒜, 𝒱). *perfectly-shared-vars* 𝒱 𝒜}›

definition *find-new-idx* :: ‹(*′nat*,*′string*) *shared-vars* ⇒ -› **where**
‹*find-new-idx* = (λ(-, 𝒱, -). *SPEC* (λ(*mem*, *k*). ¬ *alloc-failed mem* ⟶ *k* ∉# *dom-m* 𝒱))›

definition *import-variableS*
:: ‹*′string* ⇒ (*′nat*, *′string*) *shared-vars* ⇒
(*memory-allocation* × (*′nat*, *′string*) *shared-vars* × *′nat*) *nres*›
where
‹*import-variableS v* = (λ(𝒟, 𝒱, 𝒱′). *do* {
(*mem*, *k*) ← *find-new-idx* (𝒟, 𝒱, 𝒱′);
if alloc-failed mem then do {*k* ← *RES* (*UNIV* :: *′nat set*); *RETURN* (*mem*, (𝒟, 𝒱, 𝒱′), *k*)}
else RETURN (*Allocated*, (*add-mset v* 𝒟, *fmupd k v* 𝒱, *fmupd v k* 𝒱′), *k*)
})›

definition *import-variable*

:: ‹'string ⇒ ('nat, 'string) vars ⇒ (memory-allocation × ('nat, 'string) vars × 'string) nres›
where
‹*import-variable v* = (λ𝒱. *do* {
  ASSERT(*v* ∉# 𝒱);
  SPEC(λ(*mem*, 𝒱', *k*::'*string*). ¬*alloc-failed mem* ⟶ 𝒱' = *add-mset k* 𝒱 ∧ *k* = *v*)
})›

definition *is-new-variableS* :: ‹'string ⇒ ('nat, 'string) shared-vars ⇒ bool nres› **where**
‹*is-new-variableS v* = (λ(𝒟, 𝒱, 𝒱').
RETURN (*v* ∉# *dom-m* 𝒱')
)›

definition *is-new-variable* :: ‹'string ⇒ ('nat, 'string) vars ⇒ bool nres› **where**
‹*is-new-variable v* = (λ𝒱'.
  RETURN (*v* ∉# 𝒱')
)›

lemma *import-variableS-import-variable*:
fixes 𝒱 :: ‹('nat, 'string) vars›
assumes ‹(𝒜, 𝒱) ∈ *perfectly-shared-vars-rel*› **and** ‹(*v*, *v'*) ∈ *Id*›
shows ‹*import-variableS v* 𝒜 ≤ ⇓({((*mem*, 𝒜', *i*), (*mem'*, 𝒱', *j*)). *mem* = *mem'* ∧
  (𝒜', 𝒱') ∈ *perfectly-shared-vars-rel* ∧
  (¬*alloc-failed mem'* ⟶ (*i*, *j*) ∈ *perfectly-shared-var-rel* 𝒜') ∧
  (∀ *xs*. *xs* ∈ *perfectly-shared-var-rel* 𝒜 ⟶ *xs* ∈ *perfectly-shared-var-rel* 𝒜')})
 (*import-variable v'* 𝒱)›
using *assms*
unfolding *import-variableS-def import-variable-def find-new-idx-def*
by (*refine-vcg lhs-step-If*)
 (*auto intro*!: *RETURN-RES-refine simp*: *perfectly-shared-add-new-var perfectly-shared-vars-rel-def*
 perfectly-shared-var-rel-def br-def)

lemma *is-new-variable-spec*:
assumes ‹(𝒜, 𝒟𝒱) ∈ *perfectly-shared-vars-rel*› ‹(*v*,*v'*) ∈ *Id*›
shows ‹*is-new-variableS v* 𝒜 ≤ ⇓*bool-rel* (*is-new-variable v'* 𝒟𝒱)›
using *assms*
unfolding *is-new-variable-def is-new-variableS-def*
by (*auto simp*: *perfectly-shared-vars-rel-def*
 perfectly-shared-vars-simps split: *prod.splits*)

definition *import-variables*
:: ‹'string list ⇒ ('nat, 'string) vars ⇒ (memory-allocation × ('nat, 'string) vars) nres›
where
‹*import-variables vs* 𝒱 = *do* {
(*mem*, 𝒱, -, -) ← $WHILE_T$(λ(*mem*, 𝒱, *vs*, -). ¬*alloc-failed mem* ∧ *vs* ≠ [])
(λ(-, 𝒱, *vs*, *vs'*). *do* {
 ASSERT(*vs* ≠ []);
 let v = *hd vs*;
 a ← *is-new-variable v* 𝒱;
 if ¬*a then RETURN* (*Allocated* ,𝒱, *tl vs*, *vs'* @ [*v*])
 else do {
  (*mem*, 𝒱, -) ← *import-variable v* 𝒱;
  RETURN(*mem*, 𝒱, *tl vs*, *vs'* @ [*v*])
 }
 })
  (*Allocated*, 𝒱, *vs*, []);

RETURN (*mem*, $\mathcal{V}$)
}›

definition *import-variablesS*
:: ‹′*string list* ⇒ (′*nat*, ′*string*) *shared-vars* ⇒ (*memory-allocation* × (′*nat*, ′*string*) *shared-vars*) *nres*›
where
‹*import-variablesS vs* $\mathcal{V}$ = *do* {
(*mem*, $\mathcal{V}$, -) ← *WHILE*$_T$(λ(*mem*, $\mathcal{V}$, *vs*). ¬*alloc-failed mem* ∧ *vs* ≠ [])
(λ(-, $\mathcal{V}$, *vs*). *do* {
ASSERT(*vs* ≠ []);
let v = *hd vs*;
a ← *is-new-variableS v* $\mathcal{V}$;
if ¬*a then RETURN* (*Allocated* ,$\mathcal{V}$, *tl vs*)
else do {
(*mem*, $\mathcal{V}$, -) ← *import-variableS v* $\mathcal{V}$;
RETURN(*mem*, $\mathcal{V}$, *tl vs*)
}
})
(*Allocated*, $\mathcal{V}$, *vs*);
RETURN (*mem*, $\mathcal{V}$)
}›

lemma *import-variables-spec*:
‹*import-variables vs* $\mathcal{V}$ ≤ ⇓*Id* (*SPEC*(λ(*mem*, $\mathcal{V}'$). ¬*alloc-failed mem* ⟶ *set-mset* $\mathcal{V}'$ = *set-mset* $\mathcal{V}$ ∪ *set vs*))›
proof −
define *I* **where**
‹*I* ≡ (λ(*mem*, $\mathcal{V}'$, *vs*′, *vs*′′).
(¬*alloc-failed mem* ⟶ (*vs* = *vs*′′ @ *vs*′) ∧ *set-mset* $\mathcal{V}'$ = *set-mset* $\mathcal{V}$ ∪ *set vs*′′))›
show *?thesis*
unfolding *import-variables-def is-new-variable-def*
apply (*refine-vcg WHILET-rule*[**where** *I* = ‹*I*› **and**
R = ‹*measure* (λ(*mem*, $\mathcal{V}'$, *vs*′, -). (*if* ¬*alloc-failed mem then 1 else 0*) + *length vs*′)›]
is-new-variable-spec)
subgoal by *auto*
subgoal unfolding *I-def* **by** *auto*
subgoal by *auto*
subgoal for *s a b aa ba ab bb*
unfolding *I-def* **by** *auto*
subgoal for *s a b aa ba ab bb*
by *auto*
subgoal
by (*clarsimp simp*: *neq-Nil-conv import-variable-def I-def*)
subgoal
by (*auto simp*: *I-def*)
done
qed

lemma *import-variablesS-import-variables*:
assumes ‹($\mathcal{V}$, $\mathcal{V}'$) ∈ *perfectly-shared-vars-rel*› **and**
‹(*vs*, *vs*′) ∈ *Id*›
shows ‹*import-variablesS vs* $\mathcal{V}$ ≤ ⇓{(*a*,*b*). (*a*,*b*)∈*Id* $\times_r$ *perfectly-shared-vars-rel* ∧
(¬*alloc-failed* (*fst a*) ⟶ *perfectly-shared-var-rel* $\mathcal{V}$ ⊆ *perfectly-shared-var-rel* (*snd a*))} (*import-variables vs*′ $\mathcal{V}'$)›

proof −
  show *?thesis*
    unfolding *import-variablesS-def import-variables-def*
    apply (*refine-rcg WHILET-refine*[**where** $R = \langle\{((mem, \mathcal{VV}, vs), (mem', \mathcal{V}', vs', \text{-})).$
      $(mem, mem') \in Id \wedge (\mathcal{VV}, \mathcal{V}') \in perfectly\text{-}shared\text{-}vars\text{-}rel \wedge (vs, vs') \in Id \wedge$
     $(\neg alloc\text{-}failed\ mem \longrightarrow perfectly\text{-}shared\text{-}var\text{-}rel\ \mathcal{V} \subseteq perfectly\text{-}shared\text{-}var\text{-}rel\ \mathcal{VV})\}\rangle]$
      is-new-variable-spec import-variableS-import-variable)
    subgoal using *assms* **by** *auto*
    subgoal by *auto*
    subgoal by *auto*
    subgoal by *auto*
    subgoal by *auto*
    subgoal by *auto*
    subgoal by *auto*
    subgoal by *auto*
    subgoal by *auto*
    subgoal by *auto*
    subgoal by *auto*
    done
qed

definition *get-var-name* :: ‹('*nat*, '*string*) *vars* ⇒ '*string* ⇒ '*string nres*› **where**
  ‹*get-var-name* $\mathcal{V}$ *x* = *do* {
    ASSERT($x \in\# \mathcal{V}$);
    RETURN x
  }›

definition *get-var-posS* :: ‹('*nat*, '*string*) *shared-vars* ⇒ '*string* ⇒ '*nat nres*› **where**
  ‹*get-var-posS* $\mathcal{V}$ *x* = *do* {
    ASSERT($x \in\# dom\text{-}m\ (snd\ (snd\ \mathcal{V}))$);
    RETURN $(snd\ (snd\ \mathcal{V}) \propto x)$
  }›

definition *get-var-nameS* :: ‹('*nat*, '*string*) *shared-vars* ⇒ '*nat* ⇒ '*string nres*› **where**
  ‹*get-var-nameS* $\mathcal{V}$ *x* = *do* {
    ASSERT($x \in\# dom\text{-}m\ (fst\ (snd\ \mathcal{V}))$);
    RETURN $(fst\ (snd\ \mathcal{V}) \propto x)$
  }›

lemma *get-var-posS-spec*:
  fixes $\mathcal{DV}$ :: ‹('*nat*, '*string*) *vars*› **and**
    $\mathcal{A}$ :: ‹('*nat*, '*string*) *shared-vars*› **and**
    x :: '*string*
  assumes ‹$(\mathcal{A}, \mathcal{DV}) \in perfectly\text{-}shared\text{-}vars\text{-}rel$› **and**
    ‹$(x,x') \in Id$›
  shows ‹$get\text{-}var\text{-}posS\ \mathcal{A}\ x \leq \Downarrow(perfectly\text{-}shared\text{-}var\text{-}rel\ \mathcal{A})\ (get\text{-}var\text{-}name\ \mathcal{DV}\ x')$›
  using *assms* **unfolding** *get-var-posS-def get-var-name-def*
  apply *refine-vcg*
  apply (*auto simp*: *perfectly-shared-var-rel-def*
    perfectly-shared-vars-rel-def perfectly-shared-vars-simps br-def
    intro!: *ASSERT-leI*)
  apply (*simp-all add*: *perfectly-shared-vars-def in-dom-m-lookup-iff*)
  done

abbreviation *perfectly-shared-monom*

:: ‹(′nat,′string) shared-vars ⇒ (′nat list × ′string list) set›
where
‹perfectly-shared-monom 𝒱 ≡ ⟨perfectly-shared-var-rel 𝒱⟩list-rel ›

definition *import-monom-no-newS*
:: ‹(′nat, ′string) shared-vars ⇒ ′string list ⇒ (bool × ′nat list) nres›
where
‹import-monom-no-newS 𝒜 xs = do {
(new, -, xs) ← $WHILE_T$ (λ(new, xs, -). ¬new ∧ xs ≠ [])
(λ(-, xs, ys). do {
ASSERT(xs ≠ []);
let x = hd xs;
b ← is-new-variableS x 𝒜;
if b
then RETURN (True, tl xs, ys)
else do {
x ← get-var-posS 𝒜 x;
RETURN (False, tl xs, x # ys)
}
})
(False, xs, []);
RETURN (new, rev xs)
}›

definition *import-monom-no-new*
:: ‹(′nat, ′string) vars ⇒ ′string list ⇒ (bool × ′string list) nres›
where
‹import-monom-no-new 𝒜 xs = do {
(new, -, xs) ← $WHILE_T$ (λ(new, xs, -). ¬new ∧ xs ≠ [])
(λ(-, xs, ys). do {
ASSERT(xs ≠ []);
let x = hd xs;
b ← is-new-variable x 𝒜;
if b
then RETURN (True, tl xs, ys)
else do {
x ← get-var-name 𝒜 x;
RETURN (False, tl xs, ys @ [x])
}
})
(False, xs, []);
RETURN (new, xs)
}›

lemma *import-monom-no-new-spec*:
shows ‹import-monom-no-new 𝒜 xs ≤ ⇓ Id
(SPEC(λ(new, ys). (new ⟷ ¬set xs ⊆ set-mset 𝒜) ∧
(¬new ⟶ ys = xs)))›
unfolding *import-monom-no-new-def is-new-variable-def get-var-name-def*
apply (*refine-vcg*
WHILET-rule[**where** I = ‹(λ(new, ys, zs). (¬new ⟶ xs = zs @ ys) ∧ (¬new ⟶ set zs ⊆ set-mset 𝒜) ∧
(new ⟶ ¬set xs ⊆ set-mset 𝒜))› **and**
R = ‹measure (λ(-, ys, -). length ys)›])
subgoal by *auto*

```
  subgoal by auto
  subgoal by auto
  subgoal by auto
  subgoal by auto
  subgoal by auto
  subgoal by (auto simp: neq-Nil-conv)
  subgoal by (auto simp: neq-Nil-conv)
  subgoal by auto
  subgoal by auto
  subgoal by auto
  subgoal by auto
  subgoal by auto
  subgoal by auto
  subgoal by auto
  subgoal by auto
  done

lemma import-monom-no-newS-import-monom-no-new:
  assumes ‹(𝒜, 𝒱𝒟) ∈ perfectly-shared-vars-rel› ‹(xs, xs') ∈ Id›
  shows ‹import-monom-no-newS 𝒜 xs ≤ ⇓(bool-rel ×r perfectly-shared-monom 𝒜)
    (import-monom-no-new 𝒱𝒟 xs')›
  using assms
  unfolding import-monom-no-new-def import-monom-no-newS-def
  apply (refine-rcg WHILET-refine[where R = ‹bool-rel ×r ⟨Id⟩list-rel ×r {(as, bs). (rev as, bs) ∈
perfectly-shared-monom 𝒜}›]
    is-new-variable-spec get-var-posS-spec)
  subgoal by auto
  subgoal
    by auto
  subgoal
    by auto
  subgoal by auto
  subgoal by auto
  subgoal by auto
  subgoal by auto
  subgoal by (force simp: list-rel-append1)
  subgoal by auto
  done

definition import-poly-no-newS
  :: ‹('nat, 'string) shared-vars ⇒ ('string list × 'a) list ⇒ (bool × ('nat list × 'a)list) nres›
where
  ‹import-poly-no-newS 𝒜 xs = do {
  (new, -, xs) ← WHILE_T (λ(new, xs, -). ¬new ∧ xs ≠ [])
    (λ(-, xs, ys). do {
      ASSERT(xs ≠ []);
      let (x, n) = hd xs;
      (b, x) ← import-monom-no-newS 𝒜 x;
      if b
      then RETURN (True, tl xs, ys)
      else do {
        RETURN (False, tl xs, (x, n) # ys)
      }
    })
    (False, xs, []);
```

RETURN (*new*, *rev xs*)
}›

definition *import-poly-no-new*
:: ‹('*nat*, '*string*) *vars* ⇒ ('*string list* × '*a*) *list* ⇒ (*bool* × ('*string list* × '*a*) *list*) *nres*›
where
‹*import-poly-no-new* $\mathcal{A}$ *xs* = *do* {
(*new*, -, *xs*) ← $WHILE_T$ (λ(*new*, *xs*, -). ¬*new* ∧ *xs* ≠ [])
(λ(-, *xs*, *ys*). *do* {
ASSERT(*xs* ≠ []);
let (*x*, *n*) = *hd xs*;
(*b*, *x*) ← *import-monom-no-new* $\mathcal{A}$ *x*;
if b
then RETURN (*True*, *tl xs*, *ys*)
else do {
RETURN (*False*, *tl xs*, *ys* @ [(*x*, *n*)])
}
})
(*False*, *xs*, []);
RETURN (*new*, *xs*)
}›

lemma *import-poly-no-newS-import-poly-no-new*:
assumes ‹($\mathcal{A}$, $\mathcal{VD}$) ∈ *perfectly-shared-vars-rel*› ‹(*xs*, *xs'*) ∈ *Id*›
shows ‹*import-poly-no-newS* $\mathcal{A}$ *xs* ≤ ⇓(*bool-rel* $\times_r$ ⟨*perfectly-shared-monom* $\mathcal{A}$ $\times_r$ *Id*⟩*list-rel*)
(*import-poly-no-new* $\mathcal{VD}$ *xs'*)›
using *assms*
unfolding *import-poly-no-new-def import-poly-no-newS-def*
apply (*refine-rcg WHILET-refine*[**where**
R = ‹*bool-rel* $\times_r$ ⟨*Id*⟩*list-rel* $\times_r$ {(*as*, *bs*). (*rev as*, *bs*) ∈ ⟨*perfectly-shared-monom* $\mathcal{A}$ $\times_r$ *Id*⟩*list-rel*}›]
import-monom-no-newS-import-monom-no-new)
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by (*force simp*: *list-rel-append1*)
subgoal by *auto*
done

lemma *import-poly-no-new-spec*:
shows ‹*import-poly-no-new* $\mathcal{A}$ *xs* ≤ ⇓ *Id*
(*SPEC*(λ(*new*, *ys*). ¬*new* ⟶ *ys* = *xs* ∧ *vars-llist xs* ⊆ *set-mset* $\mathcal{A}$))›
proof −
define *I* **where**
[*simp*]: ‹*I* = (λ(*new*, *ys*, *zs*). ¬*new* ⟶ (*xs* = *zs* @ *ys* ∧ *vars-llist zs* ⊆ *set-mset* $\mathcal{A}$))›
show *?thesis*
unfolding *import-poly-no-new-def is-new-variable-def get-var-name-def import-variable-def*
apply (*refine-vcg import-monom-no-new-spec*[*THEN order-trans*]
WHILET-rule[**where** *I* = ‹*I*› **and**
R = ‹*measure* (λ(*mem*, *ys*, -). (*if mem then 0 else 1*) + *length ys*)›])
subgoal by *auto*
subgoal by *auto*

subgoal by *auto*
subgoal by (*auto simp*: *neq-Nil-conv*)
subgoal by *auto*
subgoal by *auto*
done
qed

definition *import-monomS*
:: ‹(′*nat*, ′*string*) *shared-vars* ⇒ ′*string list* ⇒ (- × ′*nat list* × (′*nat*, ′*string*) *shared-vars*) *nres*›
where
‹*import-monomS* $\mathcal{A}$ *xs* = *do* {
(*new*, -, *xs*, $\mathcal{A}$) ← $WHILE_T$ (λ(*mem*, *xs*, -, -). ¬*alloc-failed mem* ∧ *xs* ≠ [])
(λ(-, *xs*, *ys*, $\mathcal{A}$). *do* {
ASSERT(*xs* ≠ []);
let x = *hd xs*;
b ← *is-new-variableS x* $\mathcal{A}$;
if b
then do {
(*mem*, $\mathcal{A}$, *x*) ← *import-variableS x* $\mathcal{A}$;
if alloc-failed mem
then RETURN (*mem*, *xs*, *ys*, $\mathcal{A}$)
else RETURN (*mem*, *tl xs*, *x* # *ys*, $\mathcal{A}$)
}
else do {
x ← *get-var-posS* $\mathcal{A}$ *x*;
RETURN (*Allocated*, *tl xs*, *x* # *ys*, $\mathcal{A}$)
}
})
(*Allocated*, *xs*, [], $\mathcal{A}$);
RETURN (*new*, *rev xs*, $\mathcal{A}$)
}›

definition *import-monom*
:: ‹(′*nat*, ′*string*) *vars* ⇒ ′*string list* ⇒ (*memory-allocation* × ′*string list* × (′*nat*, ′*string*) *vars*) *nres*›
where
‹*import-monom* $\mathcal{A}$ *xs* = *do* {
(*new*, -, *xs*, $\mathcal{A}$) ← $WHILE_T$ (λ(*new*, *xs*, -, -). ¬*alloc-failed new* ∧ *xs* ≠ [])
(λ(*mem*, *xs*, *ys*, $\mathcal{A}$). *do* {
ASSERT(*xs* ≠ []);
let x = *hd xs*;
b ← *is-new-variable x* $\mathcal{A}$;
if b
then do {
(*mem*, $\mathcal{A}$, *x*) ← *import-variable x* $\mathcal{A}$;
if alloc-failed mem
then RETURN (*mem*, *xs*, *ys*, $\mathcal{A}$)
else RETURN (*mem*, *tl xs*, *ys* @ [*x*], $\mathcal{A}$)
}
else do {
x ← *get-var-name* $\mathcal{A}$ *x*;
RETURN (*mem*, *tl xs*, *ys* @ [*x*], $\mathcal{A}$)
}
})
(*Allocated*, *xs*, [], $\mathcal{A}$);
RETURN (*new*, *xs*, $\mathcal{A}$)

}⟩

lemma *import-monom-spec*:
shows ⟨*import-monom* $\mathcal{A}$ *xs* ≤ ⇓ *Id*
(*SPEC*(λ(*new*, *ys*, $\mathcal{A}'$). ¬*alloc-failed new* ⟶ *ys* = *xs* ∧ *set-mset* $\mathcal{A}'$ = *set-mset* $\mathcal{A}$ ∪ *set xs*))⟩
proof −
define *I* **where**
[*simp*]: ⟨*I* = (λ(*new*, *ys*, *zs*, $\mathcal{A}'$). ¬*alloc-failed new* ⟶ (*xs* = *zs* @ *ys* ∧ *set-mset* $\mathcal{A}'$ = *set-mset* $\mathcal{A}$ ∪ *set zs*))⟩
show *?thesis*
unfolding *import-monom-def is-new-variable-def get-var-name-def import-variable-def*
apply (*refine-vcg*
WHILET-rule[**where** *I* = ⟨*I*⟩ **and**
R = ⟨*measure* (λ(*mem*, *ys*, -). (*if alloc-failed mem then 0 else 1*) + *length ys*)⟩])
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by (*auto simp*: *neq-Nil-conv*)
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
done
qed

definition *import-polyS*
:: ⟨(′*nat*, ′*string*) *shared-vars* ⇒ (′*string list* × ′*a*) *list* ⇒
(*memory-allocation* × (′*nat list* × ′*a*)*list* × (′*nat*, ′*string*) *shared-vars*) *nres*⟩
where
⟨*import-polyS* $\mathcal{A}$ *xs* = *do* {
(*mem*,-, *xs*, $\mathcal{A}$) ← $WHILE_T$ (λ(*mem*, *xs*, -, -). ¬*alloc-failed mem* ∧ *xs* ≠ [])
(λ(*mem*, *xs*, *ys*, $\mathcal{A}$). *do* {
ASSERT(*xs* ≠ []);
let (*x*, *n*) = *hd xs*;
(*mem*, *x*, $\mathcal{A}$) ← *import-monomS* $\mathcal{A}$ *x*;
if alloc-failed mem
then RETURN (*mem*, *xs*, *ys*, $\mathcal{A}$)
else do {
RETURN (*mem*, *tl xs*, (*x*, *n*) # *ys*, $\mathcal{A}$)
}
})
(*Allocated*, *xs*, [], $\mathcal{A}$);
RETURN (*mem*, *rev xs*, $\mathcal{A}$)
}⟩

definition *import-poly*
:: ⟨(′*nat*, ′*string*) *vars* ⇒ (′*string list* × ′*a*) *list* ⇒
(*memory-allocation* × (′*string list* × ′*a*) *list* × (′*nat*, ′*string*)*vars*) *nres*⟩
where
⟨*import-poly* $\mathcal{A}$ *xs0* = *do* {
(*new*, -, *xs*, $\mathcal{A}$) ← $WHILE_T$ (λ(*new*, *xs*, -). ¬*alloc-failed new* ∧ *xs* ≠ [])

(λ(-, *xs*, *ys*, 𝒜). *do* {
ASSERT(*xs* ≠ []);
let (*x*, *n*) = *hd xs*;
(*b*, *x*, 𝒜) ← *import-monom* 𝒜 *x*;
if alloc-failed b
then RETURN (*b*, *xs*, *ys*, 𝒜)
else do {
RETURN (*Allocated*, *tl xs*, *ys* @ [(*x*, *n*)], 𝒜)
}
})
(*Allocated*, *xs0*, [], 𝒜);
ASSERT(¬*alloc-failed new* ⟶ *xs0* = *xs*);
RETURN (*new*, *xs*, 𝒜)
}›

lemma *import-poly-spec*:
fixes 𝒜 :: ‹(′*nat*, ′*string*) *vars*›
shows ‹*import-poly* 𝒜 *xs* ≤ ⇓ *Id*
(*SPEC*(λ(*new*, *ys*, 𝒜′). ¬*alloc-failed new* ⟶ *ys* = *xs* ∧ *set-mset* 𝒜′ = *set-mset* 𝒜 ∪ ⋃(*set* ‘*fst* ‘ *set xs*)))›
proof −
define *I* **where**
[*simp*]: ‹*I* = (λ(*new*, *ys*, *zs*, 𝒜′). ¬*alloc-failed new* ⟶ (*xs* = *zs* @ *ys* ∧
set-mset 𝒜′ = *set-mset* 𝒜 ∪ ⋃(*set* ‘ *fst* ‘ *set zs*)))›
show *?thesis*
unfolding *import-poly-def is-new-variable-def get-var-name-def import-variable-def*
apply (*refine-vcg import-monom-spec*[*THEN order-trans*]
WHILET-rule[**where** *I* = ‹*I*› **and**
R = ‹*measure* (λ(*mem*, *ys*, -). (*if alloc-failed mem then 0 else 1*) + *length ys*)›])
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by (*auto simp*: *neq-Nil-conv*)
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
done
qed

lemma *list-rel-append-single*: ‹(*xs*, *ys*) ∈ ⟨*R*⟩*list-rel* ⟹ (*x*, *y*) ∈ *R* ⟹ (*xs* @ [*x*], *ys* @ [*y*]) ∈ ⟨*R*⟩*list-rel*›
by (*meson list-rel-append1 list-rel-simp*(*4*) *refine-list*(*1*))

lemma *list-rel-mono*: ‹*A* ∈ ⟨*R*⟩*list-rel* ⟹ (⋀*xs*. *xs* ∈ *R* ⟹ *xs* ∈ *R*′) ⟹ *A* ∈ ⟨*R*′⟩*list-rel*›
unfolding *list-rel-def*
apply (*cases A*)
by (*simp add*: *list-all2-mono*)

lemma *import-monomS-import-monom*:
fixes 𝒱𝒟 :: ‹(′*nat*, ′*string*) *vars*› **and** $\mathcal{A}_0$ :: ‹(′*nat*, ′*string*)*shared-vars*› **and** *xs xs*′ :: ‹′*string list*›
assumes ‹($\mathcal{A}_0$, 𝒱𝒟) ∈ *perfectly-shared-vars-rel*› ‹(*xs*, *xs*′) ∈ ⟨*Id*⟩*list-rel*›
shows ‹*import-monomS* $\mathcal{A}_0$ *xs* ≤ ⇓ {((*mem*, xs_0, 𝒜), (*mem*′, ys_0, 𝒜′)). *mem* = *mem*′ ∧
(𝒜, 𝒜′) ∈ *perfectly-shared-vars-rel* ∧ (¬*alloc-failed mem* ⟶ (xs_0, ys_0) ∈ *perfectly-shared-monom* 𝒜)∧
(¬*alloc-failed mem* ⟶ (∀ *xs*. *xs* ∈ *perfectly-shared-monom* $\mathcal{A}_0$ ⟶ *xs* ∈ *perfectly-shared-monom* 𝒜))}
(*import-monom* 𝒱𝒟 *xs*′)›

using *assms*
unfolding *import-monom-def import-monomS-def*
apply (*refine-rcg WHILET-refine*[**where**
$R = \langle\{((mem::memory\text{-}allocation,\ xs_0::'string\ list,\ zs_0::'nat\ list,\ \mathcal{A} :: ('nat,\ 'string)shared\text{-}vars),$
$(mem',\ ys_0::'string\ list,\ zs_0'::'string\ list,\ \mathcal{A}' :: ('nat,\ 'string)vars)).\ mem = mem' \wedge$
$(\mathcal{A},\ \mathcal{A}') \in perfectly\text{-}shared\text{-}vars\text{-}rel \wedge (\neg alloc\text{-}failed\ mem \longrightarrow (rev\ zs_0,\ zs_0') \in perfectly\text{-}shared\text{-}monom$
$\mathcal{A}) \wedge$
$(xs_0,\ ys_0) \in \langle Id\rangle list\text{-}rel \wedge$
$(\neg alloc\text{-}failed\ mem \longrightarrow (\forall xs.\ xs \in perfectly\text{-}shared\text{-}monom\ \mathcal{A}_0 \longrightarrow xs \in perfectly\text{-}shared\text{-}monom$
$\mathcal{A}))\}\rangle$]
import-variableS-import-variable
is-new-variable-spec get-var-posS-spec)
subgoal by *auto*
subgoal
by *auto*
subgoal
by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal apply (*auto intro*!: *list-rel-append-single intro*: *list-rel-mono*)
by (*metis* (*full-types*) *list-rel-mono surj-pair*)
subgoal by *auto*
subgoal by *auto*
subgoal using *memory-allocation.exhaust-disc* **by** (*auto intro*!: *list-rel-append-single intro*: *list-rel-mono*)
subgoal by *auto*
done

abbreviation *perfectly-shared-polynom*
:: $\langle('nat,'string)\ shared\text{-}vars \Rightarrow (('nat\ list \times int)\ list \times ('string\ list \times int)\ list)\ set\rangle$
where
$\langle perfectly\text{-}shared\text{-}polynom\ \mathcal{V} \equiv \langle perfectly\text{-}shared\text{-}monom\ \mathcal{V} \times_r int\text{-}rel\rangle list\text{-}rel\rangle$

abbreviation *import-poly-rel* :: ‹-› **where**
$\langle import\text{-}poly\text{-}rel\ \mathcal{A}_0\ xs' \equiv$
$\{((mem,\ xs_0,\ \mathcal{A}),\ (mem',\ ys_0,\ \mathcal{A}')).\ mem = mem' \wedge$
$(\neg alloc\text{-}failed\ mem \longrightarrow (\mathcal{A},\ \mathcal{A}') \in perfectly\text{-}shared\text{-}vars\text{-}rel \wedge\ ys_0 = xs' \wedge (xs_0,\ ys_0) \in perfectly\text{-}shared\text{-}polynom$
$\mathcal{A}) \wedge$
$(\neg alloc\text{-}failed\ mem \longrightarrow perfectly\text{-}shared\text{-}polynom\ \mathcal{A}_0 \subseteq perfectly\text{-}shared\text{-}polynom\ \mathcal{A})\}\rangle$

lemma *import-polyS-import-poly*:
assumes $\langle(\mathcal{A}_0,\ \mathcal{VD}) \in perfectly\text{-}shared\text{-}vars\text{-}rel\rangle$ $\langle(xs,\ xs') \in \langle\langle Id\rangle list\text{-}rel \times_r Id\rangle list\text{-}rel\rangle$
shows $\langle import\text{-}polyS\ \mathcal{A}_0\ xs \le \Downarrow(import\text{-}poly\text{-}rel\ \mathcal{A}_0\ xs)$
$(import\text{-}poly\ \mathcal{VD}\ xs')\rangle$
using *assms*
unfolding *import-poly-def import-polyS-def*
apply (*refine-rcg WHILET-refine*[**where**
$R = \langle\{((mem,\ zs,\ xs_0,\ \mathcal{A}),\ (mem',\ zs',\ ys_0,\ \mathcal{A}')).\ mem = mem' \wedge$
$(\mathcal{A},\ \mathcal{A}') \in perfectly\text{-}shared\text{-}vars\text{-}rel \wedge (zs,\ zs') \in \langle\langle Id\rangle list\text{-}rel \times_r Id\rangle list\text{-}rel$
$\wedge (\neg alloc\text{-}failed\ mem \longrightarrow (rev\ xs_0,\ ys_0) \in perfectly\text{-}shared\text{-}polynom\ \mathcal{A}) \wedge$
$(\neg alloc\text{-}failed\ mem \longrightarrow perfectly\text{-}shared\text{-}polynom\ \mathcal{A}_0 \subseteq perfectly\text{-}shared\text{-}polynom\ \mathcal{A})\}\rangle$]

import-monomS-import-monom)
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by (*auto simp*: *list-rel-append1*)
subgoal for *x x′ x1 x2 x1a x2a x1b x2b x1c x2c x1d x2d x1e x2e x1f x2f x1g x2g xa x′a x1h x2h x1i x2i x1j x2j x1k x2k*
using *memory-allocation.exhaust-disc*[*of x1h* ‹*x1h* = *Allocated*›]
by (*auto intro*!: *list-rel-append-single intro*: *list-rel-mono*)
subgoal by *auto*
done

definition *drop-content* :: ‹′*string* ⇒ (′*nat*, ′*string*) *vars* ⇒ (′*nat*, ′*string*) *vars nres*›
where
‹*drop-content* = (λ*v* 𝒱′. *do* {
ASSERT(*v* ∈# 𝒱′);
RETURN (*remove1-mset v* 𝒱′)
})›

definition *drop-contentS* :: ‹′*string* ⇒ (′*nat*, ′*string*) *shared-vars* ⇒ (′*nat*, ′*string*) *shared-vars nres*›
where
‹*drop-contentS* = (λ*v* (𝒟, 𝒱, 𝒱′). *do* {
ASSERT(*v* ∈# *dom-m* 𝒱′);
if count 𝒟 *v* = *1*
then do {
let i = 𝒱′ ∝ *v*;
RETURN (*remove1-mset v* 𝒟, *fmdrop i* 𝒱, *fmdrop v* 𝒱′)
}
else
RETURN (*remove1-mset v* 𝒟, 𝒱, 𝒱′)
})›

lemma *drop-contentS-drop-content*:
assumes ‹(𝒜, 𝒱𝒟) ∈ *perfectly-shared-vars-rel*› ‹(*v*, *v*′) ∈ *Id*›
shows ‹*drop-contentS v* 𝒜 ≤ ⇓*perfectly-shared-vars-rel* (*drop-content v*′ 𝒱𝒟)›
proof −
have [*simp*]: ‹*count xs x* = *1* ⟹ *y* ∈# *remove1-mset x xs* ⟷ *y* ∈# *xs* ∧ *x* ≠*y*› **for** *x xs y*
by (*auto simp add*: *in-diff-count*)
have [*simp*]: ‹*count xs x* ≠ *1* ⟹ *x* ∈# *xs* ⟹ *y* ∈# *remove1-mset x xs* ⟷ *y* ∈# *xs*› **for** *x xs y*
by (*metis One-nat-def add-mset-remove-trivial-eq count-add-mset count-inI in-remove1-mset-neq*)
show *?thesis*
using *assms*
unfolding *drop-content-def drop-contentS-def*
apply *refine-vcg*
apply (*auto simp*: *perfectly-shared-vars-rel-def perfectly-shared-vars-def distinct-mset-dom distinct-mset-remove1-All*)
by (*metis option.inject*)
qed

definition *perfectly-shared-strings-equal*
:: ‹(′*nat*, ′*string*) *vars* ⇒ ′*string* ⇒ ′*string* ⇒ *bool nres*›
where
‹*perfectly-shared-strings-equal* $\mathcal{V}$ *x y* = *do* {
ASSERT(*x* ∈# $\mathcal{V}$ ∧ *y* ∈# $\mathcal{V}$);
RETURN (*x* = *y*)
}›

definition *perfectly-shared-strings-equal-l*
:: ‹(′*nat*,′*string*)*shared-vars* ⇒ ′*nat* ⇒ ′*nat* ⇒ *bool nres*›
where
‹*perfectly-shared-strings-equal-l* $\mathcal{V}$ *x y* = *do* {
RETURN (*x* = *y*)
}›

lemma *perfectly-shared-strings-equal-l-perfectly-shared-strings-equal*:
assumes ‹($\mathcal{A}$, $\mathcal{V}$) ∈ *perfectly-shared-vars-rel*› **and**
‹(*x*, *x*′) ∈ *perfectly-shared-var-rel* $\mathcal{A}$› **and**
‹(*y*, *y*′) ∈ *perfectly-shared-var-rel* $\mathcal{A}$›
shows ‹*perfectly-shared-strings-equal-l* $\mathcal{A}$ *x y* ≤ ⇓*bool-rel* (*perfectly-shared-strings-equal* $\mathcal{V}$ *x*′ *y*′)›
using *assms* **unfolding** *perfectly-shared-strings-equal-l-def perfectly-shared-strings-equal-def*
perfectly-shared-vars-rel-def perfectly-shared-var-rel-def br-def
by *refine-rcg*
(*auto simp*: *perfectly-shared-vars-def simp*: *add-mset-eq-add-mset dest*!: *multi-member-split*)

datatype(**in** −) *ordered* = *LESS* | *EQUAL* | *GREATER* | *UNKNOWN*

definition (**in** −)*perfect-shared-var-order* :: ‹(*nat*, *string*)*vars* ⇒ *string* ⇒ *string* ⇒ *ordered nres*› **where**
‹*perfect-shared-var-order* $\mathcal{D}$ *x y* = *do* {
ASSERT(*x* ∈# $\mathcal{D}$ ∧ *y* ∈# $\mathcal{D}$);
eq ← *perfectly-shared-strings-equal* $\mathcal{D}$ *x y*;
if eq then RETURN EQUAL
else do {
x ← *get-var-name* $\mathcal{D}$ *x*;
y ← *get-var-name* $\mathcal{D}$ *y*;
if (*x*, *y*) ∈ *var-order-rel then RETURN* (*LESS*)
else RETURN (*GREATER*)
}
}›

lemma *var-roder-rel-total*:
‹*y* ≠ *ya* ⟹ (*y*, *ya*) ∉ *var-order-rel* ⟹ (*ya*, *y*) ∈ *var-order-rel*›
unfolding *var-order-rel-def*
using *less-than-char-linear lexord-linear* **by** *blast*

lemma *perfect-shared-var-order-spec*:
assumes ‹*xs* ∈# $\mathcal{V}$› ‹*ys* ∈# $\mathcal{V}$›
shows
‹*perfect-shared-var-order* $\mathcal{V}$ *xs ys* ≤ ⇓ *Id* (*SPEC*(λ*b*. ((*b*=*LESS* ⟶ (*xs*, *ys*) ∈ *var-order-rel*) ∧
(*b*=*GREATER* ⟶ (*ys*, *xs*) ∈ *var-order-rel* ∧ ¬(*xs*, *ys*) ∈ *var-order-rel*) ∧
(*b*=*EQUAL* ⟶ *xs* = *ys*)) ∧ *b* ≠ *UNKNOWN*))›
using *assms* **unfolding** *perfect-shared-var-order-def perfectly-shared-strings-equal-def nres-monad3*
get-var-name-def
by *refine-vcg*
(*auto dest*: *var-roder-rel-total*)

definition (**in** −) *perfect-shared-term-order-rel-pre*
:: ‹(*nat*, *string*) *vars* ⇒ *string list* ⇒ *string list* ⇒ *bool*›
where
‹*perfect-shared-term-order-rel-pre* $\mathcal{V}$ *xs ys* ⟷
set xs ⊆ *set-mset* $\mathcal{V}$ ∧ *set ys* ⊆ *set-mset* $\mathcal{V}$›

definition (**in** −) *perfect-shared-term-order-rel*
:: ‹(*nat*, *string*) *vars* ⇒ *string list* ⇒ *string list* ⇒ *ordered nres*›
where
‹*perfect-shared-term-order-rel* $\mathcal{V}$ *xs ys* = *do* {
ASSERT (*perfect-shared-term-order-rel-pre* $\mathcal{V}$ *xs ys*);
(*b*, -, -) ← $WHILE_T$ (λ(*b*, *xs*, *ys*). *b* = *UNKNOWN*)
(λ(*b*, *xs*, *ys*). *do* {
if xs = [] ∧ *ys* = [] *then RETURN* (*EQUAL*, *xs*, *ys*)
else if xs = [] *then RETURN* (*LESS*, *xs*, *ys*)
else if ys = [] *then RETURN* (*GREATER*, *xs*, *ys*)
else do {
ASSERT(*xs* ≠ [] ∧ *ys* ≠ []);
eq ← *perfect-shared-var-order* $\mathcal{V}$ (*hd xs*) (*hd ys*);
if eq = *EQUAL then RETURN* (*b*, *tl xs*, *tl ys*)
else RETURN (*eq*, *xs*, *ys*)
}
}) (*UNKNOWN*, *xs*, *ys*);
RETURN b
}›

lemma (**in** −)*perfect-shared-term-order-rel-spec*:
assumes ‹*set xs* ⊆ *set-mset* $\mathcal{V}$› ‹*set ys* ⊆ *set-mset* $\mathcal{V}$›
shows
‹*perfect-shared-term-order-rel* $\mathcal{V}$ *xs ys* ≤ ⇓ *Id* (*SPEC*(λ*b*. ((*b*=*LESS* ⟶ (*xs*, *ys*) ∈ *term-order-rel*) ∧
(*b*=*GREATER* ⟶ (*ys*, *xs*) ∈ *term-order-rel*) ∧
(*b*=*EQUAL* ⟶ *xs* = *ys*)) ∧ *b* ≠ *UNKNOWN*))› (**is** ‹- ≤ ⇓ - (*SPEC* (λ*b*. *?f b* ∧ *b* ≠ *UNKNOWN*))›)
proof −
define *I* **where**
[*simp*]: ‹*I*= (λ(*b*, *xs0*, *ys0*). *?f b* ∧ (∃ *xs'*. *xs* = *xs'* @ *xs0* ∧ *ys* = *xs'* @ *ys0*))›
show *?thesis*
using *assms*
unfolding *perfect-shared-term-order-rel-def get-var-name-def perfectly-shared-strings-equal-def*
perfectly-shared-strings-equal-def
apply (*refine-vcg WHILET-rule*[**where** *I*= ‹*I*› **and**
R = ‹*measure* (λ(*b*, *xs*, *ys*). *length xs* + (*if b* = *UNKNOWN then 1 else 0*))›]
perfect-shared-var-order-spec[*THEN order-trans*])
subgoal by (*auto simp*: *perfect-shared-term-order-rel-pre-def*)
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by (*auto simp*: *neq-Nil-conv lexord-append-leftI lexord-append-rightI*)
subgoal by *auto*
subgoal by (*auto simp*: *neq-Nil-conv lexord-append-leftI lexord-append-rightI*)
subgoal by *auto*
subgoal by (*auto simp*: *neq-Nil-conv lexord-append-leftI*)

subgoal by (*auto simp*: *neq-Nil-conv*)
subgoal
by ((*subst conc-Id id-apply*)+, *rule SPEC-rule*, *rename-tac x*, *case-tac x*)
(*auto simp*: *neq-Nil-conv intro*: *var-roder-rel-total*
intro!: *lexord-append-leftI lexord-append-rightI*)
subgoal by (*auto simp*: *neq-Nil-conv lexord-append-leftI*)
subgoal by (*auto simp*: *neq-Nil-conv*)
subgoal by (*auto simp*: *neq-Nil-conv*)
subgoal by (*auto simp*: *neq-Nil-conv*)
done
qed

lemma (**in**−) *trans-var-order-rel*[*simp*]: ‹*trans var-order-rel*›
unfolding *trans-def var-order-rel-def*
apply (*intro conjI impI allI*)
by (*meson lexord-partial-trans trans-def trans-less-than-char*)

lemma (**in**−) *term-order-rel-irreflexive*:
‹$(x1f, x1d) \in$ *term-order-rel* $\Longrightarrow (x1d, x1f) \in$ *term-order-rel* $\Longrightarrow x1f = x1d$›
using *lexord-trans*[*of x1f x1d var-order-rel x1f*] *lexord-irreflexive*[*of var-order-rel x1f*]
by *simp*

lemma *get-var-nameS-spec*:
fixes $\mathcal{DV}$ :: ‹(*'nat*, *'string*) *vars*› **and**
$\mathcal{A}$ :: ‹(*'nat*, *'string*) *shared-vars*› **and**
x' :: *'string*
assumes ‹$(\mathcal{A}, \mathcal{DV}) \in$ *perfectly-shared-vars-rel*› **and**
‹$(x,x') \in$ *perfectly-shared-var-rel* $\mathcal{A}$›
shows ‹*get-var-nameS* $\mathcal{A}$ $x \le \Downarrow(Id)$ (*get-var-name* $\mathcal{DV}$ x')›
using *assms* **unfolding** *get-var-nameS-def get-var-name-def*
apply *refine-vcg*
apply (*auto simp*: *perfectly-shared-var-rel-def*
perfectly-shared-vars-rel-def perfectly-shared-vars-simps br-def
intro!: *ASSERT-leI*)
done

lemma *get-var-nameS-spec2*:
fixes $\mathcal{DV}$ :: ‹(*'nat*, *'string*) *vars*› **and**
$\mathcal{A}$ :: ‹(*'nat*, *'string*) *shared-vars*› **and**
x' :: *'string*
assumes ‹$(\mathcal{A}, \mathcal{DV}) \in$ *perfectly-shared-vars-rel*› **and**
‹$(x,x') \in$ *perfectly-shared-var-rel* $\mathcal{A}$›
‹$x' \in\# \mathcal{DV}$›
shows ‹*get-var-nameS* $\mathcal{A}$ $x \le \Downarrow(Id)$ (*RETURN* x')›
apply (*rule get-var-nameS-spec*[*THEN order-trans*, *OF assms*(*1*,*2*)])
apply (*use assms*(*3*) **in** ‹*auto simp*: *get-var-name-def*›)
done

end
theory *EPAC-Efficient-Checker*
imports *EPAC-Checker EPAC-Perfectly-Shared*
begin
hide-const (**open**) *PAC-Checker.full-checker-l*

hide-fact (**open**) *PAC-Checker.full-checker-l-def*

type-synonym *shared-poly* = ‹(*nat list* × *int*) *list*›

definition (**in** −) *add-poly-l′* **where**
‹*add-poly-l′* - = *add-poly-l*›

definition (**in** −)*add-poly-l-prep* :: ‹(*nat*,*string*)*vars* ⇒ *llist-polynomial* × *llist-polynomial* ⇒ *llist-polynomial nres*› **where**
‹*add-poly-l-prep* $\mathcal{D}$ = REC_T
(λ*add-poly-l* (*p*, *q*).
case (*p*,*q*) *of*
(*p*, []) ⇒ *RETURN p*
| ([], *q*) ⇒ *RETURN q*
| ((*xs*, *n*) # *p*, (*ys*, *m*) # *q*) ⇒ *do* {
comp ← *perfect-shared-term-order-rel* $\mathcal{D}$ *xs ys*;
if comp = *EQUAL then if n* + *m* = *0 then add-poly-l* (*p*, *q*)
else do {
pq ← *add-poly-l* (*p*, *q*);
RETURN ((*xs*, *n* + *m*) # *pq*)
}
else if comp = *LESS*
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*)
}
})›

lemma *add-poly-alt-def*[*unfolded conc-Id id-apply*]:
fixes *xs ys* :: *llist-polynomial*
assumes ‹⋃(*set* ‘ (*fst*‘*set xs*)) ⊆ *set-mset* $\mathcal{D}$› ‹⋃(*set* ‘ *fst* ‘ *set ys*) ⊆ *set-mset* $\mathcal{D}$›
shows ‹*add-poly-l-prep* $\mathcal{D}$ (*xs*, *ys*) ≤ ⇓ *Id* (*add-poly-l′* $\mathcal{D}$ (*xs*, *ys*))›
proof −
let *?Rx* = ‹{(*xs′*, *ys′*). (*xs′*, *ys′*) ∈ ⟨*Id*⟩ *list-rel* ∧ (∃ xs_0. *xs* = xs_0 @ *xs′*)}›
let *?Ry* = ‹{(*xs′*, *ys′*). (*xs′*, *ys′*) ∈ ⟨*Id*⟩ *list-rel* ∧ (∃ xs_0. *ys* = xs_0 @ *xs′*)}›
have [*refine0*]: ‹((*xs*, *ys*), *xs*, *ys*) ∈ *?Rx* $\times_r$ *?Ry*›
by *auto*
have *H*: ‹(*x1c*, *x1a*) ∈ ⟨*Id*⟩*list-rel* ⟹ (*x1c*, *x1a*) ∈ ⟨*Id*⟩*list-rel*› **for** *x1c x1a*
by *auto*
have [*intro!*]: ‹*f* ≤ *f′* ⟹ *do* {*a* ← *f*; *P a*} ≤ *do* {*a* ← *f′*; *P a*}› **for** *f f′* :: ‹- *nres*› **and** *P*
unfolding *pw-bind-inres pw-bind-nofail pw-le-iff*
by *blast*
show *?thesis*
using *assms*
unfolding *add-poly-l′-def add-poly-l-def add-poly-l-prep-def*
apply (*refine-vcg perfect-shared-term-order-rel-spec*[*THEN order-trans*])
apply (*rule H*)
subgoal by *auto*
apply (*rule H*)
subgoal by *auto*
subgoal by *auto*

apply (*rule H*)
subgoal by *auto*
subgoal by *auto*
subgoal
apply (*rule specify-left*)
apply (*rule perfect-shared-term-order-rel-spec*[*unfolded conc-Id id-apply*])
subgoal by *auto*
subgoal by *auto*
subgoal premises *p* **for** *comp*
supply [*intro*!] = *p*(*3*)[*unfolded conc-Id id-apply*]
using *p*(*1*,*2*,*4*−)
using *ordered.exhaust*[*of comp False*]
by (*auto simp*: *lexord-irreflexive dest*: *term-order-rel-irreflexive*; *fail*)+
done
done
qed

definition (**in** −)*normalize-poly-shared*
:: ‹(*nat*,*string*) *vars* ⇒ *llist-polynomial* ⇒
(*bool* × *llist-polynomial*) *nres*›
where
‹*normalize-poly-shared* $\mathcal{A}$ *xs* = *do* {
xs ← *full-normalize-poly xs*;
import-poly-no-new $\mathcal{A}$ *xs*
}›

definition *normalize-poly-sharedS*
:: ‹(*nat*,*string*) *shared-vars* ⇒ *llist-polynomial* ⇒
(*bool* × *shared-poly*) *nres*›
where
‹*normalize-poly-sharedS* $\mathcal{A}$ *xs* = *do* {
xs ← *full-normalize-poly xs*;
import-poly-no-newS $\mathcal{A}$ *xs*
}›

definition (**in** −) *mult-monoms-prep* :: ‹(*nat*,*string*)*vars* ⇒ *term-poly-list* ⇒ *term-poly-list* ⇒ *term-poly-list nres*› **where**
‹*mult-monoms-prep* $\mathcal{D}$ *xs ys* = REC_T (λ*f* (*xs*, *ys*).
do {
if xs = [] *then RETURN ys*
else if ys = [] *then RETURN xs*
else do {
ASSERT(*xs* ≠ [] ∧ *ys* ≠ []);
comp ← *perfect-shared-var-order* $\mathcal{D}$ (*hd xs*) (*hd ys*);
if comp = *EQUAL then do* {
pq ← *f* (*tl xs*, *tl ys*);
RETURN (*hd xs* # *pq*)
}
else if comp = *LESS then do* {
pq ← *f* (*tl xs*, *ys*);
RETURN (*hd xs* # *pq*)
}
else do {
pq ← *f* (*xs*, *tl ys*);
RETURN (*hd ys* # *pq*)

}
}
}) (*xs*, *ys*)›

lemma (**in** −) *mult-monoms-prep-mult-monoms*:
assumes ‹*set xs* ⊆ *set-mset* 𝒱› ‹*set ys* ⊆ *set-mset* 𝒱›
shows ‹*mult-monoms-prep* 𝒱 *xs ys* ≤ ⇓*Id* (*SPEC* ((=) (*mult-monoms xs ys*)))›
proof −
have *H*: ‹*f* ≤ *RES p* ⟹ (⋀*x*. *x* ∈ *p* ⟹ (*g x*) ∈ *Q*) ⟹ *do* {*x* ← *f*; *RETURN* (*g x*)} ≤ *RES Q*› **for** *f p g Q*
by (*meson bind-le-nofailI le-RES-nofailI nres-order-simps*(*21*) *order.trans*)
have [*dest*]: ‹ (*x*, *y*) ∈ *var-order-rel* ⟹
(*y*, *x*) ∈ *var-order-rel* ⟹ *x* = *y*› **for** *x y*
by (*meson transE trans-var-order-rel var-order-rel-antisym*)

have [*dest*]: ‹ *xa* ≠ *UNKNOWN* ⟹ *xa* ≠ *GREATER* ⟹ *xa* ≠ *LESS* ⟹ *xa* = *EQUAL* › **for** *xa*
by (*cases xa*) *auto*
show *?thesis*
using *assms*
apply (*induction xs ys rule*:*mult-monoms.induct*)
subgoal
unfolding *mult-monoms-prep-def*
by (*subst RECT-unfold*, *refine-mono*) *auto*
subgoal
unfolding *mult-monoms-prep-def*
by (*subst RECT-unfold*, *refine-mono*) *auto*
subgoal
apply (*subst mult-monoms-prep-def*)
apply (*subst RECT-unfold*, *refine-mono*)
apply (*subst mult-monoms-prep-def*[*symmetric*])+
apply (*simp only*: *prod.simps*)
apply (*refine-vcg perfect-shared-term-order-rel-spec*[*THEN order-trans*]
perfect-shared-var-order-spec[*THEN order-trans*])
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by (*auto intro*!: *H*)
done
done
qed
definition *mult-monoms-prop* :: ‹(*nat*,*string*)*vars*⇒ *llist-polynomial* ⇒ - ⇒ *llist-polynomial* ⇒ *llist-polynomial nres*› **where**
‹*mult-monoms-prop* = (λ𝒱 *qs* (*p*, *m*) *b*. *nfoldli qs* (λ-. *True*) (λ(*q*, *n*) *b*. *do* {*pq* ← *mult-monoms-prep* 𝒱 *p q*; *RETURN* ((*pq*, *m* ∗ *n*) # *b*)}) *b*)›

definition *mult-poly-raw-prop* :: ‹(*nat*,*string*) *vars*⇒ *llist-polynomial* ⇒ *llist-polynomial* ⇒ *llist-polynomial nres*› **where**
‹*mult-poly-raw-prop* 𝒱 *p q* = *nfoldli p* (λ-. *True*) (*mult-monoms-prop* 𝒱 *q*) []›

lemma *mult-monoms-prop-mult-monomials*:
assumes ‹*vars-llist qs* ⊆ *set-mset* 𝒱› ‹*set* (*fst m*) ⊆ *set-mset* 𝒱›
shows ‹*mult-monoms-prop* 𝒱 *qs m b* ≤ ⇓{(*xs*, *ys*). *mset xs* = *mset ys*} (*RES*{*map* (*mult-monomials m*) *qs* @ *b*})›
using *assms*

unfolding *mult-monoms-prop-def*
apply (*cases m*)
apply (*induction qs arbitrary*: *b*)
subgoal by (*auto intro*!: *RETURN-RES-refine*)
subgoal for *a qs aa b ba*
  apply (*cases a*)
  apply (*simp only*: *prod.simps nfoldli-simps*(*2*) *if-True nres-monad3 nres-monad1*)
  apply (*refine-vcg mult-monoms-prep-mult-monoms*[*THEN order-trans*])
  subgoal by *auto*
  subgoal by *auto*
  subgoal premises *p*
    supply [*intro*!] = *p*(*1*)[*THEN order-trans*]
    using *p*(*2*−)
    by (*auto simp*: *conc-fun-RES mult-monomials-def*)
  done
done

lemma *mult-poly-raw-prop-mult-poly-raw*:
  assumes ‹*vars-llist qs* ⊆ *set-mset* 𝒱› ‹*vars-llist ps* ⊆ *set-mset* 𝒱›
  shows ‹*mult-poly-raw-prop* 𝒱 *ps qs* ≤
    (*SPEC* (λ*c*. (*c*, *PAC-Polynomials-Operations.mult-poly-raw ps qs*) ∈ {(*xs*, *ys*). *mset xs* = *mset ys*}))›
proof −
  have [*simp*]: ‹*foldl* (λ*b x*. *map* (*mult-monomials x*) *qs* @ *b*) *b ps* = *foldl* (λ*b x*. *map* (*mult-monomials x*) *qs* @ *b*) [] *ps* @ *b*›
    if ‹*NO-MATCH* [] *b*› **for** *qs ps b*
    apply (*induction ps arbitrary*: *b*)
      apply *simp*
    by (*metis* (*no-types*, *lifting*) *append-assoc foldl-Cons self-append-conv*)

  have *H*: ‹*nfoldli ps* (λ-. *True*) (*mult-monoms-prop* 𝒱 *qs*) *b0*
    ≤ ⇓ {(*xs*, *ys*). *mset xs* = *mset ys*} (*RES* {*foldl* (λ*b x*. *map* (*mult-monomials x*) *qs* @ *b*) *b0 ps*})› **for** *b0*
    using *assms*
    apply (*induction ps arbitrary*: *b0*)
    subgoal by (*auto intro*!: *RETURN-RES-refine*)
    subgoal premises *p*
      supply [*intro*!] = *p*(*1*)[*THEN order-trans*]
      using *p*(*2*−)
      apply (*simp only*: *prod.simps nfoldli-simps*(*2*) *if-True nres-monad3 nres-monad1*)
      apply (*refine-rcg mult-monoms-prop-mult-monomials*)
      apply *auto*
      apply (*rule specify-left*)
      apply (*subst RES-SPEC-eq*[*symmetric*])
      apply (*rule mult-monoms-prop-mult-monomials*[*unfolded conc-fun-RES*])
      apply (*auto simp*: *conc-fun-RES*)
      done
    done

  show *?thesis*
    unfolding *mult-poly-raw-def mult-poly-raw-prop-def*
    by (*rule H*[*THEN order-trans*]) (*auto simp*: *conc-fun-RES*)
qed

definition (**in** −) *mult-poly-full-prop* :: ‹-› **where**
‹*mult-poly-full-prop* 𝒱 *p q* = *do* {
pq ← *mult-poly-raw-prop* 𝒱 *p q*;
ASSERT(*vars-llist pq* ⊆ *vars-llist p* ∪ *vars-llist q*);
normalize-poly pq
}›

lemma *vars-llist-mset-eq*: ‹*mset p* = *mset q* ⟹ *vars-llist p* = *vars-llist q*›
by (*auto simp*: *vars-llist-def dest*!: *mset-eq-setD*)
lemma *mult-poly-full-prop-mult-poly-full*:
assumes ‹*vars-llist qs* ⊆ *set-mset* 𝒱› ‹*vars-llist ps* ⊆ *set-mset* 𝒱›
‹(*ps*, *ps*′) ∈ *Id*› ‹(*qs*, *qs*′) ∈ *Id*›
shows ‹*mult-poly-full-prop* 𝒱 *ps qs* ≤ ⇓*Id* (*mult-poly-full ps*′ *qs*′)›
proof −
have [*refine0*]: ‹*sort-poly-spec p* ≤ ⇓ *Id* (*sort-poly-spec p*′)›
if ‹*mset p*= *mset p*′› **for** *p p*′
using *that*
unfolding *sort-poly-spec-def*
by *auto*
have *H*: ‹*x* ∈ *A* ⟹ *x*= *x*′ ⟹ *x*′ ∈ *A*› **for** *x x*′ *A*
by *auto*
show *?thesis*
using *assms*
unfolding *mult-poly-full-prop-def mult-poly-full-def normalize-poly-def*
apply (*refine-vcg mult-poly-raw-prop-mult-poly-raw*)
apply (*rule H*[*of* - ‹{(*xs*, *ys*). *mset xs* = *mset ys*}›], *assumption*)
subgoal by *auto*
subgoal by (*force dest*: *vars-llist-mset-eq vars-llist-mult-poly-raw*[*THEN set-mp*])
subgoal by *auto*
subgoal by *auto*
done
qed

definition (**in** −) *linear-combi-l-prep2* **where**
‹*linear-combi-l-prep2 i A* 𝒱 *xs* = *do* {
ASSERT(*linear-combi-l-pre i A* (*set-mset* 𝒱) *xs*);
$WHILE_T$
(λ(*p*, *xs*, *err*). *xs* ≠ [] ∧ ¬*is-cfailed err*)
(λ(*p*, *xs*, -). *do* {
ASSERT(*xs* ≠ []);
let (q_0 :: *llist-polynomial*, *i*) = *hd xs*;
if (*i* ∉# *dom-m A* ∨ ¬(*vars-llist* q_0 ⊆ *set-mset* 𝒱))
then do {
err ← *check-linear-combi-l-dom-err* q_0 *i*;
RETURN (*p*, *xs*, *error-msg i err*)
} *else do* {
ASSERT(*fmlookup A i* ≠ *None*);
let r = *the* (*fmlookup A i*);
ASSERT(*vars-llist r* ⊆ *set-mset* 𝒱);
if q_0 = [([],*1*)] *then do* {
pq ← *add-poly-l-prep* 𝒱 (*p*, *r*);
RETURN (*pq*, *tl xs*, *CSUCCESS*)
} *else do* {
(-, *q*) ← *normalize-poly-shared* 𝒱 (q_0);
ASSERT(*vars-llist q* ⊆ *set-mset* 𝒱);

```
          pq ← mult-poly-full-prop 𝒱 q r;
          ASSERT(vars-llist pq ⊆ set-mset 𝒱);
          pq ← add-poly-l-prep 𝒱 (p, pq);
          RETURN (pq, tl xs, CSUCCESS)
        }
      }
    })
    ([], xs, CSUCCESS)
  }›
lemma (in −) import-poly-no-new-spec:
  ‹import-poly-no-new 𝒱 xs ≤ ⇓Id (SPEC(λ(b, xs′). (¬b ⟶ xs = xs′) ∧ (¬b ⟷ vars-llist xs ⊆
set-mset 𝒱)))›
  unfolding import-poly-no-new-def
  apply (refine-vcg WHILET-rule[where I = ‹ λ(b, xs′, ys′). (¬b ⟶ xs = ys′ @ xs′) ∧
    (¬b ⟶ vars-llist ys′ ⊆ set-mset 𝒱) ∧
    (b ⟶ ¬vars-llist xs ⊆ set-mset 𝒱)› and R = ‹measure (λ(b, xs, -). (if b then 0 else 1) + length
xs)›]
    import-monom-no-new-spec[THEN order-trans])
  subgoal by auto
  subgoal by auto
  subgoal by auto
  subgoal by auto
  subgoal by auto
  subgoal by (clarsimp simp add: neq-Nil-conv)
  subgoal by auto
  subgoal by auto
  done

lemma linear-combi-l-prep2-linear-combi-l:
  assumes 𝒱: ‹(𝒱,𝒱′) ∈ {(x, y). y = set-mset x}›‹(i,i′)∈nat-rel›‹(A,A′)∈Id›‹(xs,xs′)∈Id›
  shows ‹linear-combi-l-prep2 i A 𝒱 xs ≤ ⇓Id (linear-combi-l i′ A′ 𝒱′ xs′)›
proof −
  have H1: ‹(if p ∨ q then P else Q) = (if p then P else if q then P else Q)› for p q P Q
    by auto
  have [intro!]: ‹check-linear-combi-l-dom-err x1e x2e ≤ ⇓ Id (check-linear-combi-l-dom-err x1e x2e)›
    for x1e x2e
    by auto
  have linear-combi-l-alt-def:
    ‹linear-combi-l i A 𝒱 xs = do {
      ASSERT(linear-combi-l-pre i A 𝒱 xs);
      WHILE_T
        (λ(p, xs, err). xs ≠ [] ∧ ¬is-cfailed err)
        (λ(p, xs, -). do {
          ASSERT(xs ≠ []);
          ASSERT (vars-llist p ⊆ 𝒱);
          let (q :: llist-polynomial, i) = hd xs;
          if (i ∉# dom-m A ∨ ¬(vars-llist q ⊆ 𝒱))
          then do {
            err ← check-linear-combi-l-dom-err q i;
            RETURN (p, xs, error-msg i err)
            }
          else do {
          ASSERT(fmlookup A i ≠ None);
          let r = the (fmlookup A i);
          ASSERT (vars-llist r ⊆ 𝒱);
```

```
        if q = [([], 1)]
        then do {
          pq ← add-poly-l′ 𝒱′ (p, r);
          RETURN (pq, tl xs, CSUCCESS)
        }
        else  do {
          q ← full-normalize-poly q;
          ASSERT (vars-llist q ⊆ 𝒱);
          let q = q;
          pq ← mult-poly-full q r;
          ASSERT (vars-llist pq ⊆ 𝒱);
          pq ← add-poly-l′ 𝒱′ (p, pq);
          RETURN (pq, tl xs, CSUCCESS)
        }
      }
    })
  ([], xs, CSUCCESS)
  }› for i A 𝒱 xs
  unfolding Let-def linear-combi-l-def add-poly-l′-def by auto
have H: ‹P = Q ⟹ P ≤⇓Id Q› for P Q
  by auto
have [refine0]: ‹⇓ Id (SPEC (λ(b, xs′). (¬ b ⟶ xa = xs′) ∧ (¬ b) = (vars-llist xa ⊆ set-mset 𝒱)))
  ≤ SPEC (λc. (c, q) ∈ {((b, c), d). ¬b ∧ c = d ∧ d = q})›
  if ‹vars-llist xa ⊆ set-mset 𝒱› ‹xa = q›
  for xa 𝒱 q
  using that by auto
show ?thesis
  using assms
  unfolding linear-combi-l-prep2-def linear-combi-l-alt-def normalize-poly-shared-def nres-monad3
  apply (refine-rcg import-poly-no-new-spec[THEN order-trans]
    mult-poly-full-prop-mult-poly-full[THEN order-trans]
    add-poly-alt-def[THEN order-trans])
  subgoal using 𝒱 by auto
  subgoal by auto
  subgoal by auto
  subgoal by auto
  subgoal using 𝒱 by auto
  apply (rule H)
  subgoal by auto
  subgoal by auto
  subgoal by auto
  subgoal by (auto simp: vars-llist-def)
  subgoal by auto
  subgoal by (auto simp: vars-llist-def)
  subgoal by (auto simp: vars-llist-def)
  apply (rule H)
  subgoal by (auto simp: add-poly-l′-def)
  subgoal by auto
  apply (rule H)
  subgoal by auto
  subgoal by auto
  subgoal using 𝒱 by auto
  subgoal by auto
  apply (solves auto)[]
  apply (solves auto)[]
```

apply (*rule H*)
subgoal by *auto*
subgoal using $\mathcal{V}$ **by** (*auto dest*!: *split-list*)
subgoal using $\mathcal{V}$ **by** (*auto dest*!: *split-list*)
subgoal by (*auto simp*: *vars-llist-def*)
apply (*rule H*)
subgoal by (*auto simp*: *add-poly-l'-def*)
subgoal by *auto*
done
qed

definition *check-linear-combi-l-prop* **where**
‹*check-linear-combi-l-prop spec A* $\mathcal{V}$ *i xs r* = *do* {
(*mem-err*, *r*) ← *import-poly-no-new* $\mathcal{V}$ *r*;
if mem-err ∨ *i* ∈# *dom-m A* ∨ *xs* = []
then do {
err ← *check-linear-combi-l-pre-err i* (*i* ∈# *dom-m A*) (*xs* = []) (*mem-err*);
RETURN (*error-msg i err*, *r*)
}
else do {
(*p*, -, *err*) ← *linear-combi-l-prep2 i A* $\mathcal{V}$ *xs*;
if (*is-cfailed err*)
then do {
RETURN (*err*, *r*)
}
else do {
b ← *weak-equality-l p r*;
b' ← *weak-equality-l r spec*;
if b then (*if b' then RETURN* (*CFOUND*, *r*) *else RETURN* (*CSUCCESS*, *r*)) *else do* {
c ← *check-linear-combi-l-mult-err p r*;
RETURN (*error-msg i c*, *r*)
}
}
}}›

lemma *check-linear-combi-l-prop-check-linear-combi-l*:
assumes ‹($\mathcal{V}$,$\mathcal{V}'$) ∈ {(*x*, *y*). *y* = *set-mset x*}› ‹(*A*, *A'*) ∈ *Id*› ‹(*i*,*i'*)∈*nat-rel*› ‹(*xs*,*xs'*)∈*Id*›‹(*r*,*r'*)∈*Id*› ‹(*spec*,*spec'*)∈*Id*›
shows ‹*check-linear-combi-l-prop spec A* $\mathcal{V}$ *i xs r* ≤ ⇓{((*b*,*r'*), *b'*). *b*=*b'* ∧ (¬*is-cfailed b* ⟶ *r*=*r'*)} (*check-linear-combi-l spec' A'* $\mathcal{V}'$ *i' xs' r'*)›
proof −
have [*refine*]: ‹*import-poly-no-new* $\mathcal{V}$ *r* ≤ ⇓ {((*mem*, *r'*), *b*). (*b*=*mem*) ∧ (¬*b* ⟶ *r'*=*r* ∧ *vars-llist r* ⊆ *set-mset* $\mathcal{V}$)} (*RES UNIV*)›
apply (*rule order-trans*)
apply (*rule import-poly-no-new-spec*)
apply (*auto simp*: *conc-fun-RES*)
done
have *H*: ‹*f*=*g* ⟹ *f* ≤ ⇓*Id g*› **for** *f g*
by *auto*

show *?thesis*
using *assms*
unfolding *check-linear-combi-l-prop-def check-linear-combi-l-def*
apply (*refine-vcg linear-combi-l-prep2-linear-combi-l*)
subgoal using *assms* **by** *auto*

apply (*rule H*)
subgoal by (*auto simp*: *check-linear-combi-l-pre-err-def*)
subgoal by (*auto simp*:*error-msg-def*)
subgoal using *assms* **by** *auto*
subgoal by *auto*
apply (*rule H*)
subgoal by *auto*
apply (*rule H*)
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
apply (*rule H*)
subgoal by *auto*
subgoal by *auto*
done
qed

definition (**in** −)*check-extension-l2-prop*
:: ‹- ⇒ - ⇒ *string multiset* ⇒ *nat* ⇒ *string* ⇒ *llist-polynomial* ⇒ (*string code-status* × *llist-polynomial* × *string multiset* × *string*) *nres*›
where
‹*check-extension-l2-prop spec A* 𝒱 *i v p* = *do* {
(*pre*, *nonew*, *mem*, *mem*′, *p*, 𝒱, *v*) ← *do* {
let pre = *i* ∉# *dom-m A* ∧ *v* ∉ *set-mset* 𝒱;
let b = *vars-llist p* ⊆ *set-mset* 𝒱;
(*mem*, *p*, 𝒱) ← *import-poly* 𝒱 *p*;
(*mem*′, 𝒱, *v*) ← *if b* ∧ *pre* ∧ ¬ *alloc-failed mem then import-variable v* 𝒱 *else RETURN* (*mem*, 𝒱, *v*);
RETURN (*pre* ∧ ¬*alloc-failed mem* ∧ ¬ *alloc-failed mem*′, *b*, *mem*, *mem*′, *p*, 𝒱, *v*)
};
if ¬*pre*
then do {
c ← *check-extension-l-dom-err i*;
RETURN (*error-msg i c*, [], 𝒱, *v*)
} *else do* {
if ¬*nonew*
then do {
c ← *check-extension-l-new-var-multiple-err v p*;
RETURN (*error-msg i c*, [], 𝒱, *v*)
}
else do {
ASSERT(*vars-llist p* ⊆ *set-mset* 𝒱);
p2 ← *mult-poly-full-prop* 𝒱 *p p*;
ASSERT(*vars-llist p2* ⊆ *set-mset* 𝒱);
let p′′ = *map* (λ(*a*,*b*). (*a*, −*b*)) *p*;
ASSERT(*vars-llist p*′′ ⊆ *set-mset* 𝒱);
q ← *add-poly-l-prep* 𝒱 (*p2*, *p*′′);
ASSERT(*vars-llist q* ⊆ *set-mset* 𝒱);
eq ← *weak-equality-l q* [];
if eq then do {
RETURN (*CSUCCESS*, *p*, 𝒱, *v*)
} *else do* {
c ← *check-extension-l-side-cond-err v p q*;

RETURN (*error-msg i c*, [], $\mathcal{V}$, *v*)
}
}
}
}›

lemma *check-extension-l2-prop-check-extension-l2*:
assumes ‹($\mathcal{V}$,$\mathcal{V}'$) ∈ {(*x*, *y*). *y* = *set-mset x*}› ‹(*spec*, *spec*′) ∈ *Id*› ‹(*A*, *A*′) ∈ *Id*› ‹(*i*,*i*′) ∈ *nat-rel*› ‹(*v*, *v*′) ∈ *Id*› ‹(*p*, *p*′) ∈ *Id*›
shows ‹*check-extension-l2-prop spec A* $\mathcal{V}$ *i v p* ≤⇓{((*err*, *q*, $\mathcal{A}$, *va*), *b*). (*b* = *err*) ∧ (¬*is-cfailed err* ⟶ *q*=*p* ∧ *v*=*va* ∧ *set-mset* $\mathcal{A}$ = *insert v* $\mathcal{V}'$)}
(*check-extension-l2 spec*′ *A*′ $\mathcal{V}'$ *i*′ *v*′ *p*′)›
proof −

have *G*[*refine*]: ‹*do* {
(*mem*, *pa*, $\mathcal{V}'$) ← *import-poly* $\mathcal{V}$ *p*;
(*mem*′, $\mathcal{V}'$, *va*) ← *if vars-llist p* ⊆ *set-mset* $\mathcal{V}$ ∧ (*i* ∉# *dom-m A* ∧ *v* ∉# $\mathcal{V}$) ∧ ¬ *alloc-failed mem*
then import-variable v $\mathcal{V}'$ *else RETURN* (*mem*, $\mathcal{V}'$, *v*);
RETURN
((*i* ∉# *dom-m A* ∧ *v* ∉# $\mathcal{V}$) ∧ ¬ *alloc-failed mem* ∧ ¬ *alloc-failed mem*′,
vars-llist p ⊆ *set-mset* $\mathcal{V}$, *mem*, *mem*′, *pa*, $\mathcal{V}'$, *va*)
} ≤ ⇓ {((*pre*, *nonew*, *mem*, *mem*′, *p*′, $\mathcal{A}$, *va*), *b*). (*b*=*pre*)∧ (*b* ⟶ ¬*alloc-failed mem* ∧ ¬*alloc-failed mem*′) ∧
(*b* ∧ *nonew* ⟶ (*p*′=*p* ∧ *set-mset* $\mathcal{A}$ = *set-mset* $\mathcal{V}$ ∪ *vars-llist p* ∪ {*v*} ∧ *va* = *v*)) ∧
((*nonew* ⟷ *vars-llist p* ⊆ *set-mset* $\mathcal{V}$))}
(*SPEC* (λ*b*. *b* ⟶ *i*′ ∉# *dom-m A*′ ∧ *v*′ ∉ $\mathcal{V}'$))›
using *assms* **unfolding** *conc-fun-RES import-variable-def nres-monad3*
apply (*subst* (*2*) *RES-SPEC-eq*)
apply (*refine-vcg import-poly-spec*[*THEN order-trans*])
apply (*clarsimp simp*:)
apply (*rule conjI impI*)
apply (*refine-vcg import-poly-spec*[*THEN order-trans*])
apply (*auto simp*: *vars-llist-def*)[]
apply (*auto simp*: *vars-llist-def*)[]
apply (*auto simp*: *vars-llist-def*)[]
done

have *H*: ‹*f*=*g* ⟹ *f* ≤ ⇓*Id g*› **for** *f g*
by *auto*
show *?thesis*
using *assms*
unfolding *check-extension-l2-prop-def check-extension-l2-def*
apply (*refine-vcg mult-poly-full-prop-mult-poly-full add-poly-alt-def*[*unfolded add-poly-l*′*-def*, *THEN order-trans*]
)
subgoal by *auto*
apply (*rule H*)
subgoal by *auto*
subgoal by (*simp add*: *error-msg-def*)
subgoal by *auto*
apply (*rule H*)
subgoal by (*auto simp*: *check-extension-l-new-var-multiple-err-def*)
subgoal by (*simp add*: *error-msg-def*)
subgoal by *auto*
subgoal by *auto*

```
    subgoal by auto
    subgoal by auto
    subgoal by auto
    subgoal using assms by (auto dest: split-list-first simp: vars-llist-def)
    subgoal by (auto simp: vars-llist-def)
    apply (rule H)
    subgoal by auto
    subgoal by auto
    apply (rule H)
    subgoal by auto
    subgoal by auto
    subgoal by (auto dest!: split-list-first simp: remove1-append)
    apply (rule H)
   subgoal by (auto simp: check-extension-l-new-var-multiple-err-def check-extension-l-side-cond-err-def)
    subgoal by (auto simp: error-msg-def)
    done
qed


definition PAC-checker-l-step-prep :: ‹- ⇒ string code-status × string multiset × - ⇒ (llist-polynomial,
string, nat) pac-step ⇒ -› where
 ‹PAC-checker-l-step-prep = (λspec (st', 𝒱, A) st. do {
   ASSERT (PAC-checker-l-step-inv spec st' (set-mset 𝒱) A);
   ASSERT (¬is-cfailed st');
   case st of
    CL - - - ⇒
      do {
       r ← full-normalize-poly (pac-res st);
       (eq, r) ← check-linear-combi-l-prop spec A 𝒱 (new-id st) (pac-srcs 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)
    }
  | Extension - - - ⇒
      do {
        r ← full-normalize-poly (pac-res st);
       (eq, r, 𝒱, v) ← check-extension-l2-prop spec A (𝒱) (new-id st) (new-var st) r;
       if ¬is-cfailed eq
     then do {
       r ← add-poly-l-prep 𝒱 ([([v], −1)], r);
       RETURN (st', 𝒱, fmupd (new-id st) r A)
       }
       else RETURN (eq, 𝒱, A)
    }}
          )›
```

```
lemma PAC-checker-l-step-prep-PAC-checker-l-step:
  assumes ‹(state, state′) ∈ {((st, 𝒱, A), (st′, 𝒱′, A′)). (st,st′)∈Id ∧ (A,A′)∈Id ∧ (¬is-cfailed st ⟶
(𝒱,𝒱′)∈ {(x, y). y = set-mset x})}›
    ‹(spec,spec′)∈Id›
    ‹(step,step′)∈Id›
  shows ‹PAC-checker-l-step-prep spec state step ≤
      ⇓{((st, 𝒱, A), (st′, 𝒱′, A′)). (st,st′)∈Id ∧ (A,A′)∈Id ∧ (¬is-cfailed st ⟶ (𝒱,𝒱′)∈ {(x, y). y =
set-mset x})}
    (PAC-checker-l-step spec′ state′ step′)›
proof -
  have H: ‹f=g ⟹ f ≤ ⇓Id g› for f g
    by auto
  show ?thesis
    using assms apply -
    unfolding PAC-checker-l-step-prep-def PAC-checker-l-step-def
    apply (simp only: split: prod.splits)
    apply (simp only: split:prod.splits pac-step.splits)
    apply (intro conjI impI allI)
    subgoal
      apply (refine-rcg check-linear-combi-l-prop-check-linear-combi-l)
      subgoal using assms by auto
      subgoal by auto
      apply (rule H)
      subgoal by auto
      subgoal by auto
      subgoal by auto
      subgoal by auto
      subgoal by auto
      subgoal by auto
      subgoal by auto
      subgoal by auto
      subgoal by auto
      done
    subgoal by auto
    subgoal by auto
    subgoal by auto
    subgoal
    apply (refine-rcg check-extension-l2-prop-check-extension-l2 add-poly-alt-def[unfolded add-poly-l′-def,
THEN order-trans])
      subgoal by auto
      subgoal by auto
      apply (rule H)
      subgoal by auto
      subgoal by auto
      subgoal by auto
      subgoal by auto
      subgoal by auto
      subgoal by auto
      subgoal by auto
      subgoal by auto
      subgoal by (auto simp add: vars-llist-def)
      apply (rule H)
      subgoal by auto
      subgoal by auto
      subgoal by auto
```

```
      done
    subgoal by auto
    subgoal by auto
    subgoal by auto
    subgoal
      apply (refine-rcg)
      subgoal by auto
      subgoal by auto
      apply (rule H)
      subgoal by auto
      subgoal by auto
      subgoal by auto
      subgoal by auto
      done
    done
qed

definition (in −) remap-polys-l2-with-err
  :: ‹llist-polynomial ⇒ llist-polynomial ⇒ (nat, string) vars ⇒ (nat, llist-polynomial) fmap ⇒
   (string code-status × (nat, string) vars × (nat, llist-polynomial) fmap) nres› where
  ‹remap-polys-l2-with-err spec′ spec0 = (λ(𝒱:: (nat, string) vars) A. do{
   ASSERT(vars-llist spec′ ⊆ vars-llist spec0);
   dom ← SPEC(λdom. set-mset (dom-m A) ⊆ dom ∧ finite dom);
   (mem, 𝒱) ← SPEC(λ(mem, 𝒱′). ¬alloc-failed mem ⟶ set-mset 𝒱′ = set-mset 𝒱 ∪ vars-llist spec0);
   (mem′, spec, 𝒱) ← if ¬alloc-failed mem then import-poly 𝒱 spec′ else SPEC(λ-. True);
   failed ← SPEC(λb::bool. alloc-failed mem ∨ alloc-failed mem′ ⟶ b);
   ASSERT(¬failed ⟶ spec = spec′);
   if failed
   then do {
      c ← remap-polys-l-dom-err;
      SPEC (λ(mem, -, -). mem = error-msg (0::nat) c)
   }
   else do {
     (err, 𝒱, A) ← FOREACH_C dom (λ(err, 𝒱,  A′). ¬is-cfailed err)
       (λi (err, 𝒱,  A′).
          if i ∈# dom-m A
          then  do {
           (err′, p, 𝒱) ← import-poly 𝒱 (the (fmlookup A i));
            if alloc-failed err′ then RETURN((CFAILED ″memory out″, 𝒱, A′))
            else do {
              ASSERT(vars-llist p ⊆ set-mset 𝒱);
              p ← full-normalize-poly p;
              eq  ← weak-equality-l p spec;
              let 𝒱 = 𝒱;
              RETURN((if eq then CFOUND else CSUCCESS), 𝒱, fmupd i p A′)
            }
          } else RETURN (err, 𝒱, A′))
       (CSUCCESS, 𝒱, fmempty);
     RETURN (err, 𝒱, A)
  }})›

lemma remap-polys-l-with-err-alt-def:
  ‹remap-polys-l-with-err spec spec0 = (λ𝒱 A. do{
  ASSERT (remap-polys-l-with-err-pre spec spec0 𝒱 A);
  dom ← SPEC(λdom. set-mset (dom-m A) ⊆ dom ∧ finite dom);
```

$\mathcal{V} \leftarrow$ *RETURN* ($\mathcal{V} \cup$ *vars-llist spec0*);
spec ← *RETURN spec*;
failed ← *SPEC*(λ-::*bool*. *True*);
if failed
then do {
  c ← *remap-polys-l-dom-err*;
  SPEC (λ(*mem*, -, -). *mem* = *error-msg* (*0*::*nat*) *c*)
}
else do {
  (*err*, $\mathcal{V}$, *A*) ← $FOREACH_C$ *dom* (λ(*err*, $\mathcal{V}$, *A′*). ¬*is-cfailed err*)
    (λ*i* (*err*, $\mathcal{V}$, *A′*).
      if i ∈# *dom-m A*
      then do {
        err′ ← *SPEC*(λ*err*. *err* ≠ *CFOUND*);
        if is-cfailed err′ then RETURN((*err′*, $\mathcal{V}$, *A′*))
        else do {
          p ← *full-normalize-poly* (*the* (*fmlookup A i*));
          eq ← *weak-equality-l p spec*;
          $\mathcal{V}$ ← *RETURN*($\mathcal{V}$ ∪ *vars-llist* (*the* (*fmlookup A i*)));
          RETURN((*if eq then CFOUND else CSUCCESS*), $\mathcal{V}$, *fmupd i p A′*)
        }
      } *else RETURN* (*err*, $\mathcal{V}$, *A′*))
    (*CSUCCESS*, $\mathcal{V}$, *fmempty*);
  RETURN (*err*, $\mathcal{V}$, *A*)
}})›
unfolding *remap-polys-l-with-err-def* **by** (*auto intro*!: *ext bind-cong*[*OF refl*])

lemma *remap-polys-l2-with-err-polys-l2-with-err*:
assumes ‹($\mathcal{V}$, $\mathcal{V}'$) ∈ {(*x*, *y*). *y* = *set-mset x*}› ‹(*A*,*A′*) ∈ *Id*› ‹(*spec*, *spec′*)∈*Id*› ‹(*spec0*, *spec0′*)∈*Id*›
shows ‹*remap-polys-l2-with-err spec spec0* $\mathcal{V}$ *A* ≤ ⇓{((*st*, $\mathcal{V}$, *A*), *st′*, $\mathcal{V}'$, *A′*).
(*st*, *st′*) ∈ *Id* ∧
(*A*, *A′*) ∈ *Id* ∧
(¬ *is-cfailed st* ⟶ ($\mathcal{V}$, $\mathcal{V}'$) ∈ {(*x*, *y*). *y* = *set-mset x*})}
(*remap-polys-l-with-err spec′ spec0′* $\mathcal{V}'$ *A′*)›
proof −
have [*refine*]: ‹*inj-on id dom*› **for** *dom*
by (*auto simp*: *inj-on-def*)
have [*refine*]: ‹((*CSUCCESS*, $\mathcal{V}$, *fmempty*), *CSUCCESS*, $\mathcal{V}'$, *fmempty*) ∈ {((*st*, $\mathcal{V}$, *A*), *st′*, $\mathcal{V}'$, *A′*).
(*st*, *st′*) ∈ *Id* ∧ (*A*, *A′*) ∈ *Id* ∧ (¬ *is-cfailed st* ⟶ ($\mathcal{V}$, $\mathcal{V}'$) ∈ {(*x*, *y*). *y* = *set-mset x*})}›
if ‹($\mathcal{V}$, $\mathcal{V}'$) ∈ {(*x*, *y*). *y* = *set-mset x*}›
for $\mathcal{V}$ $\mathcal{V}'$
using *assms that*
by *auto*
have [*refine*]: ‹*import-poly x1c p* ≤⇓{((*mem*, *ys*, $\mathcal{A}'$), (*err* :: *string code-status*)). (*alloc-failed mem* ⟷ *err* = *CFAILED ″memory out″*) ∧
(¬*alloc-failed mem* ⟷ *err* = *CSUCCESS*) ∧
(¬ *alloc-failed mem* ⟶ *ys* = *p* ∧ *set-mset* $\mathcal{A}'$ = *set-mset x1c* ∪ ⋃ (*set* ‘ *fst* ‘ *set p*))}
(*SPEC* (λ*err*. *err* ≠ *CFOUND*))› **for** *x1c p*
apply (*rule order-trans*[*OF import-poly-spec*])
apply (*auto simp*: *conc-fun-RES*)
done

have *id*: ‹*f*=*g* ⟹ *f* ≤⇓*Id g*› **for** *f g*
by *auto*
have *id2*: ‹(*f*,*g*)∈{(*x*, *y*). *y* = *set-mset x*} ⟹ (*f*,*g*)∈{(*x*, *y*). *y* = *set-mset x*}› **for** *f g*

by *auto*
have [*refine*]: ‹ $SPEC\ (\lambda(mem, \mathcal{V}').\ \neg\ alloc\text{-}failed\ mem \longrightarrow set\text{-}mset\ \mathcal{V}' = set\text{-}mset\ \mathcal{V} \cup vars\text{-}llist\ spec0)$
$\leq SPEC\ (\lambda c.\ (c, \mathcal{V}' \cup vars\text{-}llist\ spec0') \in \{((mem, x), y).\ \neg\ alloc\text{-}failed\ mem \longrightarrow y = set\text{-}mset\ x\})$›
using *assms* **by** *auto*
have [*refine*]: ‹$(x, \mathcal{V}''') \in \{((mem, x), y).\ \neg\ alloc\text{-}failed\ mem \longrightarrow y = set\text{-}mset\ x\} \Longrightarrow$
$x = (x1, \mathcal{V}'') \Longrightarrow$
$(if\ \neg\ alloc\text{-}failed\ x1\ then\ import\text{-}poly\ \mathcal{V}''\ spec\ else\ SPEC(\lambda\text{-}.\ True)) \leq SPEC\ (\lambda c.\ (c, spec') \in$
$\{((new, ys, \mathcal{A}'), spec').$
$(\neg\ alloc\text{-}failed\ new \wedge \neg\ alloc\text{-}failed\ x1 \longrightarrow$
$ys = spec \wedge\ spec' = spec \wedge set\text{-}mset\ \mathcal{A}' = set\text{-}mset\ \mathcal{V}'' \cup \bigcup\ (set\ `\ mononoms\ spec))\})$›
for $\mathcal{V}''\ \mathcal{V}'''\ x\ x1$
using *assms*
by (*auto split*: *if-splits intro*!: *import-poly-spec*[*THEN order-trans*])
have [*simp*]: ‹$(\bigcup a{\in}set\ spec'.\ set\ (fst\ a)) \subseteq (\bigcup a{\in}set\ spec0'.\ set\ (fst\ a)) \Longrightarrow$
$set\text{-}mset\ \mathcal{V} \cup (\bigcup x{\in}set\ spec0'.\ set\ (fst\ x)) \cup (\bigcup x{\in}set\ spec'.\ set\ (fst\ x)) =$
$set\text{-}mset\ \mathcal{V} \cup (\bigcup x{\in}set\ spec0'.\ set\ (fst\ x))$›
by (*auto*)
show *?thesis*
unfolding *remap-polys-l2-with-err-def remap-polys-l-with-err-alt-def*
apply (*refine-rcg*)
subgoal using *assms* **unfolding** *remap-polys-l-with-err-pre-def* **by** *auto*
subgoal using *assms* **by** *auto*
apply *assumption*
subgoal by *auto*
subgoal using *assms* **by** *auto*
subgoal by (*auto simp*: *error-msg-def*)

subgoal using *assms* **by** (*simp add*:)
subgoal by (*auto intro*!: *RES-refine*)
subgoal using *assms* **by** (*simp add*:)
subgoal using *assms* **by** (*simp add*: *remap-polys-l-with-err-pre-def vars-llist-def*)
subgoal using *assms* **by** *auto*
subgoal using *assms* **by** *auto*
subgoal by *auto*
subgoal using *assms* **by** *auto*
subgoal by (*auto simp*: *vars-llist-def*)
apply (*rule id*)
subgoal using *assms* **by** *auto*
apply (*rule id*)
subgoal using *assms* **by** *auto*
apply (*rule id2*)
subgoal using *assms* **by** (*clarsimp simp add*: *vars-llist-def*)
subgoal using *assms* **by** *auto*
subgoal using *assms* **by** *auto*
subgoal using *assms* **by** *auto*
done
qed

definition *PAC-checker-l2* **where**
‹$PAC\text{-}checker\text{-}l2\ spec\ A\ b\ st = do\ \{$
$(S, \text{-}) \leftarrow WHILE_T$
$(\lambda((b, A), n).\ \neg is\text{-}cfailed\ b \wedge n \neq [])$
$(\lambda((bA), n).\ do\ \{$
$ASSERT(n \neq [\,]);$

S ← *PAC-checker-l-step-prep spec bA* (*hd n*);
RETURN (*S*, *tl n*)
})
((*b*, *A*), *st*);
RETURN S
}›

lemma *PAC-checker-l2-PAC-checker-l*:
assumes ‹(*A*, *A'*) ∈ {(*x*, *y*). *y* = *set-mset x*} ×ᵣ *Id*› ‹(*spec*, *spec'*)∈*Id*› ‹(*st*, *st'*)∈*Id*› ‹(*b*,*b'*)∈*Id*›
shows ‹*PAC-checker-l2 spec A b st* ≤ ⇓{((*b*, *A*, *st*), (*b'*, *A'*, *st'*)).
(¬*is-cfailed b* ⟶ (*A*, *A'*) ∈ {(*x*, *y*). *y* = *set-mset x*} ∧ (*st*, *st'*)∈*Id*) ∧ (*b*,*b'*)∈*Id*} (*PAC-checker-l spec' A' b' st'*)›
proof −
show *?thesis*
unfolding *PAC-checker-l2-def PAC-checker-l-def*
apply (*refine-rcg*
PAC-checker-l-step-prep-PAC-checker-l-step
WHILET-refine[**where** *R* = ‹{(((*b*, *A*), *st*:: (*llist-polynomial*, *string*, *nat*) *pac-step list*), (*b'*, *A'*), *st'*).
(¬*is-cfailed b* ⟶ (*A*, *A'*) ∈ {(*x*, *y*). *y* = *set-mset x*} ×ᵣ *Id*) ∧ (*b*,*b'*)∈*Id* ∧ (*st*,*st'*)∈*Id*}›])
subgoal using *assms* **by** *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal using *assms* **by** *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
done
qed

definition (**in** −) *remap-polys-l2-with-err-prep* :: ‹*llist-polynomial* ⇒ *llist-polynomial* ⇒ (*nat*, *string*) *vars* ⇒ (*nat*, *llist-polynomial*) *fmap* ⇒
(*string code-status* × (*nat*, *string*) *vars* × (*nat*, *llist-polynomial*) *fmap* × *llist-polynomial*) *nres*› **where**
‹*remap-polys-l2-with-err-prep spec spec0* = (λ(𝒱:: (*nat*, *string*) *vars*) *A*. *do*{
ASSERT(*vars-llist spec* ⊆ *vars-llist spec0*);
dom ← *SPEC*(λ*dom*. *set-mset* (*dom-m A*) ⊆ *dom* ∧ *finite dom*);
(*mem*, 𝒱) ← *SPEC*(λ(*mem*, 𝒱'). ¬*alloc-failed mem* ⟶ *set-mset* 𝒱' = *set-mset* 𝒱 ∪ *vars-llist spec0*);
(*mem'*, *spec*, 𝒱) ← *if* ¬*alloc-failed mem then import-poly* 𝒱 *spec else SPEC*(λ-. *True*);
failed ← *SPEC*(λ*b*::*bool*. *alloc-failed mem* ∨ *alloc-failed mem'* ⟶ *b*);
if failed
then do {
c ← *remap-polys-l-dom-err*;
SPEC (λ(*mem*, -, -, -). *mem* = *error-msg* (*0*::*nat*) *c*)
}
else do {
(*err*, 𝒱, *A*) ← $FOREACH_C$ *dom* (λ(*err*, 𝒱, *A'*). ¬*is-cfailed err*)
(λ*i* (*err*, 𝒱, *A'*).
if i ∈# *dom-m A*
then do {
(*err'*, *p*, 𝒱) ← *import-poly* 𝒱 (*the* (*fmlookup A i*));
if alloc-failed err' then RETURN((*CFAILED ''memory out''*, 𝒱, *A'*))
else do {
ASSERT(*vars-llist p* ⊆ *set-mset* 𝒱);
p ← *full-normalize-poly p*;

eq ← *weak-equality-l p spec*;
let $\mathcal{V}$ = $\mathcal{V}$;
RETURN((*if eq then CFOUND else CSUCCESS*), $\mathcal{V}$, *fmupd i p A'*)
}
} *else RETURN* (*err*, $\mathcal{V}$, *A'*))
(*CSUCCESS*, $\mathcal{V}$, *fmempty*);
RETURN (*err*, $\mathcal{V}$, *A*, *spec*)
}})›

lemma *remap-polys-l2-with-err-prep-remap-polys-l2-with-err*:
assumes ‹(*p*, *p'*) ∈ *Id*› ‹(*q*, *q'*) ∈ *Id*› ‹(*A*,*A'*) ∈ ⟨*Id*, *Id*⟩*fmap-rel*› **and** ‹(*V*,*V'*) ∈ *Id*›
shows ‹*remap-polys-l2-with-err-prep p q V A* ≤ ⇓{((*b*, *A*, *st*, *spec'*), (*b'*, *A'*, *st'*)).
((*b*, *A*, *st*), (*b'*, *A'*, *st'*)) ∈ *Id* ∧
(¬*is-cfailed b* ⟶ *spec'* = *p'*)} (*remap-polys-l2-with-err p' q' V' A'*)›
proof −
have [*simp*]: ‹⟨*Id*, *Id*⟩*fmap-rel* = *Id*›
apply (*auto simp*: *fmap-rel-def fmlookup-dom-iff intro*!: *fmap-ext dest*: *fmdom-notD*)
by (*metis fmdom-notD fmlookup-dom-iff option.sel*)

have *1*: ‹*inj-on id* (*dom* :: *nat set*)› **for** *dom*
by *auto*
have [*refine*]:
‹(*x2e*, *x2c*)∈*Id* ⟹ ((*CSUCCESS*, *x2e*, *fmempty*), *CSUCCESS*, *x2c*, *fmempty*) ∈ *Id* $\times_r$ *Id* $\times_r$ ⟨*Id*, *Id*⟩*fmap-rel*› **for** *x2e x2c*
by *auto*
have [*refine*]: ‹*import-poly x y* ≤ ⇓ *Id* (*import-poly x' y'*)›
if ‹(*x*,*x'*) ∈ *Id*›‹(*y*,*y'*) ∈ *Id*› **for** *x x' y y'*
using *that* **by** *auto*
have [*refine*]: ‹*full-normalize-poly x* ≤ ⇓ *Id* (*full-normalize-poly x'*)›
if ‹(*x*,*x'*)∈*Id*› **for** *x x'*
using *that* **by** *auto*
have [*refine*]: ‹*weak-equality-l x y* ≤ ⇓ *bool-rel* (*weak-equality-l x' y'*)›
if ‹(*x*,*x'*)∈*Id*› ‹(*y*,*y'*)∈*Id*› **for** *x x' y y'*
using *that* **by** *auto*

show *?thesis*
unfolding *remap-polys-l2-with-err-prep-def remap-polys-l2-with-err-def*
apply (*refine-vcg 1*)
subgoal using *assms* **by** *auto*
subgoal using *assms* **by** *auto*
subgoal using *assms* **by** *auto*
subgoal using *assms* **by** *auto*
subgoal using *assms* **by** *auto*
subgoal using *assms* **by** *auto*
subgoal using *assms* **by** *auto*
subgoal using *assms* **by** *auto*
subgoal using *assms* **by** *auto*
subgoal by (*auto intro*!: *RES-refine simp*: *error-msg-def*)
subgoal using *assms* **by** *auto*
subgoal using *assms* **by** *auto*
subgoal using *assms* **by** *auto*
subgoal using *assms* **by** *auto*
subgoal using *assms* **by** *auto*
subgoal using *assms* **by** *auto*
subgoal by *auto*

subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal using *assms* **by** *auto*
subgoal by *auto*
subgoal by *auto*
subgoal using *assms* **by** *auto*
done
qed

definition *full-checker-l-prep*
:: ‹*llist-polynomial* ⇒ (*nat*, *llist-polynomial*) *fmap* ⇒ (-, *string*, *nat*) *pac-step list* ⇒
(*string code-status* × -) *nres*›
where
‹*full-checker-l-prep spec A st* = *do* {
spec′ ← *full-normalize-poly spec*;
(*b*, $\mathcal{V}$, *A*, *spec*) ← *remap-polys-l2-with-err-prep spec*′ *spec* {#} *A*;
if is-cfailed b
then RETURN (*b*, $\mathcal{V}$, *A*)
else do {
let $\mathcal{V}$ = $\mathcal{V}$;
PAC-checker-l2 spec ($\mathcal{V}$, *A*) *b st*
}
}›

lemma *remap-polys-l2-with-err-polys-l-with-err*:
assumes ‹($\mathcal{V}$, $\mathcal{V}'$) ∈ {(*x*, *y*). *y* = *set-mset x*}› ‹(*A*,*A*′) ∈ *Id*› ‹(*spec*, *spec*′)∈*Id*› ‹(*spec0*, *spec0*′)∈*Id*›
shows ‹*remap-polys-l2-with-err-prep spec spec0* $\mathcal{V}$ *A* ≤ ⇓{((*st*, $\mathcal{V}$, *A*, *spec*′′), *st*′, $\mathcal{V}'$, *A*′).
(*st*, *st*′) ∈ *Id* ∧
(*A*, *A*′) ∈ *Id* ∧
(¬ *is-cfailed st* ⟶ ($\mathcal{V}$, $\mathcal{V}'$) ∈ {(*x*, *y*). *y* = *set-mset x*} ∧ *spec*′′ = *spec*)}
(*remap-polys-l-with-err spec*′ *spec0*′ $\mathcal{V}'$ *A*′)›
proof −
have [*simp*]: ‹⟨*Id*, *Id*⟩*fmap-rel* = *Id*›
apply (*auto simp*: *fmap-rel-def fmlookup-dom-iff intro*!: *fmap-ext dest*: *fmdom-notD*)
by (*metis fmdom-notD fmlookup-dom-iff option.sel*)

have *A*: ‹(*A*, *A*′) ∈ ⟨*nat-rel*, *Id*⟩*fmap-rel*› ‹($\mathcal{V}$, $\mathcal{V}$) ∈ *Id*› ‹(*A*′,*A*′) ∈ *Id*›
‹(*spec*′,*spec*′)∈ *Id*› ‹(*spec0*′, *spec0*′) ∈ *Id*›
using *assms*(*2*) **by** *auto*
show *?thesis*
apply (*rule remap-polys-l2-with-err-prep-remap-polys-l2-with-err*[*THEN order-trans*])
apply (*rule assms A*)+
apply (*rule order-trans*)
apply (*rule ref-two-step*′)
apply (*rule remap-polys-l2-with-err-polys-l2-with-err*)
apply (*rule assms A*)+
apply (*subst conc-fun-chain*)
apply (*rule conc-fun-R-mono*)
apply (*use assms* **in** *auto*)
done
qed

lemma *full-checker-l-prep-full-checker-l*:

assumes ⟨(*spec*, *spec'*)∈*Id*⟩ ⟨(*st*, *st'*)∈*Id*⟩ ⟨(*A*,*A'*)∈*Id*⟩
shows ⟨*full-checker-l-prep spec A st* ≤⇓{((*b*, *A*, *st*), (*b'*, *A'*, *st'*)).
(¬*is-cfailed b* ⟶ (*A*, *A'*) ∈ {(*x*, *y*). *y* = *set-mset x*} ∧ (*st*, *st'*)∈*Id*) ∧ (*b*,*b'*)∈*Id*}
(*full-checker-l spec' A' st'*)⟩
proof −
have *id*: ⟨*f*=*g* ⟹ *f* ≤⇓*Id g*⟩ **for** *f g*
by *auto*
show *?thesis*
unfolding *full-checker-l-prep-def full-checker-l-def*
apply (*refine-rcg remap-polys-l2-with-err-polys-l-with-err*
PAC-checker-l2-PAC-checker-l remap-polys-l2-with-err-polys-l-with-err)
apply (*rule id*)
subgoal using *assms* **by** *auto*
subgoal by *auto*
apply (*rule assms*)
subgoal by *auto*
apply (*rule assms*)
subgoal by *auto*
subgoal using *assms* **by** *auto*
subgoal by *auto*
subgoal using *assms* **by** *auto*
subgoal using *assms* **by** *auto*
subgoal by *auto*
done
qed

lemma *full-checker-l-prep-full-checker-l2'*:
shows ⟨(*uncurry2 full-checker-l-prep*, *uncurry2 full-checker-l*) ∈ (*Id* $\times_r$ *Id*) $\times_r$ *Id* $\to_f$
⟨{((*b*, *A*, *st*), (*b'*, *A'*, *st'*)). (¬*is-cfailed b* ⟶ (*A*, *A'*) ∈ {(*x*, *y*). *y* = *set-mset x*} ∧ (*st*, *st'*)∈*Id*) ∧ (*b*,*b'*)∈*Id*}⟩*nres-rel*⟩
by (*auto intro*!: *frefI nres-relI full-checker-l-prep-full-checker-l*[*THEN order-trans*])

end
theory *EPAC-Perfectly-Shared-Vars*
imports *EPAC-Perfectly-Shared*
PAC-Checker.PAC-Checker-Relation
PAC-Checker.PAC-Map-Rel
begin
thm *import-variableS-def*
term *hm.assn*
term *iam.assn*
term *is-iam*
term *iam-rel*

type-synonym (*'string2*, *'nat*) *shared-vars-c* = ⟨*'string2 list* × (*'string2*, *'nat*) *fmap*⟩

definition *perfect-shared-vars-rel-c* :: ⟨(*'string2* × *'string*) *set* ⇒ ((*'string2*, *nat*) *shared-vars-c* × (*nat*, *'string*)*shared-vars*) *set*⟩ **where**
⟨*perfect-shared-vars-rel-c R* =
{(($\mathcal{V}$, $\mathcal{A}$), ($\mathcal{D}'$, $\mathcal{V}'$, $\mathcal{A}'$)). (∀ *i*∈#*dom-m* $\mathcal{V}'$. *i* < *length* $\mathcal{V}$) ∧
(∀ *i*∈#*dom-m* $\mathcal{V}'$. *i* < *length* $\mathcal{V}$ ∧ ($\mathcal{V}$! *i*, *the* (*fmlookup* $\mathcal{V}'$ *i*))∈ *R*) ∧
($\mathcal{A}$, $\mathcal{A}'$) ∈ ⟨*R*,*nat-rel*⟩*fmap-rel*}⟩

Random conditions with the idea to use machine words eventually

definition *find-new-idx-c* :: ⟨(′*string*, *nat*) *shared-vars-c* ⇒ (*memory-allocation* × *nat*) *nres*⟩ **where**
⟨*find-new-idx-c* = (λ(𝒱, 𝒜). *let k* = *length* 𝒱 *in if k* < *2^63−1 then RETURN* (*Allocated*, *k*) *else RETURN* (*Mem-Out*, *0*))⟩

definition *insert-variable-c* :: ⟨′*string* ⇒ *nat* ⇒ (′*string*, *nat*) *shared-vars-c* ⇒ (′*string*, *nat*) *shared-vars-c*⟩
where
⟨*insert-variable-c v k*′ = (λ(𝒱, 𝒜). (𝒱 @ [*v*], *fmupd v k*′ 𝒜))⟩

definition *import-variable-c* :: ⟨′*string* ⇒ (′*string*, *nat*) *shared-vars-c* ⇒ (*memory-allocation* × (′*string*, *nat*) *shared-vars-c* × *nat*) *nres*⟩ **where**
⟨*import-variable-c v* = (λ(𝒱𝒜). *do* {
(*err*, *k*′) ← *find-new-idx-c* (𝒱𝒜);
if alloc-failed err then do {*let k*′=*k*′; *RETURN* (*err*, (𝒱𝒜), *k*′)}
else do{
ASSERT(*k*′ < *2^63−1*);
RETURN (*Allocated*, *insert-variable-c v k*′ 𝒱𝒜, *k*′)
}
})⟩

lemma *import-variable-c-alt-def*:
⟨*import-variable-c v* = (λ(𝒱, 𝒜). *do* {
(*err*, *k*′) ← *find-new-idx-c* (𝒱, 𝒜);
if alloc-failed err then do {*let k*′=*k*′; *RETURN* (*err*, (𝒱, 𝒜), *k*′)}
else do{
ASSERT(*k*′ < *2^63−1*);
RETURN (*Allocated*, (𝒱 @ [*v*], *fmupd v k*′ 𝒜), *k*′)
}
})⟩
unfolding *import-variable-c-def insert-variable-c-def*
by *auto*

lemma *import-variable-c-import-variableS*:
fixes *A*′ :: ⟨(*nat*,′*string*) *shared-vars*⟩
assumes
A: ⟨(*A*,*A*′)∈*perfect-shared-vars-rel-c R*⟩ **and**
v: ⟨(*v*,*v*′)∈*R*⟩ ⟨*single-valued R*⟩ ⟨*single-valued* (R^{-1})⟩
shows ⟨*import-variable-c v A* ≤⇓(*Id* $\times_r$ (*perfect-shared-vars-rel-c R* $\times_r$ *nat-rel*)) (*import-variableS v*′ *A*′)⟩
proof −
have [*refine*]: ⟨*RETURN x2g* ≤ ⇓ *Id* (*RES UNIV*)⟩ **for** *x2g* :: *nat*
by *auto*
have [*refine*]: ⟨*find-new-idx-c a* ≤ ⇓ {((*err*, *k*), (*err*′, *k*′)). *err*=*err*′ ∧ *k*=*k*′ ∧ (¬*alloc-failed err* ⟶ *k* < *2^63−1* ∧ *k* = *length* (*fst a*))} (*find-new-idx b*)⟩
if ⟨(*a*,*b*) ∈ *perfect-shared-vars-rel-c R*⟩
for *b* :: ⟨(*nat*,′*string*) *shared-vars*⟩ **and** *a*
using *that* **unfolding** *find-new-idx-c-def find-new-idx-def*
by (*cases b*; *cases a*)
(*auto intro*!: *RETURN-RES-refine simp*: *Let-def perfect-shared-vars-rel-c-def*)

show *?thesis*
unfolding *import-variable-c-alt-def import-variableS-def find-new-idx-def*[*symmetric*]
apply *refine-vcg*
subgoal using *A* **by** (*auto simp*: *perfect-shared-vars-rel-c-def*)
subgoal by *auto*

```
    subgoal using A by (auto simp: perfect-shared-vars-rel-c-def)
    subgoal by auto
    subgoal
     using A v by (force simp: perfect-shared-vars-rel-c-def dest: in-diffD intro!: fmap-rel-fmupd-fmap-rel)
   done
qed


definition is-new-variable-c :: ‹'string ⇒ ('string, 'nat) shared-vars-c ⇒ bool nres› where
  ‹is-new-variable-c v = (λ(𝒱, 𝒱').
  RETURN (v ∉# dom-m 𝒱')
  )›

lemma fset-fmdom-dom-m: ‹fset (fmdom A) = set-mset (dom-m A)›
  by (simp add: dom-m-def)

lemma fmap-rel-nat-rel-dom-m-iff:
  ‹(A, B) ∈ ⟨R, S⟩fmap-rel ⟹ (v,v')∈R ⟹ v∈#dom-m A ⟷v'∈# dom-m B›
  by (auto simp: fmap-rel-alt-def distinct-mset-dom fset-fmdom-dom-m
    dest!: multi-member-split
    simp del: fmap-rel-nat-the-fmlookup)


lemma is-new-variable-c-is-new-variableS:
  shows ‹(uncurry is-new-variable-c, uncurry is-new-variableS) ∈ R ×r perfect-shared-vars-rel-c R →f
⟨bool-rel⟩nres-rel›
  by (use in ‹auto simp: perfect-shared-vars-rel-c-def fmap-rel-nat-rel-dom-m
    fmap-rel-nat-rel-dom-m-iff is-new-variable-c-def is-new-variableS-def
    intro!: frefI nres-relI›)


definition get-var-pos-c :: ‹ ('string, nat) shared-vars-c ⇒ - ⇒ nat nres› where
  ‹get-var-pos-c = (λ(xs, 𝒱) x. do {
    ASSERT(x ∈# dom-m 𝒱);
    RETURN (the (fmlookup 𝒱 x))
  })›


lemma get-var-pos-c-get-var-posS:
  fixes A' :: ‹(nat,'string) shared-vars›
  assumes
    V: ‹single-valued R› ‹single-valued (R⁻¹)›
  shows ‹(uncurry get-var-pos-c, uncurry get-var-posS) ∈ perfect-shared-vars-rel-c R ×r R →f ⟨nat-rel⟩nres-rel›
  unfolding get-var-pos-c-def get-var-posS-def uncurry-def
    apply (clarify intro!: frefI nres-relI)
  apply refine-vcg
  subgoal using assms by (auto simp: perfect-shared-vars-rel-c-def fmap-rel-nat-rel-dom-m-iff)
  subgoal
   using assms by (auto simp: perfect-shared-vars-rel-c-def fmap-rel-nat-rel-dom-m-iff dest: fmap-rel-fmlookup-rel)
  done



definition get-var-name-c :: ‹ ('string, nat) shared-vars-c ⇒ nat ⇒ 'string nres› where
  ‹get-var-name-c = (λ(xs, 𝒱) x. do {
```

```
    ASSERT(x < length xs);
    RETURN (xs ! x)
  })›


lemma get-var-name-c-get-var-nameS:
  fixes A′ :: ‹(nat,′string) shared-vars›
  assumes
    V: ‹single-valued R› ‹single-valued (R⁻¹)›
   shows ‹(uncurry get-var-name-c, uncurry get-var-nameS) ∈ perfect-shared-vars-rel-c R ×r Id →f
⟨R⟩nres-rel›
  unfolding get-var-name-c-def get-var-nameS-def uncurry-def
    apply (clarify intro!: frefI nres-relI)
  apply refine-vcg
  subgoal using assms by (auto dest!: multi-member-split simp: perfect-shared-vars-rel-c-def)
  subgoal
    using assms by (auto simp: perfect-shared-vars-rel-c-def fmap-rel-nat-rel-dom-m-iff
      dest: multi-member-split)
  done

abbreviation perfect-shared-vars-assn :: ‹(string, nat) shared-vars-c ⇒ - ⇒ assn› where
  ‹perfect-shared-vars-assn ≡ arl-assn string-assn ×a hm-fmap-assn string-assn uint64-nat-assn›
abbreviation shared-vars-assn where
  ‹shared-vars-assn ≡ hr-comp perfect-shared-vars-assn (perfect-shared-vars-rel-c Id)›

lemmas [sepref-fr-rules] = hm.lookup-hnr[FCOMP op-map-lookup-fmlookup]

sepref-definition get-var-pos-c-impl
  is ‹uncurry get-var-pos-c›
  :: ‹perfect-shared-vars-assn^k *a string-assn^k →a uint64-nat-assn›
  supply [simp] = in-dom-m-lookup-iff
  unfolding get-var-pos-c-def fmlookup′-def[symmetric]
  by sepref

sepref-definition is-new-variable-c-impl
  is ‹uncurry is-new-variable-c›
  :: ‹string-assn^k  *a  perfect-shared-vars-assn^k →a bool-assn›
  supply [simp] = in-dom-m-lookup-iff
  unfolding is-new-variable-c-def fmlookup′-def[symmetric] in-dom-m-lookup-iff
  by sepref

definition nth-uint64 where
  ‹nth-uint64 = (!)›

definition arl-get′ :: ‹′a::heap array-list ⇒ integer ⇒ ′a Heap› where
  [code del]: ‹arl-get′ a i = arl-get a (nat-of-integer i)›

definition arl-get-u :: ‹′a::heap array-list ⇒ uint64 ⇒ ′a Heap› where
  ‹arl-get-u ≡ λa i. arl-get′ a (integer-of-uint64 i)›

lemma arl-get-hnr-u[sepref-fr-rules]:
  assumes ‹CONSTRAINT is-pure A›
  shows ‹(uncurry arl-get-u, uncurry (RETURN ∘∘ op-list-get))
     ∈ [pre-list-get]a (arl-assn A)^k *a uint64-nat-assn^k → A›
proof −
```

obtain A' **where**
A: ⟨*pure* A' = *A*⟩
using *assms pure-the-pure* **by** *auto*
then have A': ⟨*the-pure A* = A'⟩
by *auto*
have [*simp*]: ⟨*the-pure* (λ*a c*. ↑ ((*c*, *a*) ∈ A')) = A'⟩
unfolding *pure-def*[*symmetric*] **by** *auto*
show *?thesis*
by *sepref-to-hoare*
(*sep-auto simp*: *uint64-nat-rel-def br-def array-assn-def is-array-def*
hr-comp-def list-rel-pres-length param-nth arl-assn-def
A' *A*[*symmetric*] *pure-def arl-get-u-def Array.nth'-def arl-get'-def*
nat-of-uint64-code[*symmetric*])
qed

definition *arl-get-u'* **where**
[*symmetric*, *code*]: ⟨*arl-get-u'* = *arl-get-u*⟩

lemma *arl-get'-nth'*[*code*]: ⟨*arl-get'* = (λ(*a*, *n*). *Array.nth' a*)⟩
unfolding *arl-get-def arl-get'-def Array.nth'-def*
by (*intro ext*) *auto*

definition *nat-of-uint64-s* :: ⟨*nat* ⇒ *nat*⟩ **where**
[*simp*]: ⟨*nat-of-uint64-s x* = *x*⟩

lemma [*refine*]:
⟨(*return o nat-of-uint64*, *RETURN o nat-of-uint64-s*) ∈ *uint64-nat-assn*^{k} $\rightarrow_a$ *nat-assn*⟩
by (*sepref-to-hoare*)
(*sep-auto simp*: *uint64-nat-rel-def br-def*)

sepref-definition *get-var-name-c-impl*
is ⟨*uncurry get-var-name-c*⟩
:: ⟨*perfect-shared-vars-assn*^{k} $*_a$ *uint64-nat-assn*^{k} $\rightarrow_a$ *string-assn*⟩
supply [*simp*] = *in-dom-m-lookup-iff*
unfolding *get-var-name-c-def fmlookup'-def*[*symmetric*]
by *sepref*

lemma [*sepref-fr-rules*]:
⟨(*uncurry is-new-variable-c-impl*, *uncurry is-new-variableS*) ∈ *string-assn*^{k} $*_a$ *shared-vars-assn*^{k} $\rightarrow_a$ *bool-assn*⟩
using *is-new-variable-c-impl.refine*[*FCOMP is-new-variable-c-is-new-variableS*, *of Id*]
by *auto*

lemma [*sepref-fr-rules*]:
⟨(*uncurry get-var-pos-c-impl*, *uncurry get-var-posS*) ∈ *shared-vars-assn*^{k} $*_a$ *string-assn*^{k} $\rightarrow_a$ *uint64-nat-assn*⟩
using *get-var-pos-c-impl.refine*[*FCOMP get-var-pos-c-get-var-posS*, *of Id*]
by *auto*

lemma [*sepref-fr-rules*]:
⟨(*uncurry get-var-name-c-impl*, *uncurry get-var-nameS*) ∈ *shared-vars-assn*^{k} $*_a$ *uint64-nat-assn*^{k} $\rightarrow_a$ *string-assn*⟩
using *get-var-name-c-impl.refine*[*FCOMP get-var-name-c-get-var-nameS*, *of Id*]
by *auto*

sepref-register *get-var-nameS get-var-posS is-new-variableS*

abbreviation *memory-allocation-rel* :: ‹(*memory-allocation* × *memory-allocation*) *set*› **where**
‹*memory-allocation-rel* ≡ *Id*›

abbreviation *memory-allocation-assn* :: ‹*memory-allocation* ⇒ *memory-allocation* ⇒ *assn*› **where**
‹*memory-allocation-assn* ≡ *id-assn*›

instantiation *memory-allocation* :: *default*
begin
definition *default-memory-allocation* :: ‹*memory-allocation*› **where**
‹*default-memory-allocation* = *Allocated*›
instance
..
end

term *import-polyS*
lemma [*sepref-import-param*]:
‹(*Allocated*, *Allocated*) ∈ *memory-allocation-rel*›
‹(*Mem-Out*, *Mem-Out*) ∈ *memory-allocation-rel*›
‹(*alloc-failed*, *alloc-failed*) ∈ *memory-allocation-rel* → *bool-rel*›
by *auto*

lemma *pow-2-63-1*: ‹*2* ^ *63* − *1* = (*9223372036854775807* :: *nat*)›
by *auto*
definition *zero-uint64-nat* **where**
‹*zero-uint64-nat* = *0*›
sepref-register *zero-uint64-nat*
lemma [*sepref-fr-rules*]:
‹(*uncurry0* (*return 0*), *uncurry0* (*RETURN zero-uint64-nat*))∈*unit-assn*^{k} $\rightarrow_a$ *uint64-nat-assn*›
unfolding *zero-uint64-nat-def uint64-nat-rel-def br-def*
by *sepref-to-hoare sep-auto*

definition *length-uint64-nat* **where**
[*simp*]: ‹*length-uint64-nat* = *length*›

definition *length-arl-u-code* :: ‹(′*a*::*heap*) *array-list* ⇒ *uint64 Heap*› **where**
‹*length-arl-u-code xs* = *do* {
n ← *arl-length xs*;
return (*uint64-of-nat n*)}›

definition *uint64-max* :: *nat* **where**
‹*uint64-max* = *2* ^*64* − *1*›

lemma *nat-of-uint64-uint64-of-nat*: ‹*b* ≤ *uint64-max* ⟹ *nat-of-uint64* (*uint64-of-nat b*) = *b*›
unfolding *uint64-of-nat-def uint64-max-def*
apply *simp*
apply *transfer*
by (*auto simp*: *take-bit-nat-eq-self*)

lemma *length-arl-u-hnr*[*sepref-fr-rules*]:
‹(*length-arl-u-code*, *RETURN o length-uint64-nat*) ∈
[λ*xs*. *length xs* ≤ *uint64-max*]$_a$ (*arl-assn R*)^k → *uint64-nat-assn*›

by *sepref-to-hoare*
(*sep-auto simp*: *uint64-nat-rel-def*
length-arl-u-code-def arl-assn-def nat-of-uint64-uint64-of-nat
arl-length-def hr-comp-def is-array-list-def list-rel-pres-length[*symmetric*]
br-def)

lemma *find-new-idx-c-alt-def*:
‹*find-new-idx-c* = (λ($\mathcal{V}$, $\mathcal{A}$). *let k* = *length* $\mathcal{V}$ *in if k* < *2^63−1 then RETURN* (*Allocated*, *length-uint64-nat* $\mathcal{V}$) *else RETURN* (*Mem-Out*, *0*))›
unfolding *find-new-idx-c-def Let-def* **by** *auto*

sepref-definition *find-new-idx-c-impl*
is ‹*find-new-idx-c*›
:: ‹*perfect-shared-vars-assn*^{k} $\rightarrow_a$ *id-assn* $\times_a$ *uint64-nat-assn*›
supply [*simp*] = *uint64-max-def*
unfolding *find-new-idx-c-alt-def pow-2-63-1 zero-uint64-nat-def*[*symmetric*]
by *sepref*

instantiation *String.literal* :: *default*
begin
definition *default-literal* :: ‹*String.literal*› **where**
‹*default-literal* = *String.implode* *''''*›
instance
..
end

sepref-definition *insert-variable-c-impl*
is ‹*uncurry2* (*RETURN ooo insert-variable-c*)›
:: ‹*string-assn*^{k} $*_a$ *uint64-nat-assn*^{k} $*_a$ *perfect-shared-vars-assn*^{d} $\rightarrow_a$ *perfect-shared-vars-assn*›
supply *arl-append-hnr*[*sepref-fr-rules*]
marl-append-hnr[*sepref-fr-rules del*]
unfolding *insert-variable-c-def*
by *sepref*

lemmas [*sepref-fr-rules*] =
find-new-idx-c-impl.refine insert-variable-c-impl.refine

sepref-definition *import-variable-c-impl*
is ‹*uncurry import-variable-c*›
:: ‹*string-assn*^{k} $*_a$ *perfect-shared-vars-assn*^{d} $\rightarrow_a$ *id-assn* $\times_a$ *perfect-shared-vars-assn* $\times_a$ *uint64-nat-assn*›
unfolding *import-variable-c-def*
by *sepref*

lemma *import-variable-c-import-variableS'*:
assumes ‹*single-valued R*› ‹*single-valued* (R^{-1})›
shows ‹(*uncurry import-variable-c*, *uncurry import-variableS*) $\in$ *R* $\times_r$ *perfect-shared-vars-rel-c R* $\rightarrow_f$ ⟨*memory-allocation-rel* $\times_r$ *perfect-shared-vars-rel-c R* $\times_r$ *nat-rel*⟩*nres-rel*›
using *import-variable-c-import-variableS*[*OF - - assms*]
by (*auto intro*!: *frefI nres-relI*)

lemma [*sepref-fr-rules*]:
‹(*uncurry import-variable-c-impl*, *uncurry import-variableS*)
$\in$ *string-assn*^{k} $*_a$ *shared-vars-assn*^{d} $\rightarrow_a$ *memory-allocation-assn* $\times_a$ *shared-vars-assn* $\times_a$ *uint64-nat-assn*›

using *import-variable-c-impl.refine*[*FCOMP import-variable-c-import-variableS′*, *of Id*]
by *auto*

definition *empty-shared-vars* :: ‹(*nat*, *string*) *shared-vars*› **where**
‹*empty-shared-vars* = ({#}, *fmempty*, *fmempty*)›

definition *empty-shared-vars-int* :: ‹(*string*, *nat*) *shared-vars-c*› **where**
‹*empty-shared-vars-int* = ([], *fmempty*)›

sepref-definition *empty-shared-vars-int-impl*
is ‹*uncurry0* (*RETURN empty-shared-vars-int*)›
:: ‹$unit\text{-}assn^k \rightarrow_a$ *perfect-shared-vars-assn*›
unfolding *empty-shared-vars-int-def*
arl.fold-custom-empty
by *sepref*

lemma *empty-shared-vars-int-empty-shared-vars*:
‹(*uncurry0* (*RETURN empty-shared-vars-int*), *uncurry0* (*RETURN empty-shared-vars*)) ∈ *unit-rel* $\rightarrow_f$ ⟨*perfect-shared-vars-rel-c R*⟩*nres-rel*›
by (*auto intro*!: *frefI nres-relI simp*: *perfect-shared-vars-rel-c-def empty-shared-vars-int-def*
empty-shared-vars-def)

lemma [*sepref-fr-rules*]:
‹(*uncurry0 empty-shared-vars-int-impl*, *uncurry0* (*RETURN empty-shared-vars*))
∈ $unit\text{-}assn^k \rightarrow_a$ *shared-vars-assn*›
using *empty-shared-vars-int-impl.refine*[*FCOMP empty-shared-vars-int-empty-shared-vars*, *of Id*]
by *auto*
sepref-register *empty-shared-vars*
end
theory *EPAC-Efficient-Checker-Synthesis*
imports *EPAC-Efficient-Checker*
EPAC-Perfectly-Shared-Vars
PAC-Checker.PAC-Checker-Synthesis
EPAC-Steps-Refine
PAC-Checker.PAC-Checker-Synthesis
begin

lemma *in-set-rel-inD*: ‹(*x*,*y*) ∈⟨*R*⟩*list-rel* ⟹ *a* ∈ *set x* ⟹ ∃ *b* ∈ *set y*. (*a*,*b*)∈ *R*›
by (*metis* (*no-types*, *lifting*) *Un-iff list.set-intros*(*1*) *list-relE3 list-rel-append1 set-append split-list-first*)

lemma *perfectly-shared-monom-eqD*: ‹(*a*, *ab*) ∈ *perfectly-shared-monom* 𝒱 ⟹ *ab* = *map* ((*the* ∘∘ *fm-lookup*) (*fst* (*snd* 𝒱))) *a*›
by (*induction a arbitrary*: *ab*)
(*auto simp*: *append-eq-append-conv2 append-eq-Cons-conv Cons-eq-append-conv*
list-rel-append1 list-rel-split-right-iff perfectly-shared-var-rel-def br-def)

lemma *perfectly-shared-monom-unique-left*:
‹(*x*, *y*) ∈ *perfectly-shared-monom* 𝒱 ⟹ (*x*, *y′*) ∈ *perfectly-shared-monom* 𝒱 ⟹ *y* = *y′*›
using *perfectly-shared-monom-eqD* **by** *blast*

lemma *perfectly-shared-monom-unique-right*:
‹(𝒱, 𝒟𝒱) ∈ *perfectly-shared-vars-rel* ⟹
(*x*, *y*) ∈ *perfectly-shared-monom* 𝒱 ⟹ (*x′*, *y*) ∈ *perfectly-shared-monom* 𝒱 ⟹ *x* = *x′*›
by (*induction x arbitrary*: *x′ y*)

(*auto simp*: *append-eq-append-conv2 append-eq-Cons-conv Cons-eq-append-conv list-rel-split-left-iff perfectly-shared-vars-rel-def perfectly-shared-vars-def list-rel-append1 list-rel-split-right-iff perfectly-shared-var-rel-def br-def add-mset-eq-add-mset*
dest!: *multi-member-split*[*of* - ‹*dom-m* -›])

lemma *perfectly-shared-polynom-unique-left*:
‹$(x, y) \in$ *perfectly-shared-polynom* $\mathcal{V} \Longrightarrow (x, y') \in$ *perfectly-shared-polynom* $\mathcal{V} \Longrightarrow y = y'$›
by (*induction x arbitrary*: *y y'*)
(*auto dest*: *perfectly-shared-monom-unique-left simp*: *list-rel-split-right-iff*)
lemma *perfectly-shared-polynom-unique-right*:
‹$(\mathcal{V}, \mathcal{D}\mathcal{V}) \in$ *perfectly-shared-vars-rel* $\Longrightarrow$
$(x, y) \in$ *perfectly-shared-polynom* $\mathcal{V} \Longrightarrow (x', y) \in$ *perfectly-shared-polynom* $\mathcal{V} \Longrightarrow x = x'$›
by (*induction x arbitrary*: *x' y*)
(*auto dest*: *perfectly-shared-monom-unique-right simp*: *list-rel-split-left-iff list-rel-split-right-iff*)

definition (**in** −)*perfect-shared-var-order-s* :: ‹(*nat*, *string*)*shared-vars* ⇒ *nat* ⇒ *nat* ⇒ *ordered nres*›
where
‹*perfect-shared-var-order-s* $\mathcal{D}$ *x y* = *do* {
  eq ← *perfectly-shared-strings-equal-l* $\mathcal{D}$ *x y*;
  if eq then RETURN EQUAL
  else do {
    x ← *get-var-nameS* $\mathcal{D}$ *x*;
    y ← *get-var-nameS* $\mathcal{D}$ *y*;
    if $(x, y) \in$ *var-order-rel then RETURN* (*LESS*)
    else RETURN (*GREATER*)
  }}›

lemma *perfect-shared-var-order-s-perfect-shared-var-order*:
assumes ‹$(\mathcal{V}, \mathcal{V}\mathcal{D}) \in$ *perfectly-shared-vars-rel*› **and**
‹$(i, i') \in$ *perfectly-shared-var-rel* $\mathcal{V}$›**and**
‹$(j, j') \in$ *perfectly-shared-var-rel* $\mathcal{V}$›
shows ‹*perfect-shared-var-order-s* $\mathcal{V}$ *i j* ≤⇓*Id* (*perfect-shared-var-order* $\mathcal{V}\mathcal{D}$ *i' j'*)›
proof −
show *?thesis*
unfolding *perfect-shared-var-order-s-def perfect-shared-var-order-def*
apply (*refine-rcg perfectly-shared-strings-equal-l-perfectly-shared-strings-equal get-var-nameS-spec*)
subgoal using *assms* **by** *metis*
subgoal using *assms* **by** *metis*
subgoal using *assms* **by** *metis*
subgoal by *auto*
subgoal using *assms* **by** *auto*
subgoal using *assms* **by** *metis*
subgoal using *assms* **by** *metis*
subgoal using *assms* **by** *metis*
subgoal by *auto*
done
qed

definition (**in** −) *perfect-shared-term-order-rel-s*
:: ‹(*nat*, *string*) *shared-vars* ⇒ *nat list*⇒ *nat list* ⇒ *ordered nres*›
where
‹*perfect-shared-term-order-rel-s* $\mathcal{V}$ *xs ys* = *do* {

$(b, \text{-}, \text{-}) \leftarrow WHILE_T\ (\lambda(b, xs, ys).\ b = UNKNOWN)$
  $(\lambda(b, xs, ys).$ *do* {
    if $xs = [] \wedge ys = []$ *then RETURN* $(EQUAL, xs, ys)$
    else if $xs = []$ *then RETURN* $(LESS, xs, ys)$
    else if $ys = []$ *then RETURN* $(GREATER, xs, ys)$
    else do {
      $ASSERT(xs \neq [] \wedge ys \neq [])$;
      $eq \leftarrow$ *perfect-shared-var-order-s* $\mathcal{V}$ (*hd xs*) (*hd ys*);
      if $eq = EQUAL$ *then RETURN* (*b*, *tl xs*, *tl ys*)
      else RETURN (*eq*, *xs*, *ys*)
    }
  }) (*UNKNOWN*, *xs*, *ys*);
  RETURN b
}›

lemma *perfect-shared-term-order-rel-s-perfect-shared-term-order-rel*:
  assumes ‹$(\mathcal{V}, \mathcal{VD}) \in$ *perfectly-shared-vars-rel*› **and**
  ‹$(xs, xs') \in$ *perfectly-shared-monom* $\mathcal{V}$› **and**
  ‹$(ys, ys') \in$ *perfectly-shared-monom* $\mathcal{V}$›
  shows ‹*perfect-shared-term-order-rel-s* $\mathcal{V}$ *xs ys* $\leq \Downarrow$*Id* (*perfect-shared-term-order-rel* $\mathcal{VD}$ *xs' ys'*)›
  using *assms*
  unfolding *perfect-shared-term-order-rel-s-def perfect-shared-term-order-rel-def*
  apply (*refine-rcg WHILET-refine*[**where** R = ‹*Id* $\times_r$ *perfectly-shared-monom* $\mathcal{V}$ $\times_r$ *perfectly-shared-monom* $\mathcal{V}$›]
    perfect-shared-var-order-s-perfect-shared-var-order)
  subgoal by *auto*
  subgoal by *auto*
  subgoal by *auto*
  subgoal by *auto*
  subgoal by *auto*
  subgoal by *auto*
  subgoal by *auto*
  subgoal by *auto*
  subgoal by (*auto simp*: *neq-Nil-conv*)
  subgoal by (*auto simp*: *neq-Nil-conv*)
  subgoal by *auto*
  subgoal by (*auto simp*: *neq-Nil-conv*)
  subgoal by *auto*
  subgoal by *auto*
  done

fun *mergeR* :: - ⇒ - ⇒ *'a list* ⇒ *'a list* ⇒ *'a list nres*
where
  mergeR Φ *f* (*x#xs*) (*y#ys*) = *do* {
      $ASSERT(\Phi\ x\ y)$;
      $b \leftarrow f\ x\ y$;
      if b then do {$zs \leftarrow$ *mergeR* Φ *f xs* (*y#ys*); *RETURN* (*x # zs*)}
      else do {$zs \leftarrow$ *mergeR* Φ *f* (*x#xs*) *ys*; *RETURN* (*y # zs*)}
    }
| *mergeR* Φ *f xs* [] = *RETURN xs*
| *mergeR* Φ *f* [] *ys* = *RETURN ys*

lemma *mergeR-merge*:
  assumes ‹$\bigwedge x\ y.\ x \in set\ xs \cup set\ ys \Longrightarrow y \in set\ xs \cup set\ ys \Longrightarrow \Phi\ x\ y$› **and**
  ‹$\bigwedge x\ y.\ x \in set\ xs \cup set\ ys \Longrightarrow y \in set\ xs \cup set\ ys \Longrightarrow f\ x\ y \leq \Downarrow Id\ (RETURN\ (f'\ x\ y))$› **and**

```
    ‹(xs,xs')∈Id›and
    ‹(ys,ys')∈Id›
  shows
    ‹mergeR Φ f xs ys ≤ ⇓Id (RETURN (merge f' xs' ys'))›
proof –
  have xs: ‹xs' = xs› ‹ys' = ys›
    using assms
    by auto
  show ?thesis
    using assms(1,2) unfolding xs
    apply (induction f' xs ys arbitrary: xs' ys' rule: merge.induct)
    subgoal for f' x xs y ys
      unfolding mergeR.simps merge.simps
      apply (refine-rcg)
      subgoal by simp
      subgoal premises p
        using p(1,2,3,4,5) p(4)[of x y, simplified]
        apply auto
        apply (smt RES-sng-eq-RETURN insert-compr ireturn-rule nres-order-simps(20) specify-left)
        apply (smt RES-sng-eq-RETURN insert-compr ireturn-rule nres-order-simps(20) specify-left)
        done
      done
    subgoal by auto
    subgoal by auto
    done
qed

lemma merge-alt:
  RETURN (merge f xs ys) = SPEC(λzs. zs = merge f xs ys ∧ set zs = set xs ∪ set ys)
  by (induction f xs ys rule: merge.induct)
   (clarsimp-all simp: Collect-conv-if insert-commute)

fun msortR :: - ⇒ - ⇒ 'a list ⇒ 'a list nres
where
  msortR Φ f [] = RETURN []
| msortR Φ f [x] = RETURN [x]
| msortR Φ f xs = do {
    as ← msortR Φ f (take (size xs div 2) xs);
    bs ← msortR Φ f (drop (size xs div 2) xs);
   mergeR Φ f as bs
  }

lemma set-msort[simp]: ‹set (msort f xs) = set xs›
   by (meson mset-eq-setD mset-msort)

lemma msortR-msort:
  assumes ‹⋀x y. x∈set xs ⟹ y∈set xs ⟹Φ x y› and
    ‹⋀x y. x∈set xs ⟹ y∈set xs ⟹ f x y ≤ ⇓Id (RETURN (f' x y))›
  shows
    ‹msortR Φ f xs ≤ ⇓Id (RETURN (msort f' xs))›
proof –
  have a: ‹set (take (length xs div 2) (y # xs)) ⊆ insert x (insert y (set xs))›
    ‹set (drop (length xs div 2) (y # xs)) ⊆ insert x (insert y (set xs))›
    for x y xs
    by (auto dest: in-set-takeD in-set-dropD)
```

have *H*: ‹*RETURN* (*msort f′* (*x#y#xs*)) = *do* {
let as = *msort f′* (*take* (*size* (*x#y#xs*) *div 2*) (*x#y#xs*));
let bs = *msort f′* (*drop* (*size* (*x#y#xs*) *div 2*) (*x#y#xs*));
ASSERT(*set* (*as*) ⊆ *set* (*x#y#xs*));
ASSERT(*set* (*bs*) ⊆ *set* (*x#y#xs*));
RETURN (*merge f′ as bs*)}› **for** *x y xs f′*
unfolding *Let-def*
by (*auto simp*: *a*)
show *?thesis*
supply *RETURN-as-SPEC-refine*[*refine2 del*]
using *assms*
apply (*induction f′ xs rule*: *msort.induct*)
subgoal by *auto*
subgoal by *auto*
subgoal premises *p* **for** *f′ x y xs*
using *p*
unfolding *msortR.simps H*
apply (*refine-vcg mergeR-merge p*)
subgoal by (*auto dest*!: *in-set-takeD*)
subgoal by (*auto dest*!: *in-set-takeD*)
subgoal by (*auto dest*!: *in-set-takeD*)
subgoal by (*auto dest*!: *in-set-takeD*)
subgoal by (*auto dest*!: *in-set-dropD*)
subgoal by (*auto dest*!: *in-set-dropD*)
subgoal by (*auto dest*!: *in-set-dropD*)
subgoal by (*auto dest*!: *in-set-dropD*)
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
done
done
qed

lemma *merge-list-rel*:
assumes ‹$\bigwedge$*x y x′ y′*. *x*∈*set xs* ⟹ *y*∈*set ys* ⟹ *x′*∈*set xs′* ⟹ *y′*∈*set ys′* ⟹ (*x*,*x′*)∈*R* ⟹ (*y*,*y′*)∈*R* ⟹ *f x y* = *f′ x′ y′*› **and**
‹(*xs*,*xs′*) ∈ ⟨*R*⟩*list-rel*› **and**
‹(*ys*,*ys′*) ∈ ⟨*R*⟩*list-rel*›
shows ‹(*merge f xs ys*, *merge f′ xs′ ys′*) ∈ ⟨*R*⟩*list-rel*›
proof −
show *?thesis*
using *assms*
proof (*induction f′ xs′ ys′ arbitrary*: *f xs ys rule*: *merge.induct*)
case (*1 f′ x′ xs′ y′ y′s*)
have ‹*f′ x′ y′* ⟹
(*PAC-Checker-Init.merge f* (*tl xs*) *ys*, *PAC-Checker-Init.merge f′ xs′* (*y′* # *y′s*)) ∈ ⟨*R*⟩*list-rel*›
apply (*rule 1*)
apply *assumption*
apply (*rule 1*(*3*); *auto dest*: *in-set-tlD*)
using *1*(*4*−*5*) **apply** (*auto simp*: *list-rel-split-left-iff*)
done
moreover have ‹¬*f′ x′ y′* ⟹
(*PAC-Checker-Init.merge f* (*xs*) (*tl ys*), *PAC-Checker-Init.merge f′* (*x′* # *xs′*) (*y′s*)) ∈ ⟨*R*⟩*list-rel*›
apply (*rule 1*)

apply *assumption*
apply (*rule 1*(*3*); *auto dest*: *in-set-tlD*)
using *1*(*4*−*5*) **apply** (*auto simp*: *list-rel-split-left-iff*)
done
ultimately show *?case*
using *1*(*1*,*4*−*5*) *1*(*3*)[*of* ‹*hd xs*› ‹*hd ys*› *x′ y′*]
by (*auto simp*: *list-rel-split-left-iff*)
qed (*auto simp*: *list-rel-split-left-iff*)

qed

lemma *msort-list-rel*:
assumes ‹$\bigwedge$*x y x′ y′*. *x*∈*set xs* ⟹ *y*∈*set xs* ⟹ *x′*∈*set xs′* ⟹ *y′*∈*set xs′* ⟹ (*x*,*x′*)∈*R* ⟹ (*y*,*y′*)∈*R* ⟹ *f x y* = *f′ x′ y′*› **and**
‹(*xs*,*xs′*) ∈ ⟨*R*⟩*list-rel*›
shows ‹(*msort f xs*, *msort f′ xs′*) ∈ ⟨*R*⟩*list-rel*›
proof −
show *?thesis*
using *assms*
proof (*induction f′ xs′ arbitrary*: *xs f rule*: *msort.induct*)
case (*3 f′′ v vb vc*)
have *xs*: ‹
(*msort f* (*take* (*length xs div 2*) *xs*), *msort f′′* (*take* (*length* (*v* # *vb* # *vc*) *div 2*) (*v* # *vb* # *vc*)))
∈ ⟨*R*⟩*list-rel*›
‹(*msort f* (*drop* (*length xs div 2*) *xs*), *msort f′′* (*drop* (*length* (*v* # *vb* # *vc*) *div 2*) (*v* # *vb* # *vc*)))
∈ ⟨*R*⟩*list-rel*›
subgoal
apply (*rule 3*)
using *3*(*3*−) **apply** (*force dest*!: *in-set-dropD in-set-takeD list-rel-imp-same-length*)
using *3*(*4*) **apply** (*auto simp*: *list-rel-imp-same-length dest*: *list-rel-takeD*)
done
subgoal
apply (*rule 3*)
using *3*(*3*−) **apply** (*force dest*!: *in-set-dropD in-set-takeD list-rel-imp-same-length*
dest:)
using *3*(*4*) **apply** (*auto simp*: *list-rel-imp-same-length dest*: *list-rel-dropD*)
done
done
have *H*: ‹(*PAC-Checker-Init.merge f* (*msort f* (*x* # *take* (*length xsaa div 2*) (*xa* # *xsaa*)))
(*msort f* (*drop* (*length xsaa div 2*) (*xa* # *xsaa*))),
PAC-Checker-Init.merge f′′ (*msort f′′* (*v* # *take* (*length vc div 2*) (*vb* # *vc*)))
(*msort f′′* (*drop* (*length vc div 2*) (*vb* # *vc*))))
∈ ⟨*R*⟩*list-rel*›
if ‹*xs* = *x* # *xa* # *xsaa*› **and**
‹ (*x*, *v*) ∈ *R*› **and**
‹(*xa*, *vb*) ∈ *R*› **and**
‹ (*xa*, *vb*) ∈ *R*›
for *x xa xsaa*
apply (*rule merge-list-rel*)
subgoal for *xb y x′ y′*
by (*rule 3*(*3*))
(*use that* **in** ‹*auto dest*: *in-set-takeD in-set-dropD*›)
subgoal
by (*use xs*(*1*) *3*(*4*) *that* **in** *auto*)
subgoal

by (*use xs*(*2*) *3*(*4*) *that* **in** *auto*)
done
show *?case*
using *3*(*3*−) *H* **by** (*auto simp*: *list-rel-split-left-iff*)
qed (*auto simp*: *list-rel-split-left-iff intro*!:)
qed

lemma *msortR-alt-def*:
‹(*msortR* Φ *f xs*) = REC_T(λ*msortR′ xs*.
if length xs ≤ *1 then RETURN xs else do* {
let xs1 = (*take* ((*size xs*) *div 2*) *xs*);
let xs2 = (*drop* ((*size xs*) *div 2*) *xs*);
as ← *msortR′ xs1*;
bs ← *msortR′ xs2*;
(*mergeR* Φ *f as bs*)
}) *xs*
›
apply (*induction* Φ *f xs rule*: *msortR.induct*)
subgoal
by (*subst RECT-unfold*, *refine-mono*) *auto*
subgoal
by (*subst RECT-unfold*, *refine-mono*) *auto*
subgoal
by (*subst RECT-unfold*, *refine-mono*) *auto*
done

definition *sort-poly-spec-s* **where**
‹*sort-poly-spec-s* 𝒱 *xs* = *msortR* (λ*xs ys*. (∀ *a*∈*set* (*fst xs*). *a* ∈# *dom-m* (*fst* (*snd* 𝒱))) ∧ (∀ *a*∈*set*(*fst ys*). *a* ∈# *dom-m* (*fst* (*snd* 𝒱))))
(λ*xs ys*. *do* {*a* ← *perfect-shared-term-order-rel-s* 𝒱 (*fst xs*) (*fst ys*); *RETURN* (*a* ≠ *GREATER*)}) *xs*›

lemma *sort-poly-spec-s-sort-poly-spec*:
assumes ‹(𝒱, 𝒱𝒟) ∈ *perfectly-shared-vars-rel*› **and**
‹(*xs*, *xs′*) ∈ *perfectly-shared-polynom* 𝒱› **and**
‹*vars-llist xs′* ⊆ *set-mset* 𝒱𝒟›
shows
‹*sort-poly-spec-s* 𝒱 *xs*
≤⇓(*perfectly-shared-polynom* 𝒱)
(*sort-poly-spec xs′*)
›
proof −
have [*iff*]: ‹*sorted-wrt* (*rel2p* (*Id* ∪ *term-order-rel*)) (*map fst* (*msort* (λ*xs ys*. *rel2p* (*Id* ∪ *term-order-rel*) (*fst xs*) (*fst ys*)) *xs′*))›
unfolding *sorted-wrt-map*
apply (*rule sorted-msort*)
apply (*smt Un-iff pair-in-Id-conv rel2p-def term-order-rel-trans transp-def*)
apply (*auto simp*: *rel2p-def*)
using *total-on-lexord-less-than-char-linear var-order-rel-def* **by** *auto*
have [*iff*]:
‹(*a*,*b*)∈ ⟨⟨(*perfectly-shared-var-rel* 𝒱)$^{-1}$⟩*list-rel* $\times_r$ *int-rel*⟩*list-rel* ⟷ (*b*,*a*)∈*perfectly-shared-polynom* 𝒱› **for** *a b*
by (*metis converse-Id converse-iff inv-list-rel-eq inv-prod-rel-eq*)

show *?thesis*
unfolding *sort-poly-spec-s-def*
apply (*rule order-trans*[*OF msortR-msort*[**where**
f'=‹ *λxs ys.* (*map* (*the o fmlookup* (*fst* (*snd* $\mathcal{V}$))) (*fst xs*), *map* (*the o fmlookup* (*fst* (*snd* $\mathcal{V}$))) (*fst ys*)) ∈ *Id* ∪ *term-order-rel*›]])
subgoal for *x y*
apply (*cases x*, *cases y*)
using *assms* **by** (*auto simp*: *list-rel-append1 list-rel-split-right-iff perfectly-shared-var-rel-def br-def perfectly-shared-vars-rel-def append-eq-append-conv2 append-eq-Cons-conv Cons-eq-append-conv dest*!: *split-list split*: *prod.splits*)
subgoal for *x y*
using *assms*(*2*,*3*) **apply** −
apply (*frule in-set-rel-inD*)
apply *assumption*
apply (*frule in-set-rel-inD*[*of* - - - *y*])
apply *assumption*
apply (*elim bexE*)+
subgoal for *x' y'*
apply (*refine-vcg perfect-shared-term-order-rel-s-perfect-shared-term-order-rel*[*OF assms*(*1*), *THEN order-trans*,
of - ‹*fst x'*› - ‹*fst y'*›])
subgoal
by (*cases x'*, *cases x*) *auto*
subgoal
by (*cases y'*, *cases y*) *auto*
subgoal
using *assms*
apply (*clarsimp dest*!: *split-list intro*!: *perfect-shared-term-order-rel-spec*[*THEN order-trans*] *simp*: *append-eq-append-conv2 append-eq-Cons-conv Cons-eq-append-conv vars-llist-def*)
apply (*rule perfect-shared-term-order-rel-spec*[*THEN order-trans*])
apply *auto*[]
apply *auto*[]
apply *simp*
apply (*clarsimp-all simp*: *perfectly-shared-monom-eqD*)
apply (*cases x*, *cases y*, *cases x'*, *cases y'*)
apply (*clarsimp-all simp flip*: *perfectly-shared-monom-eqD*)
apply (*case-tac xa*)
apply (*clarsimp-all simp flip*: *perfectly-shared-monom-eqD simp*: *lexord-irreflexive*)
by (*meson lexord-irreflexive term-order-rel-trans var-order-rel-antisym*)
done
done
unfolding *sort-poly-spec-def conc-fun-RES*
apply *auto*
apply (*subst Image-iff*)
apply (*rule-tac x*= ‹*msort* (*λxs ys. rel2p* (*Id* ∪ *term-order-rel*) (*fst xs*) (*fst ys*)) (*xs'*)› **in** *bexI*)
apply (*auto intro*!: *msort-list-rel simp flip*: *perfectly-shared-monom-eqD simp*: *assms*)
apply (*auto simp*: *rel2p-def*)
done
qed

definition *msort-coeff-s* :: ‹(*nat*,*string*)*shared-vars* ⇒ *nat list* ⇒ *nat list nres*› **where**
‹*msort-coeff-s* $\mathcal{V}$ *xs* = *msortR* (*λa b. a* ∈ *set xs* ∧ *b* ∈ *set xs*)
(*λa b. do* {

$x \leftarrow$ *get-var-nameS* $\mathcal{V}$ *a*;
$y \leftarrow$ *get-var-nameS* $\mathcal{V}$ *b*;
RETURN($a = b \vee$ *var-order* x y)
}) *xs*›

lemma *perfectly-shared-var-rel-unique-left*:
‹$(x, y) \in$ *perfectly-shared-var-rel* $\mathcal{V} \Longrightarrow (x, y') \in$ *perfectly-shared-var-rel* $\mathcal{V} \Longrightarrow y = y'$›
using *perfectly-shared-monom-unique-left*[*of* ‹[x]› ‹[y]› $\mathcal{V}$ ‹[y']›] **by** *auto*

lemma *perfectly-shared-var-rel-unique-right*:
‹$(\mathcal{V}, \mathcal{DV}) \in$ *perfectly-shared-vars-rel* $\Longrightarrow (x, y) \in$ *perfectly-shared-var-rel* $\mathcal{V} \Longrightarrow (x', y) \in$ *perfectly-shared-var-rel* $\mathcal{V} \Longrightarrow x = x'$›
using *perfectly-shared-monom-unique-right*[*of* $\mathcal{V}$ $\mathcal{DV}$ ‹[x]› ‹[y]› ‹[x']›]
by *auto*

lemma *msort-coeff-s-sort-coeff*:
fixes *xs'* :: ‹*string list*› **and**
$\mathcal{V}$ :: ‹(*nat*,*string*)*shared-vars*›
assumes
‹(*xs*, *xs'*) $\in$ *perfectly-shared-monom* $\mathcal{V}$› **and**
‹$(\mathcal{V}, \mathcal{DV}) \in$ *perfectly-shared-vars-rel*› **and**
‹*set xs'* $\subseteq$ *set-mset* $\mathcal{DV}$›
shows ‹*msort-coeff-s* $\mathcal{V}$ *xs* $\leq \Downarrow$(*perfectly-shared-monom* $\mathcal{V}$) (*sort-coeff xs'*)›
proof −
have *H*: ‹$x \in$ *set xs* $\Longrightarrow \exists x' \in$ *set xs'*. $(x,x') \in$ *perfectly-shared-var-rel* $\mathcal{V} \wedge x' \in\# \mathcal{DV}$› **for** x
using *assms*(*1*,*3*) **by** (*auto dest*: *in-set-rel-inD*)
define *f* **where**
‹*f x y* $\longleftrightarrow x = y \vee$ *var-order* (*fst* (*snd* $\mathcal{V}$) $\propto x$) (*fst* (*snd* $\mathcal{V}$) $\propto y$)› **for** *x y*
have [*simp*]: ‹$x \in$ *set xs* $\Longrightarrow x' \in$ *set xs'* $\Longrightarrow (x, x') \in$ *perfectly-shared-var-rel* $\mathcal{V} \Longrightarrow$
fst (*snd* $\mathcal{V}$) $\propto x = x'$› **for** *x x'*
using *assms*(*2*)
by (*auto simp*: *perfectly-shared-vars-rel-def perfectly-shared-var-rel-def br-def*)
have [*intro*]: ‹*transp* ($\lambda x\ y.\ x = y \vee (x, y) \in$ *var-order-rel*)›
by (*smt transE trans-var-order-rel transp-def*)
have [*intro*]: ‹*sorted-wrt* (*rel2p* (*Id* $\cup$ *var-order-rel*)) (*msort* ($\lambda a\ b.\ a = b \vee$ *var-order a b*) *xs'*)›
using *var-roder-rel-total* **by** (*auto intro*!: *sorted-msort simp*: *rel2p-def*[*abs-def*])
show *?thesis*
unfolding *msort-coeff-s-def*
apply (*rule msortR-msort*[*of* - - - *f*, *THEN order-trans*])
subgoal by *auto*
subgoal for *x y*
unfolding *f-def*
apply (*frule H*[*of x*])
apply (*frule H*[*of y*])
apply (*elim bexE*)
apply (*refine-vcg get-var-nameS-spec2*[*THEN order-trans*] *assms*)
apply (*solves auto*)
apply (*solves auto*)
apply (*subst Down-id-eq*)
apply (*refine-vcg get-var-nameS-spec2*[*THEN order-trans*] *assms*)
apply (*solves auto*)
apply (*solves auto*)
apply (*auto simp*: *perfectly-shared-var-rel-def br-def*)
done

subgoal
apply (*subst Down-id-eq*)
apply (*auto simp*: *sort-coeff-def intro*!: *RETURN-RES-refine*)
apply (*rule-tac x* = ‹*msort* (λ*a b*. *a* = *b* ∨ *var-order a b*) *xs*′› **in** *exI*)
apply (*force intro*!: *msort-list-rel assms simp*: *f-def*
dest: *perfectly-shared-var-rel-unique-left*
perfectly-shared-var-rel-unique-right[*OF assms*(*2*)])
done
done
qed

type-synonym *sllist-polynomial* = ‹(*nat list* × *int*) *list*›

definition *sort-all-coeffs-s* :: ‹(*nat*,*string*)*shared-vars* ⇒ *sllist-polynomial* ⇒ *sllist-polynomial nres*› **where**
‹*sort-all-coeffs-s* 𝒱 *xs* = *monadic-nfoldli xs* (λ-. *RETURN True*) (λ(*a*, *n*) *b*. *do* {*ASSERT*((*a*,*n*)∈*set xs*);*a* ← *msort-coeff-s* 𝒱 *a*; *RETURN* ((*a*, *n*) # *b*)}) []›

fun *merge-coeffs0-s* :: ‹*sllist-polynomial* ⇒ *sllist-polynomial*› **where**
‹*merge-coeffs0-s*[] = []› |
‹*merge-coeffs0-s* [(*xs*, *n*)] = (*if n* = *0 then* [] *else* [(*xs*, *n*)])› |
‹*merge-coeffs0-s* ((*xs*, *n*) # (*ys*, *m*) # *p*) =
(*if xs* = *ys*
then if n + *m* ≠ *0 then merge-coeffs0-s* ((*xs*, *n* + *m*) # *p*) *else merge-coeffs0-s p*
else if n = *0 then merge-coeffs0-s* ((*ys*, *m*) # *p*)
else(*xs*, *n*) # *merge-coeffs0-s* ((*ys*, *m*) # *p*)))›

lemma *merge-coeffs0-s-merge-coeffs0*:
fixes *xs* :: ‹*sllist-polynomial*› **and**
𝒱 :: ‹(*nat*,*string*)*shared-vars*›
assumes
‹(*xs*, *xs*′) ∈ *perfectly-shared-polynom* 𝒱› **and**
𝒱: ‹(𝒱, 𝒟𝒱) ∈ *perfectly-shared-vars-rel*›
shows ‹(*merge-coeffs0-s xs*, *merge-coeffs0 xs*′) ∈ *perfectly-shared-polynom* 𝒱›
using *assms*
apply (*induction xs*′ *arbitrary*: *xs rule*: *merge-coeffs0.induct*)
subgoal by *auto*
subgoal by (*auto simp*: *list-rel-split-left-iff*)
subgoal premises *p* **for** *xs n ys m p xsa*
using *p*(*1*)[*of* ‹(-, - + -) # *tl* (*tl xsa*)›] *p*(*2*)[*of* ‹*tl* (*tl xsa*)›] *p*(*3*)[*of* ‹*tl xsa*›] *p*(*4*)[*of* ‹*tl xsa*›] *p*(*5*−)
using *perfectly-shared-monom-unique-right*[*OF* 𝒱, *of* - *xs*]
perfectly-shared-monom-unique-left[*of* ‹*fst* (*hd xsa*)› - 𝒱]
apply (*auto 4 1 simp*: *list-rel-split-left-iff*
dest:)
apply *smt*
done
done

lemma *list-rel-mono-strong*: ‹*A* ∈ ⟨*R*⟩*list-rel* ⟹ (⋀*xs*. *fst xs* ∈ *set* (*fst A*) ⟹ *snd xs* ∈ *set* (*snd A*) ⟹ *xs* ∈ *R* ⟹ *xs* ∈ *R*′) ⟹ *A* ∈ ⟨*R*′⟩*list-rel*›
unfolding *list-rel-def*
apply (*cases A*)
apply (*simp add*: *list.rel-mono-strong*)
done

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

‹*full-normalize-poly-s* $\mathcal{V}$ *p* = *do* {
  p ← *sort-all-coeffs-s* $\mathcal{V}$ *p*;
  p ← *sort-poly-spec-s* $\mathcal{V}$ *p*;
  RETURN (*merge-coeffs0-s p*)
}›

lemma *sort-all-coeffs-s-sort-all-coeffs*:
  fixes *xs* :: ‹*sllist-polynomial*› **and**
    $\mathcal{V}$ :: ‹(*nat*,*string*)*shared-vars*›
  assumes
    ‹(*xs*, *xs'*) ∈ *perfectly-shared-polynom* $\mathcal{V}$› **and**
    $\mathcal{V}$: ‹($\mathcal{V}$, $\mathcal{DV}$) ∈ *perfectly-shared-vars-rel*› **and**
    ‹*vars-llist xs'* ⊆ *set-mset* $\mathcal{DV}$›
  shows ‹*sort-all-coeffs-s* $\mathcal{V}$ *xs* ≤ ⇓(*perfectly-shared-polynom* $\mathcal{V}$) (*sort-all-coeffs xs'*)›
proof −
  have [*refine*]: ‹(*xs*, *xs'*) ∈ ⟨{(*a*,*b*). *a*∈*set xs* ∧ (*a*,*b*)∈ *perfectly-shared-monom* $\mathcal{V}$ $\times_r$ *int-rel*}⟩*list-rel*›
    by (*rule list-rel-mono-strong*[*OF assms*(*1*)])
     (*use assms*(*3*) **in** *auto*)

  show *?thesis*
    unfolding *sort-all-coeffs-s-def sort-all-coeffs-def*
    apply (*refine-vcg* $\mathcal{V}$ *msort-coeff-s-sort-coeff*)
    subgoal by *auto*
    subgoal by *auto*
    subgoal by *auto*
    subgoal using *assms* **by** (*auto dest*!: *split-list*)
    subgoal by *auto*
    done
qed

definition *vars-llist-in-s* :: ‹(*nat*, *string*) *shared-vars* ⇒ *llist-polynomial* ⇒ *bool*› **where**
  ‹*vars-llist-in-s* = (λ($\mathcal{V}$,$\mathcal{D}$,$\mathcal{D}'$) *p*. *vars-llist p* ⊆ *set-mset* (*dom-m* $\mathcal{D}'$))›

lemma *vars-llist-in-s-vars-llist*[*simp*]:
  assumes ‹($\mathcal{V}$, $\mathcal{DV}$) ∈ *perfectly-shared-vars-rel*›
  shows ‹*vars-llist-in-s* $\mathcal{V}$ *p* ⟷ *vars-llist p* ⊆ *set-mset* $\mathcal{DV}$›
  using *assms* **unfolding** *perfectly-shared-vars-rel-def perfectly-shared-vars-def vars-llist-in-s-def*
  by *auto*

definition (**in** −)*add-poly-l-s* :: ‹(*nat*,*string*)*shared-vars* ⇒ *sllist-polynomial* × *sllist-polynomial* ⇒ *sllist-polynomial nres*› **where**
  ‹*add-poly-l-s* $\mathcal{D}$ = REC_T
  (λ*add-poly-l* (*p*, *q*).
  case (*p*,*q*) *of*
    (*p*, []) ⇒ *RETURN p*
    | ([], *q*) ⇒ *RETURN q*
    | ((*xs*, *n*) # *p*, (*ys*, *m*) # *q*) ⇒ *do* {
    comp ← *perfect-shared-term-order-rel-s* $\mathcal{D}$ *xs ys*;
    if comp = *EQUAL then if n* + *m* = *0 then add-poly-l* (*p*, *q*)
    else do {
      pq ← *add-poly-l* (*p*, *q*);
      RETURN ((*xs*, *n* + *m*) # *pq*)
    }
    else if comp = *LESS*

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*)
}
})›

lemma *add-poly-l-s-add-poly-l*:
fixes *xs* :: ‹*sllist-polynomial* × *sllist-polynomial*›
assumes ‹($\mathcal{V}$, $\mathcal{VD}$) ∈ *perfectly-shared-vars-rel*› **and**
‹(*xs*, *xs'*) ∈ *perfectly-shared-polynom* $\mathcal{V}$ $\times_r$ *perfectly-shared-polynom* $\mathcal{V}$›
shows ‹*add-poly-l-s* $\mathcal{V}$ *xs* ≤ ⇓(*perfectly-shared-polynom* $\mathcal{V}$) (*add-poly-l-prep* $\mathcal{VD}$ *xs'*)›
proof −
have *x*: ‹*x* ∈ ⟨*perfectly-shared-monom* $\mathcal{V}$ $\times_r$ *int-rel*⟩*list-rel* ⟹ *x* ∈ ⟨*perfectly-shared-monom* $\mathcal{V}$ $\times_r$ *int-rel*⟩*list-rel*› **for** *x*
by *auto*
show *?thesis*
unfolding *add-poly-l-s-def add-poly-l-prep-def*
apply (*refine-rcg assms perfect-shared-term-order-rel-s-perfect-shared-term-order-rel*)
apply (*rule x*)
subgoal by *auto*
apply (*rule x*)
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
apply (*rule x*)
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*

by (*auto*)
qed

definition (**in** −) *mult-monoms-s* :: ‹(*nat*,*string*)*shared-vars* ⇒ *nat list* ⇒ *nat list* ⇒ *nat list nres*›
where
‹*mult-monoms-s* $\mathcal{D}$ *xs ys* = REC_T (λ*f* (*xs*, *ys*).
do {
  if xs = [] *then RETURN ys*
  else if ys = [] *then RETURN xs*
  else do {
    ASSERT(*xs* ≠ [] ∧ *ys* ≠ []);
    comp ← *perfect-shared-var-order-s* $\mathcal{D}$ (*hd xs*) (*hd ys*);
    if comp = *EQUAL then do* {
      pq ← *f* (*tl xs*, *tl ys*);
      RETURN (*hd xs* # *pq*)
    }
    else if comp = *LESS then do* {

```
      pq ← f (tl xs, ys);
      RETURN (hd xs # pq)
    }
    else do {
      pq ← f (xs, tl ys);
      RETURN (hd ys # pq)
    }
  }
}) (xs, ys)›

lemma mult-monoms-s-simps:
 ‹mult-monoms-s 𝒱 xs ys =
 do {
   if xs = [] then RETURN ys
   else if ys = [] then RETURN xs
   else do {
     ASSERT(xs ≠ [] ∧ ys ≠ []);
     comp ← perfect-shared-var-order-s 𝒱 (hd xs) (hd ys);
     if comp = EQUAL then do {
       pq ← mult-monoms-s 𝒱 (tl xs) (tl ys);
       RETURN (hd xs # pq)
     }
     else if comp = LESS then do {
       pq ← mult-monoms-s 𝒱 (tl xs) ys;
       RETURN (hd xs # pq)
     }
     else do {
       pq ← mult-monoms-s 𝒱 xs (tl ys);
       RETURN (hd ys # pq)
     }
   }
 }›
  apply (subst mult-monoms-s-def)
  apply (subst RECT-unfold, refine-mono)
  unfolding prod.case[of - ‹(xs,ys)›]
  apply (subst mult-monoms-s-def[symmetric])+
  apply (auto intro!: bind-cong[OF refl])
  done

lemma mult-monoms-s-mult-monoms-prep:
  fixes xs
  assumes ‹(𝒱, 𝒱𝒟) ∈ perfectly-shared-vars-rel› and
    ‹(xs, xs') ∈ perfectly-shared-monom 𝒱›
    ‹(ys, ys') ∈ perfectly-shared-monom 𝒱›
  shows ‹mult-monoms-s 𝒱 xs ys ≤ ⇓(perfectly-shared-monom 𝒱) ((mult-monoms-prep 𝒱𝒟 xs' ys'))›
proof -
  have [refine]: ‹((xs, ys), xs', ys') ∈ perfectly-shared-monom 𝒱 ×r perfectly-shared-monom 𝒱›
    using assms by auto
  have x: ‹a ≤ ⇓ (perfectly-shared-monom 𝒱) b ⟹ a ≤ ⇓ (perfectly-shared-monom 𝒱) b› for a b
    by auto
  show ?thesis
    using assms unfolding mult-monoms-s-def mult-monoms-prep-def
    apply (refine-vcg perfect-shared-var-order-s-perfect-shared-var-order)
    subgoal by auto
    subgoal by auto
```

subgoal by *auto*
subgoal by *auto*
subgoal by (*auto simp*: *neq-Nil-conv*)
subgoal by (*auto simp*: *neq-Nil-conv*)
subgoal by *auto*
apply (*rule x*)
subgoal by (*auto simp*: *neq-Nil-conv*)
subgoal by (*auto simp*: *neq-Nil-conv*)
subgoal by *auto*
apply (*rule x*)
subgoal by (*auto simp*: *neq-Nil-conv*)
subgoal by (*auto simp*: *neq-Nil-conv*)
apply (*rule x*)
subgoal by (*auto simp*: *neq-Nil-conv*)
subgoal by (*auto simp*: *neq-Nil-conv*)
done
qed

definition (**in** −) *mult-term-s*
:: ‹(*nat*,*string*)*shared-vars*⇒ *sllist-polynomial* ⇒ - ⇒ *sllist-polynomial* ⇒ *sllist-polynomial nres*›
where
‹*mult-term-s* = ($\lambda\mathcal{V}$ *qs* (*p*, *m*) *b*. *nfoldli qs* (λ-. *True*) (λ(*q*, *n*) *b*. *do* {*pq* ← *mult-monoms-s* $\mathcal{V}$ *p q*; *RETURN* ((*pq*, *m* ∗ *n*) # *b*)}) *b*)›

definition *mult-poly-s* :: ‹(*nat*,*string*) *shared-vars*⇒ *sllist-polynomial* ⇒ *sllist-polynomial* ⇒ *sllist-polynomial nres*› **where**
‹*mult-poly-s* $\mathcal{V}$ *p q* = *nfoldli p* (λ-. *True*) (*mult-term-s* $\mathcal{V}$ *q*) []›

lemma *mult-term-s-mult-monoms-prop*:
fixes *xs*
assumes ‹($\mathcal{V}$, $\mathcal{VD}$) ∈ *perfectly-shared-vars-rel*› **and**
‹(*xs*, *xs*′) ∈ *perfectly-shared-polynom* $\mathcal{V}$›
‹(*ys*, *ys*′) ∈ *perfectly-shared-monom* $\mathcal{V}$ $\times_r$ *int-rel*›
‹(*zs*, *zs*′) ∈ *perfectly-shared-polynom* $\mathcal{V}$›
shows ‹*mult-term-s* $\mathcal{V}$ *xs ys zs* ≤ ⇓(*perfectly-shared-polynom* $\mathcal{V}$) (*mult-monoms-prop* $\mathcal{VD}$ *xs*′ *ys*′ *zs*′)›
proof −
show *?thesis*
using *assms*
unfolding *mult-term-s-def mult-monoms-prop-def*
by (*refine-rcg mult-monoms-s-mult-monoms-prep*)
auto
qed

lemma *mult-poly-s-mult-poly-raw-prop*:
fixes *xs*
assumes ‹($\mathcal{V}$, $\mathcal{VD}$) ∈ *perfectly-shared-vars-rel*› **and**
‹(*xs*, *xs*′) ∈ *perfectly-shared-polynom* $\mathcal{V}$›
‹(*ys*, *ys*′) ∈ *perfectly-shared-polynom* $\mathcal{V}$›
shows ‹*mult-poly-s* $\mathcal{V}$ *xs ys* ≤ ⇓(*perfectly-shared-polynom* $\mathcal{V}$) (*mult-poly-raw-prop* $\mathcal{VD}$ *xs*′ *ys*′)›
proof −
show *?thesis*
using *assms*
unfolding *mult-poly-s-def mult-poly-raw-prop-def*
by (*refine-rcg mult-term-s-mult-monoms-prop*)

auto
qed

lemma *op-eq-uint64-nat*[*sepref-fr-rules*]:
‹(*uncurry* (*return oo* ((=) :: *uint64* ⇒ -)), *uncurry* (*RETURN oo* (=))) ∈
uint64-nat-assn^k $*_a$ *uint64-nat-assn*^{k} $\rightarrow_a$ *bool-assn*›
by *sepref-to-hoare* (*sep-auto simp*: *uint64-nat-rel-def br-def*)

abbreviation *ordered-assn* :: ‹*ordered* ⇒ - ⇒ -› **where**
‹*ordered-assn* ≡ *id-assn*›

lemma *op-eq-ordered-assn*[*sepref-fr-rules*]:
‹(*uncurry* (*return oo* ((=) :: *ordered* ⇒ -)), *uncurry* (*RETURN oo* (=))) ∈
ordered-assn^k $*_a$ *ordered-assn*^{k} $\rightarrow_a$ *bool-assn*›
by *sepref-to-hoare* (*sep-auto simp*: *uint64-nat-rel-def br-def*)

abbreviation *monom-s-rel* **where**
‹*monom-s-rel* ≡ ⟨*uint64-nat-rel*⟩*list-rel*›

abbreviation *monom-s-assn* **where**
‹*monom-s-assn* ≡ *list-assn uint64-nat-assn*›

abbreviation *poly-s-assn* **where**
‹*poly-s-assn* ≡ *list-assn* (*monom-s-assn* $\times_a$ *int-assn*)›

sepref-decl-intf *wordered* **is** *ordered*

sepref-register *EQUAL LESS GREATER UNKNOWN get-var-nameS perfect-shared-var-order-s perfect-shared-term-ord*
lemma [*sepref-fr-rules*]:
‹(*uncurry0* (*return EQUAL*), *uncurry0* (*RETURN EQUAL*)) ∈ *unit-assn*^{k} $\rightarrow_a$ *id-assn*›
‹(*uncurry0* (*return LESS*), *uncurry0* (*RETURN LESS*)) ∈ *unit-assn*^{k} $\rightarrow_a$ *id-assn*›
‹(*uncurry0* (*return GREATER*), *uncurry0* (*RETURN GREATER*)) ∈ *unit-assn*^{k} $\rightarrow_a$ *id-assn*›
‹(*uncurry0* (*return UNKNOWN*), *uncurry0* (*RETURN UNKNOWN*)) ∈ *unit-assn*^{k} $\rightarrow_a$ *id-assn*›
by (*sepref-to-hoare*; *sep-auto*)+

sepref-definition *perfect-shared-var-order-s-impl*
is ‹*uncurry2 perfect-shared-var-order-s*›
:: ‹*shared-vars-assn*^{k} $*_a$ *uint64-nat-assn*^{k} $*_a$ *uint64-nat-assn*^{k} $\rightarrow_a$ *id-assn*›
unfolding *perfect-shared-var-order-s-def perfectly-shared-strings-equal-l-def*
term-order-rel′-def[*symmetric*]
term-order-rel′-alt-def
var-order-rel″
by *sepref*

lemmas [*sepref-fr-rules*] = *perfect-shared-var-order-s-impl.refine*

sepref-definition *perfect-shared-term-order-rel-s-impl*
is ‹*uncurry2 perfect-shared-term-order-rel-s*›
:: ‹*shared-vars-assn*^{k} $*_a$ *monom-s-assn*^{k} $*_a$ *monom-s-assn*^{k} $\rightarrow_a$ *id-assn*›
unfolding *perfect-shared-term-order-rel-s-def*
by *sepref*

lemmas [*sepref-fr-rules*] = *perfect-shared-term-order-rel-s-impl.refine*

sepref-definition *add-poly-l-prep-impl*
  is ‹*uncurry add-poly-l-s*›
  :: ‹$shared\text{-}vars\text{-}assn^k *_a (poly\text{-}s\text{-}assn \times_a poly\text{-}s\text{-}assn)^k \rightarrow_a poly\text{-}s\text{-}assn$›
  unfolding *add-poly-l-s-def*
    HOL-list.fold-custom-empty
    term-order-rel′-def[*symmetric*]
    term-order-rel′-alt-def
  by *sepref*

lemma [*sepref-fr-rules*]:
  ‹(*return o is-Nil*, *RETURN o is-Nil*)∈ $(list\text{-}assn\ R)^k \rightarrow_a bool\text{-}assn$›
  by (*sepref-to-hoare*)
    (*sep-auto split*: *list.splits*)

sepref-definition *mult-monoms-s-impl*
  is ‹*uncurry2 mult-monoms-s*›
  :: ‹$shared\text{-}vars\text{-}assn^k *_a monom\text{-}s\text{-}assn^k *_a monom\text{-}s\text{-}assn^k \rightarrow_a monom\text{-}s\text{-}assn$›
  unfolding *mult-monoms-s-def conv-to-is-Nil*
  unfolding
    HOL-list.fold-custom-empty
    term-order-rel′-def[*symmetric*]
    term-order-rel′-alt-def
  by *sepref*

lemmas [*sepref-fr-rules*] =
  mult-monoms-s-impl.refine

sepref-definition *mult-term-s-impl*
  is ‹*uncurry3 mult-term-s*›
  :: ‹$shared\text{-}vars\text{-}assn^k *_a poly\text{-}s\text{-}assn^k *_a (monom\text{-}s\text{-}assn \times_a int\text{-}assn)^k *_a poly\text{-}s\text{-}assn^k \rightarrow_a poly\text{-}s\text{-}assn$›
  unfolding *mult-term-s-def conv-to-is-Nil*
  unfolding
    HOL-list.fold-custom-empty
    term-order-rel′-def[*symmetric*]
    term-order-rel′-alt-def
  by *sepref*

lemmas [*sepref-fr-rules*] =
  mult-term-s-impl.refine

sepref-definition *mult-poly-s-impl*
  is ‹*uncurry2 mult-poly-s*›
  :: ‹$shared\text{-}vars\text{-}assn^k *_a poly\text{-}s\text{-}assn^k *_a poly\text{-}s\text{-}assn^k \rightarrow_a poly\text{-}s\text{-}assn$›
  unfolding *mult-poly-s-def conv-to-is-Nil*
  unfolding
    HOL-list.fold-custom-empty
  by *sepref*

lemmas [*sepref-fr-rules*] =
  mult-poly-s-impl.refine

sepref-register *take drop*
lemma [*sepref-fr-rules*]:

```
  assumes ‹CONSTRAINT is-pure R›
  shows ‹(uncurry (return oo take), uncurry (RETURN oo take)) ∈ nat-assn^k *a (list-assn R)^k →a
list-assn R›
  apply sepref-to-hoare
  using assms unfolding is-pure-conv CONSTRAINT-def
  apply (sep-auto simp add: list-assn-pure-conv)
  apply (sep-auto simp: pure-def list-rel-takeD)
  done

lemma [sepref-fr-rules]:
  assumes ‹CONSTRAINT is-pure R›
  shows ‹(uncurry (return oo drop), uncurry (RETURN oo drop)) ∈ nat-assn^k *a (list-assn R)^k →a
list-assn R›
  apply sepref-to-hoare
  using assms unfolding is-pure-conv CONSTRAINT-def
  apply (sep-auto simp add: list-assn-pure-conv)
  apply (sep-auto simp: pure-def list-rel-dropD)
  done

definition mergeR-vars :: ‹(nat, string) shared-vars ⇒ sllist-polynomial ⇒ sllist-polynomial ⇒ sllist-polynomial
nres› where
  ‹mergeR-vars 𝒱 = mergeR
    (λxs ys. (∀a∈set (fst xs). a ∈# dom-m (fst (snd 𝒱))) ∧ (∀a∈set(fst ys). a ∈# dom-m (fst (snd
𝒱))))
      (λxs ys. do {a ← perfect-shared-term-order-rel-s 𝒱 (fst xs) (fst ys); RETURN (a ≠ GREATER)})›
lemma mergeR-alt-def:
  ‹(mergeR Φ f xs ys) = REC_T(λmergeR xs.
  case xs of
    ([], ys) ⇒ RETURN ys
  | (xs, []) ⇒ RETURN xs
  | (x # xs, y # ys) ⇒ do {
     ASSERT(Φ x y);
      b ← f x y;
     if b then do {
        zs ← mergeR (xs, y # ys);
        RETURN (x # zs)
     }
     else do {
      zs ← mergeR (x # xs, ys);
      RETURN (y # zs)
     }
  })
  (xs, ys)›
 apply (induction Φ f xs ys rule: mergeR.induct)
 subgoal
   apply (subst RECT-unfold, refine-mono)
   apply (simp add:)
   apply (rule bind-cong[OF refl])+
   apply auto
   done
 subgoal
   by (subst RECT-unfold, refine-mono)
    (simp split: list.splits)
 subgoal
   by (subst RECT-unfold, refine-mono) auto
```

done

sepref-definition *mergeR-vars-impl*
is ⟨*uncurry2 mergeR-vars*⟩
:: ⟨*shared-vars-assn*^{k} $*_a$ *poly-s-assn*^{k} $*_a$ *poly-s-assn*^{k} $\rightarrow_a$ *poly-s-assn*⟩
supply [[*goals-limit* = *1*]]
unfolding *mergeR-vars-def mergeR-alt-def*
by *sepref*

lemmas [*sepref-fr-rules*] =
mergeR-vars-impl.refine

abbreviation *msortR-vars* **where**
⟨*msortR-vars* ≡ *sort-poly-spec-s*⟩
lemmas *msortR-vars-def* = *sort-poly-spec-s-def*

sepref-register *mergeR-vars msortR-vars*

sepref-definition *msortR-vars-impl*
is ⟨*uncurry msortR-vars*⟩
:: ⟨*shared-vars-assn*^{k} $*_a$ *poly-s-assn*^{k} $\rightarrow_a$ *poly-s-assn*⟩
supply [[*goals-limit* = *1*]]
unfolding *msortR-vars-def msortR-alt-def mergeR-vars-def*[*symmetric*]
by *sepref*

lemmas [*sepref-fr-rules*] =
msortR-vars-impl.refine

fun *merge-coeffs-s* :: ⟨*sllist-polynomial* ⇒ *sllist-polynomial*⟩ **where**
⟨*merge-coeffs-s* [] = []⟩ |
⟨*merge-coeffs-s* [(*xs*, *n*)] = [(*xs*, *n*)]⟩ |
⟨*merge-coeffs-s* ((*xs*, *n*) # (*ys*, *m*) # *p*) =
(*if xs* = *ys*
then if n + *m* ≠ *0 then merge-coeffs-s* ((*xs*, *n* + *m*) # *p*) *else merge-coeffs-s p*
else (*xs*, *n*) # *merge-coeffs-s* ((*ys*, *m*) # *p*))⟩

lemma *perfectly-shared-merge-coeffs-merge-coeffs*:
assumes
⟨($\mathcal{V}$, $\mathcal{DV}$) ∈ *perfectly-shared-vars-rel*⟩
⟨(*xs*, *xs*′) ∈ *perfectly-shared-polynom* $\mathcal{V}$⟩
shows ⟨(*merge-coeffs-s xs*, *merge-coeffs xs*′) ∈ (*perfectly-shared-polynom* $\mathcal{V}$)⟩
using *assms*
apply (*induction xs arbitrary*: *xs*′ *rule*: *merge-coeffs-s.induct*)
subgoal
by *auto*
subgoal
by (*auto simp*: *list-rel-split-right-iff*)
subgoal
by(*auto simp*: *list-rel-split-right-iff dest*: *perfectly-shared-monom-unique-left perfectly-shared-monom-unique-right*)
done

definition *normalize-poly-s* :: ⟨-⟩ **where**
⟨*normalize-poly-s* $\mathcal{V}$ *p* = *do* {
p ← *msortR-vars* $\mathcal{V}$ *p*;

RETURN (*merge-coeffs-s p*)
}⟩

lemma *normalize-poly-s-normalize-poly-s*:
assumes
⟨($\mathcal{V}$, $\mathcal{DV}$) ∈ *perfectly-shared-vars-rel*⟩
⟨(*xs*, *xs'*) ∈ *perfectly-shared-polynom* $\mathcal{V}$⟩ **and**
⟨*vars-llist xs'* ⊆ *set-mset* $\mathcal{DV}$⟩
shows ⟨*normalize-poly-s* $\mathcal{V}$ *xs* ≤ ⇓ (*perfectly-shared-polynom* $\mathcal{V}$) (*normalize-poly xs'*)⟩
unfolding *normalize-poly-s-def normalize-poly-def*
by (*refine-rcg sort-poly-spec-s-sort-poly-spec*[*unfolded msortR-vars-def*[*symmetric*]] *assms perfectly-shared-merge-coeffs-merge-coeffs*)

definition *check-linear-combi-l-s-dom-err* :: ⟨*sllist-polynomial* ⇒ *nat* ⇒ *string nres*⟩ **where**
⟨*check-linear-combi-l-s-dom-err p r* = *SPEC* (λ-. *True*)⟩

definition *mult-poly-full-s* :: ⟨-⟩ **where**
⟨*mult-poly-full-s* $\mathcal{V}$ *p q* = *do* {
pq ← *mult-poly-s* $\mathcal{V}$ *p q*;
normalize-poly-s $\mathcal{V}$ *pq*
}⟩

lemma *mult-poly-full-s-mult-poly-full-prop*:
assumes
⟨($\mathcal{V}$, $\mathcal{DV}$) ∈ *perfectly-shared-vars-rel*⟩
⟨(*xs*, *xs'*) ∈ *perfectly-shared-polynom* $\mathcal{V}$⟩ **and**
⟨(*ys*, *ys'*) ∈ *perfectly-shared-polynom* $\mathcal{V}$⟩ **and**
⟨*vars-llist xs'* ⊆ *set-mset* $\mathcal{DV}$⟩ **and**
⟨*vars-llist ys'* ⊆ *set-mset* $\mathcal{DV}$⟩
shows ⟨*mult-poly-full-s* $\mathcal{V}$ *xs ys* ≤ ⇓ (*perfectly-shared-polynom* $\mathcal{V}$) (*mult-poly-full-prop* $\mathcal{DV}$ *xs' ys'*)⟩
unfolding *mult-poly-full-s-def mult-poly-full-prop-def*
by (*refine-rcg mult-poly-s-mult-poly-raw-prop assms normalize-poly-s-normalize-poly-s*)
(*use assms* **in** *auto*)
definition (**in** −)*linear-combi-l-prep-s*
:: ⟨*nat* ⇒ - ⇒ (*nat*, *string*) *shared-vars* ⇒ - ⇒ (*sllist-polynomial* × (*llist-polynomial* × *nat*) *list* × *string code-status*) *nres*⟩
where
⟨*linear-combi-l-prep-s i A* $\mathcal{V}$ *xs* = *do* {
$WHILE_T$
(λ(*p*, *xs*, *err*). *xs* ≠ [] ∧ ¬*is-cfailed err*)
(λ(*p*, *xs*, -). *do* {
ASSERT(*xs* ≠ []);
let (*q* :: *llist-polynomial*, *i*) = *hd xs*;
if (*i* ∉# *dom-m A* ∨ ¬(*vars-llist-in-s* $\mathcal{V}$ *q*))
then do {
err ← *check-linear-combi-l-s-dom-err p i*;
RETURN (*p*, *xs*, *error-msg i err*)
} *else do* {
ASSERT(*fmlookup A i* ≠ *None*);
let r = *the* (*fmlookup A i*);
if q = [([], *1*)]
then do {
pq ← *add-poly-l-s* $\mathcal{V}$ (*p*, *r*);
RETURN (*pq*, *tl xs*, *CSUCCESS*)}
else do {

(*no-new*, *q*) ← *normalize-poly-sharedS* $\mathcal{V}$ (*q*);
q ← *mult-poly-full-s* $\mathcal{V}$ *q r*;
pq ← *add-poly-l-s* $\mathcal{V}$ (*p*, *q*);
RETURN (*pq*, *tl xs*, *CSUCCESS*)
}
}
})
([], *xs*, *CSUCCESS*)
}›

lemma *normalize-poly-sharedS-normalize-poly-shared*:
assumes
‹($\mathcal{V}$, $\mathcal{DV}$) ∈ *perfectly-shared-vars-rel*›
‹(*xs*, *xs'*) ∈ *Id*›
shows ‹*normalize-poly-sharedS* $\mathcal{V}$ *xs*
≤ ⇓(*bool-rel* $\times_r$ *perfectly-shared-polynom* $\mathcal{V}$)
(*normalize-poly-shared* $\mathcal{DV}$ *xs'*)›
proof −
have [*refine*]: ‹*full-normalize-poly xs* ≤ ⇓ *Id* (*full-normalize-poly xs'*)›
using *assms* **by** *auto*
show *?thesis*
unfolding *normalize-poly-sharedS-def normalize-poly-shared-def*
by (*refine-rcg assms import-poly-no-newS-import-poly-no-new*)
qed

lemma *linear-combi-l-prep-s-linear-combi-l-prep*:
assumes
‹($\mathcal{V}$, $\mathcal{DV}$) ∈ *perfectly-shared-vars-rel*›
‹(*A*,*B*) ∈ ⟨*nat-rel*, *perfectly-shared-polynom* $\mathcal{V}$⟩*fmap-rel*›
‹(*xs*,*xs'*) ∈ *Id*›
shows ‹*linear-combi-l-prep-s i A* $\mathcal{V}$ *xs*
≤ ⇓(*perfectly-shared-polynom* $\mathcal{V}$ $\times_r$ *Id* $\times_r$ *Id*)
(*linear-combi-l-prep2 j B* $\mathcal{DV}$ *xs'*)›
proof −
have [*refine*]: ‹*check-linear-combi-l-s-dom-err a b*
≤ ⇓ *Id*
(*check-linear-combi-l-dom-err c d*)› **for** *a b c d*
unfolding *check-linear-combi-l-dom-err-def check-linear-combi-l-s-dom-err-def*
by *auto*
show *?thesis*
unfolding *linear-combi-l-prep-s-def linear-combi-l-prep2-def*
apply (*refine-rcg normalize-poly-sharedS-normalize-poly-shared*
mult-poly-full-s-mult-poly-full-prop add-poly-l-s-add-poly-l)
subgoal using *assms* **by** *auto*
subgoal by *auto*
subgoal by *auto*
subgoal using *assms* **by** *auto*
subgoal by *auto*
subgoal using *fmap-rel-nat-rel-dom-m*[*OF assms*(*2*)] **unfolding** *in-dom-m-lookup-iff* **by** *auto*
subgoal using *assms* **by** *auto*
subgoal using *assms* **by** *auto*
subgoal using *assms* **by** *auto*
subgoal by *auto*
subgoal using *assms* **by** *auto*

subgoal by *auto*
subgoal using *assms* **by** *auto*
subgoal by *auto*
subgoal using *assms* **by** *auto*
subgoal using *assms* **by** *auto*
subgoal by *auto*
subgoal by *auto*
done
qed

definition *check-linear-combi-l-s-mult-err* :: ‹*sllist-polynomial* ⇒ *sllist-polynomial* ⇒ *string nres*› **where**
‹*check-linear-combi-l-s-mult-err pq r* = *SPEC* (λ-. *True*)›

definition *weak-equality-l-s* :: ‹*sllist-polynomial* ⇒ *sllist-polynomial* ⇒ *bool nres*› **where**
‹*weak-equality-l-s p q* = *RETURN* (*p* = *q*)›

definition *check-linear-combi-l-s* **where**
‹*check-linear-combi-l-s spec A* $\mathcal{V}$ *i xs r* = *do* {
(*mem-err*, *r*) ← *import-poly-no-newS* $\mathcal{V}$ *r*;
if mem-err ∨ *i* ∈# *dom-m A* ∨ *xs* = []
then do {
err ← *check-linear-combi-l-pre-err i* (*i* ∈# *dom-m A*) (*xs* = []) (*mem-err*);
RETURN (*error-msg i err*, *r*)
}
else do {
(*p*, -, *err*) ← *linear-combi-l-prep-s i A* $\mathcal{V}$ *xs*;
if (*is-cfailed err*)
then do {
RETURN (*err*, *r*)
}
else do {
b ← *weak-equality-l-s p r*;
b' ← *weak-equality-l-s r spec*;
if b then (*if b' then RETURN* (*CFOUND*, *r*) *else RETURN* (*CSUCCESS*, *r*)) *else do* {
c ← *check-linear-combi-l-s-mult-err p r*;
RETURN (*error-msg i c*, *r*)
}
}
}}›
definition *weak-equality-l-s'* :: ‹-› **where**
‹*weak-equality-l-s'* - = *weak-equality-l-s*›

definition *weak-equality-l'* :: ‹-› **where**
‹*weak-equality-l'* - = *weak-equality-l*›

lemma *weak-equality-l-s-weak-equality-l*:
fixes *a* :: *sllist-polynomial* **and** *b* :: *llist-polynomial* **and** $\mathcal{V}$ :: ‹(*nat*,*string*)*shared-vars*›
assumes
‹($\mathcal{V}$, $\mathcal{DV}$) ∈ *perfectly-shared-vars-rel*›
‹(*a*,*b*) ∈ *perfectly-shared-polynom* $\mathcal{V}$›
‹(*c*,*d*) ∈ *perfectly-shared-polynom* $\mathcal{V}$›
shows
‹*weak-equality-l-s'* $\mathcal{V}$ *a c* ≤⇓*bool-rel* (*weak-equality-l'* $\mathcal{DV}$ *b d*)›
using *assms perfectly-shared-polynom-unique-left*[*OF assms*(*2*), *of d*]

perfectly-shared-polynom-unique-right[*OF assms*(*1*,*2*), *of c*]
unfolding *weak-equality-l-s-def weak-equality-l-def weak-equality-l′-def weak-equality-l-s′-def*
by *auto*

lemma *check-linear-combi-l-s-check-linear-combi-l*:
assumes
‹($\mathcal{V}$, $\mathcal{DV}$) ∈ *perfectly-shared-vars-rel*›
‹(*A*,*B*) ∈ ⟨*nat-rel*, *perfectly-shared-polynom* $\mathcal{V}$⟩*fmap-rel*› **and**
‹(*xs*,*xs′*)∈ *Id*›
‹(*r*,*r′*)∈*Id*›
‹(*i*,*j*)∈*nat-rel*›
‹(*spec*, *spec′*) ∈ *perfectly-shared-polynom* $\mathcal{V}$›
shows ‹*check-linear-combi-l-s spec A* $\mathcal{V}$ *i r xs*
≤ ⇓(*Id* $\times_r$ *perfectly-shared-polynom* $\mathcal{V}$)
(*check-linear-combi-l-prop spec′ B* $\mathcal{DV}$ *j r′ xs′*)›
proof −
have [*refine*]: ‹*check-linear-combi-l-pre-err a b c d* ≤ ⇓*Id* (*check-linear-combi-l-pre-err u x y z*)›
for *a b c d u x y z*
by (*auto simp*: *check-linear-combi-l-pre-err-def*)
have [*refine*]: ‹*check-linear-combi-l-s-mult-err a b* ≤ ⇓*Id* (*check-linear-combi-l-mult-err u x*)›
for *a b u x*
by (*auto simp*: *check-linear-combi-l-s-mult-err-def check-linear-combi-l-mult-err-def*)

show *?thesis*
unfolding *check-linear-combi-l-s-def check-linear-combi-l-prop-def*
apply (*refine-rcg import-poly-no-newS-import-poly-no-new assms linear-combi-l-prep-s-linear-combi-l-prep weak-equality-l-s-weak-equality-l*[*unfolded weak-equality-l′-def weak-equality-l-s′-def*])
subgoal using *assms* **by** *auto*
subgoal using *assms* **by** *auto*
subgoal using *assms* **by** *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal using *assms* **by** *auto*
done
qed

definition *check-extension-l-s-new-var-multiple-err* :: ‹*string* ⇒ *sllist-polynomial* ⇒ *string nres*› **where**
‹*check-extension-l-s-new-var-multiple-err v p* = *SPEC* (λ-. *True*)›

definition *check-extension-l-s-side-cond-err*
:: ‹*string* ⇒ *sllist-polynomial* ⇒ *sllist-polynomial* ⇒ *sllist-polynomial* ⇒ *string nres*›
where
‹*check-extension-l-s-side-cond-err v p p′ q* = *SPEC* (λ-. *True*)›
term *is-new-variable*
definition (**in** −)*check-extension-l2-s*
:: ‹- ⇒ - ⇒ (*nat*,*string*)*shared-vars* ⇒ *nat* ⇒ *string* ⇒ *llist-polynomial* ⇒
(*string code-status* × *sllist-polynomial* × (*nat*,*string*)*shared-vars* × *nat*) *nres*›

where
‹*check-extension-l2-s spec A* $\mathcal{V}$ *i v p* = *do* {
n ← *is-new-variableS v* $\mathcal{V}$;
let pre = *i* ∉# *dom-m A* ∧ *n*;
let nonew = *vars-llist-in-s* $\mathcal{V}$ *p*;
(*mem*, *p*, $\mathcal{V}$) ← *import-polyS* $\mathcal{V}$ *p*;
let pre = (*pre* ∧ ¬*alloc-failed mem*);
if ¬*pre*
then do {
c ← *check-extension-l-dom-err i*;
RETURN (*error-msg i c*, [], $\mathcal{V}$, *0*)
} *else do* {
if ¬*nonew*
then do {
c ← *check-extension-l-s-new-var-multiple-err v p*;
RETURN (*error-msg i c*, [], $\mathcal{V}$, *0*)
}
else do {
(*mem'*, $\mathcal{V}$, *v'*) ← *import-variableS v* $\mathcal{V}$;
if alloc-failed mem'
then do {
c ← *check-extension-l-dom-err i*;
RETURN (*error-msg i c*, [], $\mathcal{V}$, *0*)
} *else*
do {
p2 ← *mult-poly-full-s* $\mathcal{V}$ *p p*;
let p'' = *map* (λ(*a*,*b*). (*a*, −*b*)) *p*;
q ← *add-poly-l-s* $\mathcal{V}$ (*p2*, *p''*);
eq ← *weak-equality-l-s q* [];
if eq then do {
RETURN (*CSUCCESS*, *p*, $\mathcal{V}$, *v'*)
} *else do* {
c ← *check-extension-l-s-side-cond-err v p p'' q*;
RETURN (*error-msg i c*, [], $\mathcal{V}$, *v'*)
}
}
}
}
}›

lemma *list-rel-tlD*: ‹(*a*, *b*) ∈ ⟨*R*⟩*list-rel* ⟹ (*tl a*, *tl b*) ∈ ⟨*R*⟩*list-rel*›
by (*metis list.sel*(*2*) *list.sel*(*3*) *list-rel-simp*(*1*) *list-rel-simp*(*2*) *list-rel-simp*(*4*) *neq-NilE*)

lemma *check-extension-l2-prop-alt-def*:
‹*check-extension-l2-prop spec A* $\mathcal{V}$ *i v p* = *do* {
n ← *is-new-variable v* $\mathcal{V}$;
let pre = *i* ∉# *dom-m A* ∧ *n*;
let nonew = *vars-llist p* ⊆ *set-mset* $\mathcal{V}$;
(*mem*, *p*, $\mathcal{V}$) ← *import-poly* $\mathcal{V}$ *p*;
(*mem'*, $\mathcal{V}$, *va*) ← *if pre* ∧ *nonew* ∧ ¬ *alloc-failed mem then import-variable v* $\mathcal{V}$ *else RETURN* (*mem*, $\mathcal{V}$, *v*);
let pre = ((*pre* ∧ ¬*alloc-failed mem*) ∧ ¬*alloc-failed mem'*);

if ¬*pre*
then do {
c ← *check-extension-l-dom-err i*;

```
  RETURN (error-msg i c, [], 𝒱, va)
} else do {
   if ¬nonew
   then do {
     c ← check-extension-l-new-var-multiple-err v p;
     RETURN (error-msg i c, [], 𝒱, va)
   }
   else do {
      ASSERT(vars-llist p ⊆ set-mset 𝒱);
      p2 ← mult-poly-full-prop 𝒱 p p;
      ASSERT(vars-llist p2 ⊆ set-mset 𝒱);
      let p'' = map (λ(a,b). (a, −b)) p;
      ASSERT(vars-llist p'' ⊆ set-mset 𝒱);
      q ← add-poly-l-prep 𝒱 (p2, p'');
      ASSERT(vars-llist q ⊆ set-mset 𝒱);
      eq ← weak-equality-l q [];
      if eq then do {
        RETURN (CSUCCESS, p, 𝒱, va)
      } else do {
       c ← check-extension-l-side-cond-err v p q;
       RETURN (error-msg i c, [], 𝒱, va)
      }
    }
   }
  }
}›
```

unfolding *check-extension-l2-prop-def Let-def check-extension-l-side-cond-err-def is-new-variable-def*
by (*auto intro*!: *bind-cong*[*OF refl*])

lemma *check-extension-l2-prop-alt-def2*:

```
‹check-extension-l2-prop spec A 𝒱 i v p = do {
n ← is-new-variable v 𝒱;
let pre = i ∉# dom-m A ∧ n;
let nonew = vars-llist p ⊆ set-mset 𝒱;
(mem, p, 𝒱) ← import-poly 𝒱 p;
let pre = (pre ∧ ¬alloc-failed mem);
if ¬pre
then do {
  c ← check-extension-l-dom-err i;
  RETURN (error-msg i c, [], 𝒱, v)
 } else do {
   if ¬nonew
   then do {
     c ← check-extension-l-new-var-multiple-err v p;
     RETURN (error-msg i c, [], 𝒱, v)
   }
   else do {
   (mem', 𝒱, va) ← import-variable v 𝒱;
   if (alloc-failed mem')
   then do {
    c ← check-extension-l-dom-err i;
    RETURN (error-msg i c, [], 𝒱, va)
   }
   else do {
      ASSERT(vars-llist p ⊆ set-mset 𝒱);
```

p2 ← *mult-poly-full-prop* $\mathcal{V}$ *p p*;
ASSERT(*vars-llist p2* ⊆ *set-mset* $\mathcal{V}$);
let p'' = *map* (λ(*a*,*b*). (*a*, −*b*)) *p*;
ASSERT(*vars-llist* p'' ⊆ *set-mset* $\mathcal{V}$);
q ← *add-poly-l-prep* $\mathcal{V}$ (*p2*, p'');
ASSERT(*vars-llist q* ⊆ *set-mset* $\mathcal{V}$);
eq ← *weak-equality-l q* [];
if eq then do {
RETURN (*CSUCCESS*, *p*, $\mathcal{V}$, *va*)
} *else do* {
c ← *check-extension-l-side-cond-err v p q*;
RETURN (*error-msg i c*, [], $\mathcal{V}$, *va*)
}
}
}
}
}›
unfolding *check-extension-l2-prop-alt-def*
unfolding *Let-def check-extension-l-side-cond-err-def de-Morgan-conj not-not*
by (*subst if-conn*(*2*))
(*auto intro*!: *bind-cong*[*OF refl*])

lemma *list-rel-mapI*: ‹(*xs*,*ys*) ∈ ⟨*R*⟩*list-rel* ⟹ (⋀*x y*. *x* ∈ *set xs* ⟹ *y* ∈ *set ys* ⟹ (*x*,*y*)∈*R* ⟹ (*f x*, *g y*) ∈ *S*) ⟹ (*map f xs*, *map g ys*) ∈ ⟨*S*⟩*list-rel*›
by (*induction xs arbitrary*: *ys*)
(*auto simp*: *list-rel-split-right-iff*)

lemma *perfectly-shared-var-rel-perfectly-shared-monom-mono*:
‹(∀ *xs*. *xs* ∈ *perfectly-shared-var-rel* $\mathcal{A}$ ⟶ *xs* ∈ *perfectly-shared-var-rel* $\mathcal{A}'$) ⟷
(∀ *xs*. *xs* ∈ *perfectly-shared-monom* $\mathcal{A}$ ⟶ *xs* ∈ *perfectly-shared-monom* $\mathcal{A}'$)›
by (*metis list-rel-mono old.prod.exhaust list-rel-simp*(*2*) *list-rel-simp*(*4*))

lemma *perfectly-shared-var-rel-perfectly-shared-polynom-mono*:
‹(∀ *xs*. *xs* ∈ *perfectly-shared-var-rel* $\mathcal{A}$ ⟶ *xs* ∈ *perfectly-shared-var-rel* $\mathcal{A}'$) ⟷
(∀ *xs*. *xs* ∈ *perfectly-shared-polynom* $\mathcal{A}$ ⟶ *xs* ∈ *perfectly-shared-polynom* $\mathcal{A}'$)›
unfolding *perfectly-shared-var-rel-perfectly-shared-monom-mono*
apply (*auto intro*: *list-rel-mono*)
apply (*drule-tac x* = ‹[(*a*,*1*)]› **in** *spec*)
apply (*drule-tac x* = ‹[(*b*,*1*)]› **in** *spec*)
apply *auto*
done

lemma *check-extension-l2-s-check-extension-l2*:
assumes
‹($\mathcal{V}$, $\mathcal{DV}$) ∈ *perfectly-shared-vars-rel*›
‹(*A*,*B*) ∈ ⟨*nat-rel*, *perfectly-shared-polynom* $\mathcal{V}$⟩*fmap-rel*› **and**
‹(*r*,*r'*)∈*Id*›
‹(*i*,*j*)∈*nat-rel*›
‹(*spec*, *spec'*) ∈ *perfectly-shared-polynom* $\mathcal{V}$›
‹(*v*,*v'*)∈*Id*›
shows ‹*check-extension-l2-s spec A* $\mathcal{V}$ *i v r*
≤ ⇓{((*err*, *p*, *A*, *v*), (*err'*, *p'*, *A'*, *v'*)).
(*err*, *err'*) ∈ *Id* ∧

(¬*is-cfailed err* ⟶
(*p*, *p′*) ∈ *perfectly-shared-polynom A* ∧ (*v*, *v′*) ∈ *perfectly-shared-var-rel A* ∧
(*A*, *A′*) ∈ {(*a*,*b*). (*a*,*b*) ∈ *perfectly-shared-vars-rel* ∧ *perfectly-shared-polynom* $\mathcal{V}$ ⊆ *perfectly-shared-polynom a*})}
(*check-extension-l2-prop spec′ B* $\mathcal{DV}$ *j v′ r′*)⟩
proof −
have [*refine*]: ⟨*check-extension-l-s-new-var-multiple-err a b* ≤⇓*Id* (*check-extension-l-new-var-multiple-err a′ b′*)⟩ **for** *a a′ b b′*
by (*auto simp*: *check-extension-l-s-new-var-multiple-err-def check-extension-l-new-var-multiple-err-def*)

have [*refine*]: ⟨*check-extension-l-dom-err i* ≤ ⇓ (*Id*) (*check-extension-l-dom-err j*)⟩
by (*auto simp*: *check-extension-l-dom-err-def*)
have [*refine*]: ⟨*check-extension-l-s-side-cond-err a b c d* ≤ ⇓*Id* (*check-extension-l-side-cond-err a′ b′ c′*)⟩ **for** *a b c d a′ b′ c′ d′*
by (*auto simp*: *check-extension-l-s-side-cond-err-def check-extension-l-side-cond-err-def*)
have *G*: ⟨(*a*, *b*) ∈ *import-poly-rel* $\mathcal{V}$ *x* ⟹ ¬*alloc-failed* (*fst a*) ⟹
(*snd* (*snd a*), *snd* (*snd b*)) ∈ *perfectly-shared-vars-rel*⟩ **for** *a b x*
by *auto*

show *?thesis*
unfolding *check-extension-l2-s-def check-extension-l2-prop-alt-def2 nres-monad3*
apply (*refine-rcg import-polyS-import-poly assms mult-poly-full-s-mult-poly-full-prop import-variableS-import-variable*[*unfolded perfectly-shared-var-rel-perfectly-shared-polynom-mono*] *is-new-variable-spec*)
subgoal using *assms* **by** *auto*
subgoal using *assms* **by** (*auto simp add*: *perfectly-shared-vars-rel-def perfectly-shared-vars-def*)
subgoal using *assms* **by** (*auto*)
subgoal using *assms* **by** (*auto*)
subgoal using *assms* **by** (*auto*)
subgoal using *assms* **by** *auto*
subgoal by *auto*
subgoal using *assms* **by** *auto*
subgoal using *assms* **by** *auto*
subgoal using *assms* **by** *auto*
subgoal using *assms* **by** *auto*
apply (*rule add-poly-l-s-add-poly-l*)
subgoal by *auto*
subgoal for - - *x x′ x1 x2 x1a x2a x1b x2b x1c x2c xa x′a x1d x2d x1e x2e x1f x2f x1g x2g p2 p2a*
using *assms*
by (*auto intro*!: *list-rel-mapI*[*of* - - ⟨*perfectly-shared-monom x1g* $\times_r$ *int-rel*⟩])
apply (*rule weak-equality-l-s-weak-equality-l*[*unfolded weak-equality-l′-def weak-equality-l-s′-def*])
defer apply *assumption*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal using *assms* **by** *auto*
apply (*solves auto*)
done
qed

definition *PAC-checker-l-step-s*
:: ⟨*sllist-polynomial* ⇒ *string code-status* × (*nat*,*string*)*shared-vars* × - ⇒ (*llist-polynomial*, *string*, *nat*) *pac-step* ⇒ -⟩
where

⟨*PAC-checker-l-step-s* = (*λspec* (*st′*, 𝒱, *A*) *st*. *do* {
ASSERT (¬*is-cfailed st′*);
case st of
CL - - - ⇒
do {
r ← *full-normalize-poly* (*pac-res st*);
(*eq*, *r*) ← *check-linear-combi-l-s spec A* 𝒱 (*new-id st*) (*pac-srcs 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*)
}
| *Extension* - - - ⇒
do {
r ← *full-normalize-poly* (*pac-res st*);
(*eq*, *r*, 𝒱, *v*) ← *check-extension-l2-s spec A* (𝒱) (*new-id st*) (*new-var st*) *r*;
if ¬*is-cfailed eq*
then do {
r ← *add-poly-l-s* 𝒱 ([([*v*], −*1*)], *r*);
RETURN (*st′*, 𝒱, *fmupd* (*new-id st*) *r A*)
}
else RETURN (*eq*, 𝒱, *A*)
}}
)⟩

lemma *is-cfailed-merge-cstatus*:
is-cfailed (*merge-cstatus c d*) ⟷ *is-cfailed c* ∨ *is-cfailed d*
by (*cases c*; *cases d*) *auto*

lemma (**in** −) *fmap-rel-mono2*:
⟨*x* ∈ ⟨*A*,*B*⟩*fmap-rel* ⟹ *B* ⊆*B′* ⟹ *x* ∈ ⟨*A*,*B′*⟩*fmap-rel*⟩
by (*auto simp*: *fmap-rel-alt-def*)

lemma *PAC-checker-l-step-s-PAC-checker-l-step-s*:
assumes
⟨(𝒱, 𝒟𝒱) ∈ *perfectly-shared-vars-rel*⟩
⟨(*A*,*B*) ∈ ⟨*nat-rel*, *perfectly-shared-polynom* 𝒱⟩*fmap-rel*⟩ **and**
⟨(*spec*, *spec′*) ∈ *perfectly-shared-polynom* 𝒱⟩ **and**
⟨(*err*, *err′*) ∈ *Id*⟩ **and**
⟨(*st*,*st′*)∈*Id*⟩
shows ⟨*PAC-checker-l-step-s spec* (*err*, 𝒱, *A*) *st*
≤ ⇓{((*err*, 𝒱′, *A′*), (*err′*, 𝒟𝒱′, *B′*)).
(*err*, *err′*) ∈ *Id* ∧
(¬*is-cfailed err* ⟶ ((𝒱′, 𝒟𝒱′) ∈ *perfectly-shared-vars-rel* ∧(*A′*,*B′*) ∈ ⟨*nat-rel*, *perfectly-shared-polynom* 𝒱′⟩*fmap-rel* ∧
perfectly-shared-polynom 𝒱 ⊆ *perfectly-shared-polynom* 𝒱′))}
(*PAC-checker-l-step-prep spec′* (*err′*, 𝒟𝒱, *B*) *st′*)⟩
proof −
have [*refine*]: ⟨*check-del-l spec A* (*EPAC-Checker-Specification.pac-step.pac-src1 st*)

$\leq \Downarrow$ *Id*
(*check-del-l spec′ B*
(*EPAC-Checker-Specification.pac-step.pac-src1 st′*))›
by (*auto simp*: *check-del-l-def*)
have *HID*: ‹$f = f' \Longrightarrow f \leq \Downarrow Id\ f'$› **for** *f f′*
by *auto*
show *?thesis*
unfolding *PAC-checker-l-step-s-def PAC-checker-l-step-prep-def pac-step.case-eq-if prod.simps*
apply (*refine-rcg check-linear-combi-l-s-check-linear-combi-l check-extension-l2-s-check-extension-l2 add-poly-l-s-add-poly-l*)
subgoal using *assms* **by** *auto*
subgoal using *assms* **by** *auto*
apply (*rule HID*)
subgoal using *assms* **by** *auto*
subgoal using *assms* **by** *auto*
subgoal using *assms* **by** *auto*
subgoal by *auto*
subgoal using *assms* **by** *auto*
subgoal using *assms* **by** *auto*
subgoal using *assms* **by** *auto*
subgoal by *auto*
subgoal using *assms* **by** (*auto simp*: *is-cfailed-merge-cstatus intro*!: *fmap-rel-fmupd-fmap-rel*)
subgoal by *auto*
subgoal using *assms* **by** *auto*
apply (*rule HID*)
subgoal using *assms* **by** *auto*
subgoal using *assms* **by** *auto*
subgoal using *assms* **by** *auto*
subgoal using *assms* **by** *auto*
subgoal using *assms* **by** *auto*
subgoal using *assms* **by** *auto*
subgoal using *assms* **by** *auto*
subgoal using *assms* **by** *auto*
subgoal by *auto*
subgoal by *auto*
subgoal using *assms* **by** (*auto intro*!: *fmap-rel-fmupd-fmap-rel intro*: *fmap-rel-mono2*)
subgoal by *auto*
subgoal by *auto*
subgoal using *assms* **by** (*auto intro*!: *fmap-rel-fmdrop-fmap-rel*)
subgoal by *auto*
done
qed

lemma *PAC-checker-l-step-s-PAC-checker-l-step-s2*:
assumes
‹(*st*,*st′*)∈*Id*›
‹(*spec*, *spec′*) ∈ *perfectly-shared-polynom* (*fst* (*snd errVA*))› **and**
‹((*errVA*), (*err′DVB*)) ∈ *Id* $\times_r$ *perfectly-shared-vars-rel* $\times_r$ ⟨*nat-rel*, *perfectly-shared-polynom* (*fst* (*snd errVA*))⟩*fmap-rel*›
shows ‹*PAC-checker-l-step-s spec* (*errVA*) *st*
$\leq \Downarrow$ {((*err*, $\mathcal{V}'$, *A′*), (*err′*, $\mathcal{DV}'$, *B′*)).
(*err*, *err′*) ∈ *Id* ∧
(¬*is-cfailed err* ⟶ (($\mathcal{V}'$, $\mathcal{DV}'$) ∈ *perfectly-shared-vars-rel* ∧(*A′*,*B′*) ∈ ⟨*nat-rel*, *perfectly-shared-polynom* $\mathcal{V}'$⟩*fmap-rel* ∧

perfectly-shared-polynom (*fst* (*snd* *err𝒱A*)) ⊆ *perfectly-shared-polynom* 𝒱′))}
(*PAC-checker-l-step-prep spec′* (*err′𝒟𝒱B*) *st′*)›
using *PAC-checker-l-step-s-PAC-checker-l-step-s*[*of* ‹*fst* (*snd* *err𝒱A*)› ‹*fst* (*snd* *err′𝒟𝒱B*)›
‹*snd* (*snd* *err𝒱A*)› ‹*snd* (*snd* *err′𝒟𝒱B*)› *spec spec′* ‹*fst* (*err𝒱A*)› ‹*fst* (*err′𝒟𝒱B*)› *st st′*] *assms*
by (*cases err𝒱A*; *cases err′𝒟𝒱B*)
auto

definition *fully-normalize-and-import* **where**
‹*fully-normalize-and-import* 𝒱 *p* = *do* {
p ← *sort-all-coeffs p*;
(*err*, *p*, 𝒱) ← *import-polyS* 𝒱 *p*;
if alloc-failed err
then RETURN (*err*, *p*, 𝒱)
else do {
p ← *normalize-poly-s* 𝒱 *p*;
RETURN (*err*, *p*, 𝒱)
}}›

fun *vars-llist-l* **where**
‹*vars-llist-l* [] = []› |
‹*vars-llist-l* (*x*#*xs*) = *fst x* @ *vars-llist-l xs*›

lemma *set-vars-llist-l*[*simp*]: ‹*set*(*vars-llist-l xs*) = *vars-llist xs*›
by (*induction xs*)
(*auto*)

lemma *vars-llist-l-append*[*simp*]: ‹*vars-llist-l* (*a* @ *b*) = *vars-llist-l a* @ *vars-llist-l b*›
by (*induction a*) *auto*

definition (**in** −) *remap-polys-s-with-err* :: ‹*llist-polynomial* ⇒ *llist-polynomial* ⇒ (*nat*, *string*) *shared-vars* ⇒ (*nat*, *llist-polynomial*) *fmap* ⇒
(*string code-status* × (*nat*, *string*) *shared-vars* × (*nat*, *sllist-polynomial*) *fmap* × *sllist-polynomial*) *nres*› **where**
‹*remap-polys-s-with-err spec spec0* = (λ(𝒱:: (*nat*, *string*) *shared-vars*) *A*. *do*{
ASSERT(*vars-llist spec* ⊆ *vars-llist spec0*);
dom ← *SPEC*(λ*dom*. *set-mset* (*dom-m A*) ⊆ *dom* ∧ *finite dom*);
(*mem*, 𝒱) ← *import-variablesS* (*vars-llist-l spec0*) 𝒱;
(*mem′*, *spec*, 𝒱) ← *if* ¬*alloc-failed mem then import-polyS* 𝒱 *spec else RETURN* (*mem*, [], 𝒱);
failed ← *SPEC*(λ*b*::*bool*. *alloc-failed mem* ∨ *alloc-failed mem′* ⟶ *b*);
if failed
then do {
c ← *remap-polys-l-dom-err*;
RETURN (*error-msg* (*0* :: *nat*) *c*, 𝒱, *fmempty*, [])
}
else do {
(*err*, 𝒱, *A*) ← $FOREACH_C$ *dom* (λ(*err*, 𝒱, *A′*). ¬*is-cfailed err*)
(λ*i* (*err*, 𝒱, *A′*).
if i ∈# *dom-m A*
then do {
(*err′*, *p*, 𝒱) ← *import-polyS* 𝒱 (*the* (*fmlookup A i*));
if alloc-failed err′ then RETURN((*CFAILED ″memory out″*, 𝒱, *A′*))
else do {
p ← *full-normalize-poly-s* 𝒱 *p*;
eq ← *weak-equality-l-s′* 𝒱 *p spec*;
let 𝒱 = 𝒱;

```
        RETURN((if eq then CFOUND else CSUCCESS), 𝒱, fmupd i p A′)
      }
    } else RETURN (err, 𝒱, A′))
  (CSUCCESS, 𝒱, fmempty);
  RETURN (err, 𝒱, A, spec)
        }})›

lemma full-normalize-poly-alt-def:
 ‹full-normalize-poly p0 = do {
   p ← sort-all-coeffs p0;
   ASSERT(vars-llist p ⊆ vars-llist p0);
   p ← sort-poly-spec p;
   ASSERT(vars-llist p ⊆ vars-llist p0);
   RETURN (merge-coeffs0 p)
 }› (is ‹?A = ?B›)
proof −
 have sort-poly-spec1: ‹(p,p′)∈ Id ⟹ sort-poly-spec p ≤ ⇓ Id (sort-poly-spec p′)› for p p′
   by auto

  have sort-all-coeffs2: ‹sort-all-coeffs xs ≤⇓{(ys,ys′). (ys,ys′) ∈ Id ∧ vars-llist ys ⊆ vars-llist xs}
(sort-all-coeffs xs)› for xs
 proof −
   term xs
   have [refine]: ‹(xs, xs) ∈ ⟨{(ys,ys′). (ys,ys′) ∈ Id ∧ ys ∈ set xs}⟩list-rel›
    by (rule list-rel-mono-strong[of - Id])
      (auto)
    have [refine]: ‹(x1a,x1)∈ Id ⟹ sort-coeff x1a  ≤ ⇓ {(ys,ys′). (ys,ys′) ∈ Id ∧ set ys ⊆ set x1a}
(sort-coeff x1)› for x1a x1
     unfolding sort-coeff-def
     by (auto intro!: RES-refine dest: mset-eq-setD)

   show ?thesis
     unfolding sort-all-coeffs-def
     apply refine-vcg
     subgoal by auto
     subgoal by auto
     subgoal by (auto dest!: split-list)
     done
 qed

 have sort-poly-spec1: ‹(p,p′)∈ Id ⟹ sort-poly-spec p ≤ ⇓ Id (sort-poly-spec p′)› for p p′
   by auto
  have sort-poly-spec2: ‹(p,p′)∈Id ⟹ sort-poly-spec p ≤ ⇓ {(ys,ys′). (ys,ys′) ∈ Id ∧ vars-llist ys ⊆
vars-llist p} (sort-poly-spec p′)›
   for p p′
   by (auto simp: sort-poly-spec-def intro!: RES-refine dest: vars-llist-mset-eq)
 have ‹?A ≤ ⇓Id ?B›
   unfolding full-normalize-poly-def
   by (refine-rcg sort-poly-spec1) auto
 moreover have ‹?B ≤ ⇓Id ?A›
   unfolding full-normalize-poly-def
   apply (rule bind-refine[OF sort-all-coeffs2])
   apply (refine-vcg sort-poly-spec2)
   subgoal by auto
   subgoal by auto
```

    subgoal by *auto*
    subgoal by *auto*
    done
  ultimately show *?thesis*
    by *auto*
qed

definition *full-normalize-poly′* :: ‹-› **where**
  ‹*full-normalize-poly′* - = *full-normalize-poly*›

lemma *full-normalize-poly-s-full-normalize-poly*:
  fixes *xs* :: ‹*sllist-polynomial*› **and**
    $\mathcal{V}$ :: ‹(*nat*,*string*)*shared-vars*›
  assumes
    ‹(*xs*, *xs′*) ∈ *perfectly-shared-polynom* $\mathcal{V}$› **and**
    $\mathcal{V}$: ‹($\mathcal{V}$, $\mathcal{DV}$) ∈ *perfectly-shared-vars-rel*› **and**
    ‹*vars-llist xs′* ⊆ *set-mset* $\mathcal{DV}$›
  shows ‹*full-normalize-poly-s* $\mathcal{V}$ *xs* ≤ ⇓(*perfectly-shared-polynom* $\mathcal{V}$) (*full-normalize-poly′* $\mathcal{DV}$ *xs′*)›
proof −
  show *?thesis*
    unfolding *full-normalize-poly-s-def full-normalize-poly-alt-def full-normalize-poly′-def*
    apply (*refine-rcg sort-all-coeffs-s-sort-all-coeffs assms*
      sort-poly-spec-s-sort-poly-spec merge-coeffs0-s-merge-coeffs0)
    subgoal using *assms* **by** *auto*
    done
qed

lemma *remap-polys-l2-with-err-prep-alt-def*:
  ‹*remap-polys-l2-with-err-prep spec spec0* = (λ($\mathcal{V}$:: (*nat*, *string*) *vars*) *A*. *do*{
   ASSERT(*vars-llist spec* ⊆ *vars-llist spec0*);
   dom ← *SPEC*(λ*dom*. *set-mset* (*dom-m A*) ⊆ *dom* ∧ *finite dom*);
   (*mem*, $\mathcal{V}$) ← *SPEC*(λ(*mem*, $\mathcal{V}'$). ¬*alloc-failed mem* ⟶ *set-mset* $\mathcal{V}'$ = *set-mset* $\mathcal{V}$ ∪ *vars-llist spec0*);
   (*mem′*, *spec*, $\mathcal{V}$) ← *if* ¬*alloc-failed mem then import-poly* $\mathcal{V}$ *spec else SPEC*(λ-. *True*);
   failed ← *SPEC*(λ*b*::*bool*. *alloc-failed mem* ∨ *alloc-failed mem′* ⟶ *b*);
   if failed
   then do {
      c ← *remap-polys-l-dom-err*;
      SPEC (λ(*mem*, -, -, -). *mem* = *error-msg* (*0*::*nat*) *c*)
   }
   else do {
     (*err*, $\mathcal{V}$, *A*) ← $FOREACH_C$ *dom* (λ(*err*, $\mathcal{V}$, *A′*). ¬*is-cfailed err*)
       (λ*i* (*err*, $\mathcal{V}$, *A′*).
          if i ∈# *dom-m A*
          then do {
           (*err′*, *p*, $\mathcal{V}$) ← *import-poly* $\mathcal{V}$ (*the* (*fmlookup A i*));
            if alloc-failed err′ then RETURN((*CFAILED* ″*memory out*″, $\mathcal{V}$, *A′*))
            else do {
              ASSERT(*vars-llist p* ⊆ *set-mset* $\mathcal{V}$);
              p ← *full-normalize-poly′* $\mathcal{V}$ *p*;
              eq ← *weak-equality-l′* $\mathcal{V}$ *p spec*;
              let $\mathcal{V}$ = $\mathcal{V}$;
              RETURN((*if eq then CFOUND else CSUCCESS*), $\mathcal{V}$, *fmupd i p A′*)
            }
          } *else RETURN* (*err*, $\mathcal{V}$, *A′*))
       (*CSUCCESS*, $\mathcal{V}$, *fmempty*);

RETURN (*err*, $\mathcal{V}$, *A*, *spec*)
}})›
unfolding *full-normalize-poly′-def weak-equality-l′-def*
by(*auto simp*: *remap-polys-l2-with-err-prep-def*
intro!: *ext bind-cong*[*OF refl*])

lemma *remap-polys-s-with-err-remap-polys-l2-with-err-prep*:
fixes $\mathcal{V}$ :: ‹(*nat*, *string*) *shared-vars*›
assumes
$\mathcal{V}$: ‹($\mathcal{V}$, $\mathcal{DV}$) ∈ *perfectly-shared-vars-rel*› **and**
AB: ‹(*A*,*B*) ∈ ⟨*nat-rel*, *Id*⟩*fmap-rel*› **and**
‹(*spec*, *spec′*) ∈ ⟨⟨*Id*⟩*list-rel* $\times_r$ *int-rel*⟩*list-rel*› **and**
spec0: ‹(*spec0*, *spec0′*) ∈ ⟨⟨*Id*⟩*list-rel* $\times_r$ *int-rel*⟩*list-rel*›
shows
‹*remap-polys-s-with-err spec spec0* $\mathcal{V}$ *A* ≤
⇓{((*err*, $\mathcal{V}$, *A*, *fspec*), (*err′*, $\mathcal{V}'$, *A′*, *fspec′*)).
(*err*, *err′*) ∈ *Id* ∧
(¬*is-cfailed err* ⟶ (*fspec*, *fspec′*) ∈ *perfectly-shared-polynom* $\mathcal{V}$ ∧
((*err*, $\mathcal{V}$, *A*), (*err′*, $\mathcal{V}'$, *A′*)) ∈ *Id* $\times_r$ *perfectly-shared-vars-rel* $\times_r$⟨*nat-rel*, *perfectly-shared-polynom* $\mathcal{V}$⟩*fmap-rel*)}
(*remap-polys-l2-with-err-prep spec′ spec0′* $\mathcal{DV}$ *B*)›
proof −
have *vars-spec*: ‹(*vars-llist-l spec0*, *vars-llist-l spec0*) ∈ *Id*›
by *auto*
have [*refine*]: ‹*import-variablesS* (*vars-llist-l spec0*) $\mathcal{V}$
≤ ⇓ {((*mem*, $\mathcal{VV}$), (*mem′*, $\mathcal{VV}'$)). *mem*=*mem′* ∧ (¬*alloc-failed mem* ⟶ ($\mathcal{VV}$, $\mathcal{VV}'$) ∈ *perfectly-shared-vars-rel* ∧
perfectly-shared-polynom $\mathcal{V}$ ⊆ *perfectly-shared-polynom* $\mathcal{VV}$)}
(*SPEC* (λ(*mem*, $\mathcal{V}'$). ¬*alloc-failed mem* ⟶ *set-mset* $\mathcal{V}'$ = *set-mset* $\mathcal{DV}$ ∪ *vars-llist spec0′*))›
apply (*rule import-variablesS-import-variables*[*OF* $\mathcal{V}$ *vars-spec*, *THEN order-trans*])
apply (*rule ref-two-step′*[*THEN order-trans*])
apply (*rule import-variables-spec*)
apply (*use spec0* **in** ‹*auto simp*: *conc-fun-RES*
dest!: *spec*[*of* - ‹$\mathcal{DV}$ + *mset* (*vars-llist-l spec0*)›]›)
by (*meson perfectly-shared-var-rel-perfectly-shared-polynom-mono subset-eq*)

have *1*: ‹*inj-on id* (*dom* :: *nat set*)› **for** *dom*
by (*auto simp*: *inj-on-def*)
have [*refine*]: ‹(*x2e*, *x2c*) ∈ *perfectly-shared-vars-rel* ⟹
((*CSUCCESS*, *x2e*, *fmempty*), *CSUCCESS*, *x2c*, *fmempty*)
∈ {((*mem*,$\mathcal{A}$, *A*), (*mem′*,$\mathcal{A}'$, *A′*)). (*mem*,*mem′*) ∈ *Id* ∧
(¬*is-cfailed mem* ⟶ (($\mathcal{A}$, *A*), ($\mathcal{A}'$, *A′*))∈ *perfectly-shared-vars-rel* $\times_r$⟨*nat-rel*, *perfectly-shared-polynom* $\mathcal{A}$⟩*fmap-rel* ∧
perfectly-shared-polynom x2e ⊆ *perfectly-shared-polynom* $\mathcal{A}$)}›
for *x2e x2c*
by *auto*
have [*simp*]: ‹*A* ∝ *xb* = *B* ∝ *xb*› **for** *xb*
using *AB* **unfolding** *fmap-rel-alt-def* **apply** *auto* **by** (*metis in-dom-m-lookup-iff*)
show *?thesis*
unfolding *remap-polys-s-with-err-def remap-polys-l2-with-err-prep-alt-def Let-def*
apply (*refine-rcg import-polyS-import-poly 1 full-normalize-poly-s-full-normalize-poly*
weak-equality-l-s-weak-equality-l)
subgoal using *assms* **by** *auto*
subgoal using *assms* **by** *auto*
subgoal by *auto*

subgoal by *auto*
subgoal using *assms* **by** *auto*
subgoal by (*auto intro*!: *RETURN-RES-refine*)
subgoal by *auto*
subgoal by *auto*
subgoal by (*clarsimp intro*!: *RETURN-RES-refine*)
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal using *assms* **by** *auto*
subgoal using *assms* **by** *auto*
subgoal by *auto*
subgoal using *assms* **by** *auto*
subgoal using *assms* **by** *auto*
subgoal using *assms* **by** *auto*
subgoal by *auto*
subgoal by *simp*
subgoal by *auto*
subgoal
by (*auto intro*!: *fmap-rel-fmupd-fmap-rel*
intro: *fmap-rel-mono2*)
subgoal by *auto*
subgoal
by (*auto intro*!: *fmap-rel-fmupd-fmap-rel*
intro: *fmap-rel-mono2*)
done
qed

definition *PAC-checker-l-s* **where**
‹*PAC-checker-l-s 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-s spec bA* (*hd n*);
RETURN (*S*, *tl n*)
})
((*b*, *A*), *st*);
RETURN S
}›

lemma *PAC-checker-l-s-PAC-checker-l-prep-s*:
assumes
‹(𝒱, 𝒟𝒱) ∈ *perfectly-shared-vars-rel*›
‹(*A*,*B*) ∈ ⟨*nat-rel*, *perfectly-shared-polynom* 𝒱⟩*fmap-rel*› **and**
‹(*spec*, *spec*′) ∈ *perfectly-shared-polynom* 𝒱› **and**
‹(*err*, *err*′) ∈ *Id*› **and**
‹(*st*,*st*′)∈*Id*›
shows ‹*PAC-checker-l-s spec* (𝒱, *A*) *err st*
≤ ⇓{((*err*, 𝒱′, *A*′), (*err*′, 𝒟𝒱′, *B*′)).
(*err*, *err*′) ∈ *Id* ∧
(¬*is-cfailed err* ⟶ ((𝒱′, 𝒟𝒱′) ∈ *perfectly-shared-vars-rel* ∧(*A*′,*B*′) ∈ ⟨*nat-rel*, *perfectly-shared-polynom* 𝒱′⟩*fmap-rel*))}
(*PAC-checker-l2 spec*′ (𝒟𝒱, *B*) *err*′ *st*′)›

proof −
  show *?thesis*
    unfolding *PAC-checker-l-s-def PAC-checker-l2-def*
    apply (*refine-rcg PAC-checker-l-step-s-PAC-checker-l-step-s2*
      WHILET-refine[**where** $R = \langle\{((err, \mathcal{V}', A'), err', \mathcal{DV}', B').$
  $(err, err') \in Id \wedge (\neg$ *is-cfailed err* $\longrightarrow$
  $(\mathcal{V}', \mathcal{DV}') \in$ *perfectly-shared-vars-rel* $\wedge$
      $(A', B') \in \langle$*nat-rel*, *perfectly-shared-polynom* $\mathcal{V}'\rangle$*fmap-rel* $\wedge$
    perfectly-shared-polynom $\mathcal{V} \subseteq$ *perfectly-shared-polynom* $\mathcal{V}')\} \times_r Id\rangle$])
    subgoal using *assms* **by** *auto*
    subgoal by *auto*
    subgoal by *auto*
    subgoal by *auto*
    subgoal using *assms* **by** *auto*
    subgoal by *auto*
    subgoal by *force*
    subgoal by *auto*
    done
qed

definition *full-checker-l-s*
  :: ‹*llist-polynomial* ⇒ (*nat*, *llist-polynomial*) *fmap* ⇒ (-, *string*, *nat*) *pac-step list* ⇒
    (*string code-status* × -) *nres*›
where
  ‹*full-checker-l-s spec A st* = *do* {
    spec' ← *full-normalize-poly spec*;
    (*b*, $\mathcal{V}$, *A*, *spec'*) ← *remap-polys-s-with-err spec' spec* ({#}, *fmempty*, *fmempty*) *A*;
    if is-cfailed b
    then RETURN (*b*, $\mathcal{V}$, *A*)
    else do {
      PAC-checker-l-s spec' ($\mathcal{V}$, *A*) *b st*
     }
  }›
lemma *full-checker-l-s-full-checker-l-prep*:
  assumes
    ‹(*A*,*B*) ∈ ⟨*nat-rel*, *Id*⟩*fmap-rel*› **and**
    ‹(*spec*, *spec'*) ∈ ⟨⟨*Id*⟩*list-rel* $\times_r$ *int-rel*⟩*list-rel*› **and**
    ‹(*st*,*st'*)∈*Id*›
  shows ‹*full-checker-l-s spec A st*
    ≤ ⇓{((*err*, -), (*err'*, -)). (*err*, *err'*) ∈ *Id*}
    (*full-checker-l-prep spec' B st'*)›
proof −
  have [*refine*]: ‹*full-normalize-poly spec* ≤ ⇓ (⟨⟨*Id*⟩*list-rel* $\times_r$ *int-rel*⟩*list-rel*) (*full-normalize-poly spec'*)›
    using *assms* **by** *auto*
  have [*refine*]: ‹(({#}, *fmempty*, *fmempty*), {#}) ∈ *perfectly-shared-vars-rel*›
     by (*auto simp*: *perfectly-shared-vars-rel-def perfectly-shared-vars-def*)
   have *H*: ‹(*x1d*, *x1a*) ∈ *perfectly-shared-vars-rel*›
     ‹(*x1e*, *x1b*) ∈ ⟨*nat-rel*, *perfectly-shared-polynom x1d*⟩*fmap-rel*›
     ‹(*x2e*, *x2b*) ∈ *perfectly-shared-polynom x1d*›
     ‹(*x1c*, *x1*) ∈ *Id*›
     if ‹(*x*, *x'*)
       ∈ {((*err*, $\mathcal{V}$, *A*, *fspec*), *err'*, $\mathcal{V}'$, *A'*, *fspec'*).
       (*err*, *err'*) ∈ *Id* ∧
       (¬ *is-cfailed err* ⟶
       (*fspec*, *fspec'*) ∈ *perfectly-shared-polynom* $\mathcal{V}$ ∧

((*err*, $\mathcal{V}$, *A*), *err'*, $\mathcal{V}'$, *A'*)
∈ *Id* $\times_r$ *perfectly-shared-vars-rel* $\times_r$ ⟨*nat-rel*, *perfectly-shared-polynom* $\mathcal{V}$⟩*fmap-rel*)}›
‹*x2a* = (*x1b*, *x2b*)›
‹*x2* = (*x1a*, *x2a*)›
‹*x'* = (*x1*, *x2*)›
‹*x2d* = (*x1e*, *x2e*)›
‹*x2c* = (*x1d*, *x2d*)›
‹*x* = (*x1c*, *x2c*)›
‹¬ *is-cfailed x1c*›
for *spec' spec'a x x' x1 x2 x1a x2a x1b x2b x1c x2c x1d x2d x1e x2e*
using *that* **by** *auto*
term *PAC-checker-l2*
thm *PAC-checker-l-s-PAC-checker-l-prep-s*
show *?thesis*
unfolding *full-checker-l-s-def full-checker-l-prep-def*
apply (*refine-rcg PAC-checker-l-s-PAC-checker-l-prep-s*[*THEN order-trans*]
remap-polys-s-with-err-remap-polys-l2-with-err-prep assms)
subgoal by (*auto simp*: *perfectly-shared-vars-rel-def perfectly-shared-vars-def*)
subgoal using *assms* **by** *auto*
apply (*rule H*(*1*); *assumption*)
apply (*rule H*(*2*); *assumption*)
apply (*rule H*(*3*); *assumption*)
apply (*rule H*(*4*); *assumption*)
subgoal by (*auto intro*!: *conc-fun-R-mono*)
done
qed

lemma *full-checker-l-s-full-checker-l-prep'*:
‹(*uncurry2 full-checker-l-s*, *uncurry2 full-checker-l-prep*)∈
(⟨⟨*Id*⟩*list-rel* $\times_r$ *int-rel*⟩*list-rel* $\times_r$ ⟨*nat-rel*, *Id*⟩*fmap-rel*) $\times_r$ *Id* $\rightarrow_f$
⟨{((*err*, -), (*err'*, -)). (*err*, *err'*) ∈ *Id*}⟩*nres-rel*›
by (*auto intro*!: *frefI nres-relI full-checker-l-s-full-checker-l-prep*[*THEN order-trans*])

definition *merge-coeff-s* :: ‹(*nat*,*string*)*shared-vars* ⇒ *nat list* ⇒ *nat list* ⇒ *nat list* ⇒ *nat list nres*›
where
‹*merge-coeff-s* $\mathcal{V}$ *xs* = *mergeR* (λ*a b*. *a* ∈ *set xs* ∧ *b* ∈ *set xs*)
(λ*a b*. *do* {
x ← *get-var-nameS* $\mathcal{V}$ *a*;
y ← *get-var-nameS* $\mathcal{V}$ *b*;
RETURN(*a* = *b* ∨ *var-order x y*)
})›

term *get-var-nameS*
sepref-definition *merge-coeff-s-impl*
is ‹*uncurry3 merge-coeff-s*›
:: ‹*shared-vars-assn*^{k} $*_a$ (*monom-s-assn*)^k $*_a$ (*monom-s-assn*)^k $*_a$ (*monom-s-assn*)^k $\rightarrow_a$ *monom-s-assn*›
supply [[*goals-limit=1*]]
unfolding *merge-coeff-s-def mergeR-alt-def var-order'-def*[*symmetric*]
by *sepref*

sepref-register *merge-coeff-s msort-coeff-s sort-all-coeffs-s*
lemmas [*sepref-fr-rules*] = *merge-coeff-s-impl.refine*

lemma *msort-coeff-s-alt-def*:
‹*msort-coeff-s* $\mathcal{V}$ *xs* = *do* {

let zs = COPY xs;
REC_T
(λ*msortR′ xsa. if length xsa* ≤ *1 then RETURN* (*ASSN-ANNOT monom-s-assn xsa*) *else do* {
let xs1 = *ASSN-ANNOT monom-s-assn* (*take* (*length xsa div 2*) *xsa*);
let xs2 = *ASSN-ANNOT monom-s-assn* (*drop* (*length xsa div 2*) *xsa*);
as ← *msortR′ xs1*;
let as = *ASSN-ANNOT monom-s-assn as*;
bs ← *msortR′ xs2*;
let bs = *ASSN-ANNOT monom-s-assn bs*;
merge-coeff-s $\mathcal{V}$ *zs as bs*
})
xs}⟩
unfolding *msort-coeff-s-def merge-coeff-s-def*[*symmetric*]
msortR-alt-def ASSN-ANNOT-def Let-def COPY-def
by *auto*

sepref-definition *msort-coeff-s-impl*
is ⟨*uncurry msort-coeff-s*⟩
:: ⟨$shared\text{-}vars\text{-}assn^k *_a (monom\text{-}s\text{-}assn)^k \rightarrow_a monom\text{-}s\text{-}assn$⟩
supply [[*goals-limit=1*]]
unfolding *msort-coeff-s-alt-def*
unfolding *var-order′-def*[*symmetric*]
by *sepref*

lemmas [*sepref-fr-rules*] = *msort-coeff-s-impl.refine*

sepref-definition *sort-all-coeffs-s′-impl*
is ⟨*uncurry sort-all-coeffs-s*⟩
:: ⟨$shared\text{-}vars\text{-}assn^k *_a poly\text{-}s\text{-}assn^d \rightarrow_a poly\text{-}s\text{-}assn$⟩
unfolding *sort-all-coeffs-s-def HOL-list.fold-custom-empty*
by *sepref*

lemmas [*sepref-fr-rules*] = *sort-all-coeffs-s′-impl.refine*

lemma *merge-coeffs0-s-alt-def*:
⟨(*RETURN o merge-coeffs0-s*) *p* =
REC_T(λ*f p*.
(*case p of*
[] ⇒ *RETURN* []
| [*p*] => *if snd* (*COPY p*)= *0 then RETURN* [] *else RETURN* [*p*]
| (*a* # *b* # *p*) ⇒
(*let* (*xs*, *n*) = *COPY a*; (*ys*, *m*) = *COPY b in*
if xs = *ys*
then if n + *m* ≠ *0 then f* ((*xs*, *n* + *m*) # (*COPY p*)) *else f p*
else if n = *0 then*
do {*p* ← *f* (*b* # (*COPY p*));
RETURN p}
else do {*p* ← *f* (*b* # (*COPY p*));
RETURN (*a* # *p*)})))
p⟩
unfolding *COPY-def Let-def*
apply (*subst eq-commute*)
apply (*induction p rule*: *merge-coeffs0-s.induct*)
subgoal by (*subst RECT-unfold*, *refine-mono*) *auto*

subgoal by (*subst RECT-unfold*, *refine-mono*) *auto*
subgoal by (*subst RECT-unfold*, *refine-mono*) (*auto simp*: *let-to-bind-conv*)
done

lemma [*sepref-import-param*]: ‹(((=)), ((=))) ∈ ⟨*uint64-nat-rel*⟩ *list-rel* → ⟨*uint64-nat-rel*⟩ *list-rel* → *bool-rel*›
proof −
have ‹*IS-LEFT-UNIQUE* (⟨*uint64-nat-rel*⟩ *list-rel*)›
by (*intro safe-constraint-rules*)
moreover have ‹*IS-RIGHT-UNIQUE* (⟨*uint64-nat-rel*⟩ *list-rel*)›
by (*intro safe-constraint-rules*)
ultimately show *?thesis*
by (*sep-auto simp*: *IS-LEFT-UNIQUE-def single-valued-def*
simp flip: *inv-list-rel-eq*)
qed

lemma *is-pure-monom-s-assn*: ‹*is-pure monom-s-assn*›
‹*is-pure* (*monom-s-assn* $\times_a$ *int-assn*)›
by (*auto simp add*: *list-assn-pure-conv*)

sepref-definition *merge-coeffs0-s-impl*
is ‹*RETURN o merge-coeffs0-s*›
:: ‹*poly-s-assn*^{k} $\rightarrow_a$ *poly-s-assn*›
unfolding *merge-coeffs0-s-alt-def HOL-list.fold-custom-empty*
by *sepref*

lemmas [*sepref-fr-rules*] = *merge-coeffs0-s-impl.refine*

sepref-definition *full-normalize-poly′-impl*
is ‹*uncurry full-normalize-poly-s*›
:: ‹*shared-vars-assn*^{k} $*_a$ *poly-s-assn*^{k} $\rightarrow_a$ *poly-s-assn*›
unfolding *full-normalize-poly-s-def*
by *sepref*

lemma *weak-equality-l-s-alt-def*:
‹*weak-equality-l-s* = *RETURN oo* (λ*p q*. *p* = *q*)›
unfolding *weak-equality-l-s-def weak-equality-l-s-def* **by** (*auto intro*!: *ext*)

lemma [*sepref-import-param*]
: ‹(((=)), ((=))) ∈ ⟨⟨*uint64-nat-rel*⟩ *list-rel* $\times_r$ *int-rel*⟩ *list-rel* → ⟨⟨*uint64-nat-rel*⟩ *list-rel* $\times_r$ *int-rel*⟩ *list-rel* → *bool-rel*›
proof −
let *?A* = ‹⟨⟨*uint64-nat-rel*⟩ *list-rel* $\times_r$ *int-rel*⟩ *list-rel*›
have ‹*IS-LEFT-UNIQUE* (⟨*uint64-nat-rel*⟩ *list-rel*)›
by (*intro safe-constraint-rules*)
then have ‹*IS-LEFT-UNIQUE* (*?A*)›
by (*intro safe-constraint-rules*)
moreover have ‹*IS-RIGHT-UNIQUE* (⟨*uint64-nat-rel*⟩ *list-rel*)›
by (*intro safe-constraint-rules*)
then have ‹*IS-RIGHT-UNIQUE* (*?A*)›
by (*intro safe-constraint-rules*)
ultimately show *?thesis*
by (*sep-auto simp*: *IS-LEFT-UNIQUE-def single-valued-def*

simp flip: *inv-list-rel-eq*)
qed

sepref-definition *weak-equality-l-s-impl*
is ⟨*uncurry weak-equality-l-s*⟩
:: ⟨*poly-s-assn*^{k} $*_a$ *poly-s-assn*^{k} $\rightarrow_a$ *bool-assn*⟩
unfolding *weak-equality-l-s-alt-def*
by *sepref*

code-printing constant *arl-get-u'* ⇀ (*SML*) (*fn/* ()/ =>/ *Array.sub/* ((*fn/* (*a*,*b*)/ =>/ *a*) ((-)),/ *Word32.toInt* ((-))))

abbreviation *polys-s-assn* **where**
⟨*polys-s-assn* ≡ *hm-fmap-assn uint64-nat-assn poly-s-assn*⟩

sepref-definition *import-monom-no-newS-impl*
is ⟨*uncurry* (*import-monom-no-newS* :: (*nat*,*string*)*shared-vars* ⇒ - ⇒(*bool* × -) *nres*)⟩
:: ⟨*shared-vars-assn*^{k} $*_a$ (*list-assn string-assn*)^k $\rightarrow_a$ *bool-assn* $\times_a$ *list-assn uint64-nat-assn*⟩
unfolding *import-monom-no-newS-def HOL-list.fold-custom-empty*
by *sepref*
sepref-register *import-monom-no-newS import-poly-no-newS check-linear-combi-l-pre-err*
lemmas [*sepref-fr-rules*] =
import-monom-no-newS-impl.refine weak-equality-l-s-impl.refine

sepref-definition *import-poly-no-newS-impl*
is ⟨*uncurry* (*import-poly-no-newS* :: (*nat*,*string*)*shared-vars* ⇒ *llist-polynomial* ⇒(*bool* × *sllist-polynomial*) *nres*)⟩
:: ⟨*shared-vars-assn*^{k} $*_a$ *poly-assn*^{k} $\rightarrow_a$ *bool-assn* $\times_a$ *poly-s-assn*⟩
unfolding *import-poly-no-newS-def HOL-list.fold-custom-empty*
by *sepref*

lemmas [*sepref-fr-rules*] =
import-poly-no-newS-impl.refine

definition *check-linear-combi-l-pre-err-impl* **where**
⟨*check-linear-combi-l-pre-err-impl i pd p mem* =
(*if pd then* ''*The polynomial with id* '' @ *show* (*nat-of-uint64 i*) @ '' *was not found*'' *else* '''') @
(*if p then* ''*The co−factor from* '' @ *show* (*nat-of-uint64 i*) @ '' *was empty*'' *else* '''')@
(*if mem then* ''*Memory out*'' *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* ((λ*x y*. *return oo* (*check-linear-combi-l-pre-err-impl x y*))),
uncurry3 (*check-linear-combi-l-pre-err*)) ∈ *uint64-nat-assn*^{k} $*_a$ *bool-assn*^{k} $*_a$ *bool-assn*^{k} $*_a$ *bool-assn*^{k} $\rightarrow_a$ *raw-string-assn*⟩
unfolding *check-linear-combi-l-pre-err-impl-def check-linear-combi-l-pre-err-def list-assn-pure-conv*
apply *sepref-to-hoare*
apply *sep-auto*
done

lemma *vars-llist-in-s-single*: ⟨*RETURN* (*vars-llist-in-s* $\mathcal{V}$ [(*xs*, *a*)]) =

REC_T (*λf xs. case xs of*
[] ⇒ *RETURN True*
| *x # xs* ⇒ *do* {
b ← *is-new-variableS x* $\mathcal{V}$;
if b then RETURN False
else f xs
}) (*xs*)›
apply (*subst eq-commute*)
apply (*cases* $\mathcal{V}$)
apply (*induction xs*)
subgoal
by (*subst RECT-unfold*, *refine-mono*)
(*auto simp*: *vars-llist-in-s-def*)
subgoal
by (*subst RECT-unfold*, *refine-mono*)
(*auto simp*: *vars-llist-in-s-def is-new-variableS-def*)
done

lemma *vars-llist-in-s-alt-def*: ‹(*RETURN oo vars-llist-in-s*) $\mathcal{V}$ *xs* =
REC_T (*λf xs. case xs of*
[] ⇒ *RETURN True*
| (*x*, *a*) # *xs* ⇒ *do* {
b ← *RETURN* (*vars-llist-in-s* $\mathcal{V}$ [(*x*, *a*)]);
if ¬*b then RETURN False*
else f xs
}) *xs*›
apply (*subst eq-commute*)
apply (*cases* $\mathcal{V}$)
apply (*induction xs*)
subgoal
by (*subst RECT-unfold*, *refine-mono*)
(*auto simp*: *vars-llist-in-s-def*)
subgoal
by (*subst RECT-unfold*, *refine-mono*)
(*auto simp*: *vars-llist-in-s-def is-new-variableS-def split*: *prod.splits*)
done

sepref-definition *vars-llist-in-s-impl*
is ‹*uncurry* (*RETURN oo vars-llist-in-s*)›
:: ‹*shared-vars-assn*^{k} $*_a$ *poly-assn*^{k} $\rightarrow_a$ *bool-assn*›
unfolding *vars-llist-in-s-alt-def*
vars-llist-in-s-single
by *sepref*
lemmas [*sepref-fr-rules*] = *vars-llist-in-s-impl.refine*

definition *check-linear-combi-l-s-dom-err-impl* :: ‹- ⇒ *uint64* ⇒ -› **where**
‹*check-linear-combi-l-s-dom-err-impl x p* =
''Poly not found in CL from x '' @ *show* (*nat-of-uint64 p*)›

lemma [*sepref-fr-rules*]:
‹(*uncurry* (*return oo* (*check-linear-combi-l-s-dom-err-impl*)),
uncurry (*check-linear-combi-l-s-dom-err*)) ∈ *poly-s-assn*^{k} $*_a$ *uint64-nat-assn*^{k} $\rightarrow_a$ *raw-string-assn*›
unfolding *check-linear-combi-l-s-dom-err-def check-linear-combi-l-s-dom-err-impl-def list-assn-pure-conv*
apply *sepref-to-hoare*
apply *sep-auto*

done
sepref-register *check-linear-combi-l-s-dom-err-impl mult-poly-s normalize-poly-s*

sepref-definition *normalize-poly-sharedS-impl*
is ‹*uncurry normalize-poly-sharedS*›
:: ‹ *shared-vars-assn*^{k} $*_a$ *poly-assn*^{k} $\rightarrow_a$ *bool-assn* $\times_a$ *poly-s-assn*›
unfolding *normalize-poly-sharedS-def*
by *sepref*

lemmas [*sepref-fr-rules*] = *normalize-poly-sharedS-impl.refine*
mult-poly-s-impl.refine
lemma *merge-coeffs-s-alt-def*:
‹(*RETURN o merge-coeffs-s*) *p* =
REC_T(λ*f p*.
(*case p of*
[] ⇒ *RETURN* []
| [-] => *RETURN p*
| ((*xs*, *n*) # (*ys*, *m*) # *p*) ⇒
(*if xs* = *ys*
then if n + *m* ≠ *0 then f* ((*xs*, *n* + *m*) # *COPY p*) *else f p*
else do {*p* ← *f* ((*ys*, *m*) # *p*); *RETURN* ((*xs*, *n*) # *p*)})))
p›
apply (*subst eq-commute*)
apply (*induction p rule*: *merge-coeffs-s.induct*)
subgoal by (*subst RECT-unfold*, *refine-mono*) *auto*
subgoal by (*subst RECT-unfold*, *refine-mono*) *auto*
subgoal for *x p y q*
by (*subst RECT-unfold*, *refine-mono*) *auto*
done

sepref-definition *merge-coeffs-s-impl*
is ‹(*RETURN o merge-coeffs-s*)›
:: ‹*poly-s-assn*^{k} $\rightarrow_a$ *poly-s-assn*›
unfolding *merge-coeffs-s-alt-def*
HOL-list.fold-custom-empty
by *sepref*

lemmas [*sepref-fr-rules*] = *merge-coeffs-s-impl.refine*

sepref-definition *normalize-poly-s-impl*
is ‹*uncurry normalize-poly-s*›
:: ‹*shared-vars-assn*^{k} $*_a$ *poly-s-assn*^{k} $\rightarrow_a$ *poly-s-assn*›
unfolding *normalize-poly-s-def*
by *sepref*

lemmas [*sepref-fr-rules*] = *normalize-poly-s-impl.refine*

sepref-definition *mult-poly-full-s-impl*
is ‹*uncurry2 mult-poly-full-s*›
:: ‹*shared-vars-assn*^{k} $*_a$ *poly-s-assn*^{k}$*_a$ *poly-s-assn*^{k} $\rightarrow_a$ *poly-s-assn*›
unfolding *mult-poly-full-s-def*
by *sepref*

lemmas [*sepref-fr-rules*] = *mult-poly-full-s-impl.refine*
add-poly-l-prep-impl.refine

sepref-register *add-poly-l-s*

sepref-definition *linear-combi-l-prep-s-impl*
  is ‹*uncurry3 linear-combi-l-prep-s*›
  :: ‹*uint64-nat-assn*^{k} $*_a$ *polys-s-assn*^{k} $*_a$ *shared-vars-assn*^{k} $*_a$
 (*list-assn* (*poly-assn* $\times_a$ *uint64-nat-assn*))^d $\to_a$ *poly-s-assn* $\times_a$ (*list-assn* (*poly-assn* $\times_a$ *uint64-nat-assn*))
$\times_a$ *status-assn raw-string-assn*
  ›
  supply [[*goals-limit=1*]]
  unfolding *linear-combi-l-prep-s-def*
    in-dom-m-lookup-iff
    fmlookup'-def[*symmetric*] *conv-to-is-Nil*
  unfolding *is-Nil-def*
    HOL-list.fold-custom-empty
  apply (*rewrite* **in** ‹*op-HOL-list-empty*› *annotate-assn*[**where** *A*=‹*poly-s-assn*›])
  by *sepref*

lemmas [*sepref-fr-rules*] = *linear-combi-l-prep-s-impl.refine*

definition *check-linear-combi-l-s-mult-err-impl* :: ‹- ⇒ - ⇒ -› **where**
  ‹*check-linear-combi-l-s-mult-err-impl x p* =
  ''Unequal polynom found in CL '' @ *show* (*map* (λ(*a*,*b*). (*map nat-of-uint64 a*, *b*)) *p*) @
  '' but '' @ *show* (*map* (λ(*a*,*b*). (*map nat-of-uint64 a*, *b*)) *x*)›

lemma [*sepref-fr-rules*]:
  ‹(*uncurry* (*return oo* (*check-linear-combi-l-s-mult-err-impl*)),
    uncurry (*check-linear-combi-l-s-mult-err*)) ∈ *poly-s-assn*^{k} $*_a$ *poly-s-assn*^{k} $\to_a$ *raw-string-assn*›
  unfolding *check-linear-combi-l-s-mult-err-impl-def check-linear-combi-l-s-mult-err-def list-assn-pure-conv*
   apply *sepref-to-hoare*
   apply *sep-auto*
   done

sepref-definition *check-linear-combi-l-s-impl*
  is ‹*uncurry5 check-linear-combi-l-s*›
  :: ‹*poly-s-assn*^{k} $*_a$ *polys-s-assn*^{k} $*_a$ *shared-vars-assn*^{k} $*_a$ *uint64-nat-assn*^{k} $*_a$
 (*list-assn* (*poly-assn* $\times_a$ *uint64-nat-assn*))^d $*_a$ *poly-assn*^{k} $\to_a$ *status-assn raw-string-assn* $\times_a$ *poly-s-assn*
  ›
  unfolding *check-linear-combi-l-s-def*
    in-dom-m-lookup-iff
    fmlookup'-def[*symmetric*]
  by *sepref*

sepref-register *fmlookup'*
lemma *check-extension-l2-s-alt-def*:
  ‹*check-extension-l2-s spec A* $\mathcal{V}$ *i v p* = *do* {
  n ← *is-new-variableS v* $\mathcal{V}$;
  let t = *fmlookup' i A*;
  pre ← *RETURN* (*t* = *None*);
  let pre = *pre* ∧ *n*;
  let nonew = *vars-llist-in-s* $\mathcal{V}$ *p*;
  (*mem*, *p*, $\mathcal{V}$) ← *import-polyS* $\mathcal{V}$ *p*;
  let pre = (*pre* ∧ ¬*alloc-failed mem*);
  if ¬*pre*
  then do {

```
    c ← check-extension-l-dom-err i;
    RETURN (error-msg i c, [], 𝒱, 0)
  } else do {
      if ¬nonew
      then do {
        c ← check-extension-l-s-new-var-multiple-err v p;
        RETURN (error-msg i c, [], 𝒱, 0)
      }
      else do {
        (mem′, 𝒱, v′) ← import-variableS v 𝒱;
        if alloc-failed mem′
        then do {
          c ← check-extension-l-dom-err i;
          RETURN (error-msg i c, [], 𝒱, 0)
        } else
        do {
         p2 ← mult-poly-full-s 𝒱 p p;
         let p′′ = map (λ(a,b). (a, −b)) p;
         q ← add-poly-l-s 𝒱 (p2, p′′);
         eq ← weak-equality-l-s q [];
         if eq then do {
           RETURN (CSUCCESS, p, 𝒱, v′)
         } else do {
          c ← check-extension-l-s-side-cond-err v p p′′ q;
          RETURN (error-msg i c, [], 𝒱, v′)
        }
      }
     }
  }
  }›
  unfolding check-extension-l2-s-def fmlookup′-def[symmetric] Let-def
     in-dom-m-lookup-iff
   by (auto intro!: bind-cong[OF refl])

definition uminus-poly :: ‹- ⇒ -› where
  ‹uminus-poly p′ = map (λ(a, b). (a, − b)) p′›

lemma [sepref-import-param]: ‹(uminus-poly, uminus-poly) ∈ ⟨monom-s-rel ×r int-rel⟩list-rel → ⟨monom-s-rel
×r int-rel⟩list-rel›
proof −
  have ‹(a, a′) ∈ ⟨monom-s-rel ×r int-rel⟩list-rel ⟹
    (EPAC-Efficient-Checker-Synthesis.uminus-poly a,
     EPAC-Efficient-Checker-Synthesis.uminus-poly a′)
    ∈ ⟨monom-s-rel ×r int-rel⟩list-rel› for a a′
    apply (induction a arbitrary: a′)
    subgoal by (auto simp: uminus-poly-def)
    subgoal for a as a′
      by (cases a′; cases a)
        (auto simp: uminus-poly-def)
    done
  then show ?thesis
    by (auto intro!: frefI)
qed

sepref-register import-monomS import-polyS
```

sepref-definition *import-monomS-impl*
is ‹*uncurry import-monomS*›
:: ‹*shared-vars-assn*^{d} $*_a$ *monom-assn*^{k} $\rightarrow_a$ *memory-allocation-assn* $\times_a$ *monom-s-assn* $\times_a$ *shared-vars-assn*›
supply [[*goals-limit=1*]]
unfolding *import-monomS-def*
HOL-list.fold-custom-empty
by *sepref*

lemmas [*sepref-fr-rules*] =
import-monomS-impl.refine

sepref-definition *import-polyS-impl*
is ‹*uncurry import-polyS*›
:: ‹*shared-vars-assn*^{d} $*_a$ *poly-assn*^{k} $\rightarrow_a$ *memory-allocation-assn* $\times_a$ *poly-s-assn* $\times_a$ *shared-vars-assn*›
supply [[*goals-limit=1*]]
unfolding *import-polyS-def*
HOL-list.fold-custom-empty
by *sepref*

lemmas [*sepref-fr-rules*] =
import-polyS-impl.refine

definition *check-extension-l-s-new-var-multiple-err-impl* :: ‹*String.literal* ⇒ - ⇒ -› **where**
‹*check-extension-l-s-new-var-multiple-err-impl x p* =
''*Variable already defined* '' @ *show x* @
'' *but* '' @ *show* (*map* (λ(*a*,*b*). (*map nat-of-uint64 a*, *b*)) *p*)›

lemma [*sepref-fr-rules*]:
‹((*uncurry* (*return oo* (*check-extension-l-s-new-var-multiple-err-impl*)),
uncurry (*check-extension-l-s-new-var-multiple-err*)) ∈ *string-assn*^{k} $*_a$ *poly-s-assn*^{k} $\rightarrow_a$ *raw-string-assn*›
unfolding *check-extension-l-s-new-var-multiple-err-impl-def check-extension-l-s-new-var-multiple-err-def list-assn-pure-conv*
apply *sepref-to-hoare*
apply *sep-auto*
done

definition *check-extension-l-s-side-cond-err-impl* :: ‹*String.literal* ⇒ - ⇒ -› **where**
‹*check-extension-l-s-side-cond-err-impl x p p' q'* =
''*p^2*− *p* != *0* '' @ *show x* @
'' *but* '' @ *show* (*map* (λ(*a*,*b*). (*map nat-of-uint64 a*, *b*)) *p*) @
'' *and* '' @ *show* (*map* (λ(*a*,*b*). (*map nat-of-uint64 a*, *b*)) *p'*) @
'' *and* '' @ *show* (*map* (λ(*a*,*b*). (*map nat-of-uint64 a*, *b*)) *q'*)›

abbreviation *comp4* (**infixl** *oooo 55*) **where** *f oooo g* ≡ λ*x*. *f ooo* (*g x*)
abbreviation *comp5* (**infixl** *ooooo 55*) **where** *f ooooo g* ≡ λ*x*. *f oooo* (*g x*)

lemma [*sepref-fr-rules*]:
‹((*uncurry3* (*return oooo* (*check-extension-l-s-side-cond-err-impl*)),
uncurry3 (*check-extension-l-s-side-cond-err*)) ∈ *string-assn*^{k} $*_a$ *poly-s-assn*^{k}$*_a$ *poly-s-assn*^{k}$*_a$ *poly-s-assn*^{k}
$\rightarrow_a$ *raw-string-assn*›
unfolding *check-extension-l-s-side-cond-err-impl-def check-extension-l-s-side-cond-err-def list-assn-pure-conv*
apply *sepref-to-hoare*
apply *sep-auto*
done

sepref-register *mult-poly-full-s weak-equality-l-s check-extension-l-s-side-cond-err check-extension-l2-s check-linear-combi-l-s is-cfailed check-del-l*

sepref-definition *check-extension-l-impl*
  is ⟨*uncurry5 check-extension-l2-s*⟩
  :: ⟨$poly\text{-}s\text{-}assn^k *_a polys\text{-}s\text{-}assn^k *_a shared\text{-}vars\text{-}assn^d *_a uint64\text{-}nat\text{-}assn^k *_a$
  $string\text{-}assn^k *_a poly\text{-}assn^k \rightarrow_a status\text{-}assn\ raw\text{-}string\text{-}assn \times_a poly\text{-}s\text{-}assn \times_a shared\text{-}vars\text{-}assn \times_a$
uint64-nat-assn
  ⟩
  supply [[*goals-limit=1*]]
  unfolding *check-extension-l2-s-alt-def*
    in-dom-m-lookup-iff
    fmlookup'-def[*symmetric*]
    not-not is-None-def
    uminus-poly-def[*symmetric*]
    HOL-list.fold-custom-empty
    zero-uint64-nat-def[*symmetric*]
  by *sepref*

lemma [*sepref-fr-rules*]:
  ⟨(*return o is-cfailed*, *RETURN o is-cfailed*) ∈ $(status\text{-}assn\ raw\text{-}string\text{-}assn)^k \rightarrow_a bool\text{-}assn$⟩
  apply *sepref-to-hoare*
  apply (*sep-auto*)
  apply (*case-tac x*; *case-tac xi*; *sep-auto*)+
  done

sepref-definition *check-del-l-impl*
  is ⟨*uncurry2 check-del-l*⟩
  :: ⟨$poly\text{-}s\text{-}assn^k *_a polys\text{-}s\text{-}assn^k *_a uint64\text{-}nat\text{-}assn^k \rightarrow_a status\text{-}assn\ raw\text{-}string\text{-}assn$⟩
  unfolding *check-del-l-def*
  by *sepref*

lemmas [*sepref-fr-rules*] =
  check-extension-l-impl.refine
  check-linear-combi-l-s-impl.refine
  check-del-l-impl.refine

sepref-definition *PAC-checker-l-step-s-impl*
  is ⟨*uncurry2 PAC-checker-l-step-s*⟩
  :: ⟨$poly\text{-}s\text{-}assn^k *_a (status\text{-}assn\ raw\text{-}string\text{-}assn \times_a shared\text{-}vars\text{-}assn \times_a polys\text{-}s\text{-}assn)^d *_a$
    $(pac\text{-}step\text{-}rel\text{-}assn\ (uint64\text{-}nat\text{-}assn)\ poly\text{-}assn\ string\text{-}assn)^k \rightarrow_a status\text{-}assn\ raw\text{-}string\text{-}assn \times_a$
$shared\text{-}vars\text{-}assn \times_a polys\text{-}s\text{-}assn$
  ⟩
  supply [[*goals-limit* = *1*]]
  supply [*intro*] = *is-Mult-lastI*
  unfolding *PAC-checker-l-step-s-def Let-def*
    pac-step.case-eq-if
    HOL-list.fold-custom-empty
  by *sepref*

lemmas [*sepref-fr-rules*] = *PAC-checker-l-step-s-impl.refine*

fun *vars-llist-s2* :: ⟨- ⇒ - *list*⟩ **where**

‹*vars-llist-s2* [] = []› |
‹*vars-llist-s2* ((*a*,-) # *xs*) = *a* @ *vars-llist-s2* *xs*›

lemma [*sepref-import-param*]:
‹(*vars-llist-s2*, *vars-llist-s2*) ∈ ⟨⟨*string-rel*⟩*list-rel* $\times_r$ *int-rel*⟩*list-rel* → ⟨*string-rel*⟩*list-rel*›
apply (*intro fun-relI*)
subgoal for *a b*
apply (*induction a arbitrary*: *b*)
subgoal by *auto*
subgoal for *a as b*
by (*cases a*, *cases b*)
(*force simp*: *list-rel-append1*)+
done
done
sepref-register *PAC-checker-l-step-s*
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*)›
apply (*intro ext*)
apply (*case-tac x*; *case-tac xa*)
apply *simp-all*
apply (*simp-all add*: *relAPP-def p2rel-def pure-def*)
unfolding *pure-def*[*symmetric*] *list-assn-pure-conv*
apply (*auto simp*: *pure-def relAPP-def*)
done

lemma *safe-epac-step-rel-assn*[*safe-constraint-rules*]:
‹*CONSTRAINT is-pure K* ⟹ *CONSTRAINT is-pure V* ⟹ *CONSTRAINT is-pure R* ⟹
CONSTRAINT is-pure (*EPAC-Checker.pac-step-rel-assn K V R*)›
by (*auto simp*: *step-rewrite-pure*(*1*)[*symmetric*] *is-pure-conv*)

sepref-definition *PAC-checker-l-s-impl*
is ‹*uncurry3 PAC-checker-l-s*›
:: ‹*poly-s-assn*^{k} $*_a$ (*shared-vars-assn* $\times_a$ *polys-s-assn*)^d $*_a$(*status-assn raw-string-assn*)^d $*_a$
(*list-assn* (*pac-step-rel-assn* (*uint64-nat-assn*) *poly-assn string-assn*))^d $\rightarrow_a$
status-assn raw-string-assn $\times_a$ *shared-vars-assn* $\times_a$ *polys-s-assn*
›
supply [[*goals-limit* = *1*]]
supply [*intro*] = *is-Mult-lastI*
unfolding *PAC-checker-l-s-def Let-def*
pac-step.case-eq-if
neq-Nil-conv
conv-to-is-Nil is-Nil-def
by *sepref*

lemmas [*sepref-fr-rules*] = *PAC-checker-l-s-impl.refine*

definition *memory-out-msg* :: ‹*string*› **where**
‹*memory-out-msg* = *″memory out″*›

lemma [*sepref-fr-rules*]: ‹(*uncurry0* (*return memory-out-msg*), *uncurry0* (*RETURN memory-out-msg*))
∈ *unit-assn*^{k} $\rightarrow_a$ *raw-string-assn*›
unfolding *memory-out-msg-def*
by *sepref-to-hoare sep-auto*

definition (**in** −) *remap-polys-l2-with-err-s* :: ‹*llist-polynomial* ⇒ *llist-polynomial* ⇒ (*nat*, *llist-polynomial*) *fmap* ⇒ (*nat*, *string*) *shared-vars* ⇒
(*string code-status* × (*nat*, *string*) *shared-vars* × (*nat*, *sllist-polynomial*) *fmap* × *sllist-polynomial*) *nres*› **where**
‹*remap-polys-l2-with-err-s spec spec0 A* (𝒱 :: (*nat*, *string*) *shared-vars*) = *do*{
ASSERT(*vars-llist spec* ⊆ *vars-llist spec0*);
n ← *upper-bound-on-dom A*;
(*mem*, 𝒱) ← *import-variablesS* (*vars-llist-s2 spec0*) 𝒱;
(*mem'*, *spec*, 𝒱) ← *if* ¬*alloc-failed mem then import-polyS* 𝒱 *spec else RETURN* (*mem*, [], 𝒱);
failed ← *RETURN* (*alloc-failed mem* ∨ *alloc-failed mem'* ∨ *n* ≥ *2^64*);
if failed
then do {
c ← *remap-polys-l-dom-err*;
RETURN (*error-msg* (*0*::*nat*) *c*, 𝒱, *fmempty*, [])
}
else do {
(*err*, *A*, 𝒱) ← *nfoldli* ([*0*..<*n*]) (λ(*err*, *A'*, 𝒱). ¬*is-cfailed err*)
(λ*i* (*err*, *A'* :: (*nat*, *sllist-polynomial*) *fmap*, 𝒱 :: (*nat*,*string*) *shared-vars*).
if i ∈# *dom-m A*
then do {
(*err'*, *p*, 𝒱 :: (*nat*,*string*) *shared-vars*) ← *import-polyS* (𝒱 :: (*nat*,*string*) *shared-vars*) (*the* (*fmlookup A i*));
if alloc-failed err' then RETURN((*CFAILED ''memory out''*, *A'*, 𝒱 :: (*nat*,*string*) *shared-vars*))
else do {
p ← *full-normalize-poly-s* 𝒱 *p*;
eq ← *weak-equality-l-s p spec*;
RETURN((*if eq then CFOUND else CSUCCESS*), *fmupd i p A'*, 𝒱 :: (*nat*,*string*) *shared-vars*)
}
} *else RETURN* (*err*, *A'*, 𝒱 :: (*nat*,*string*) *shared-vars*))
(*CSUCCESS*, *fmempty* :: (*nat*, *sllist-polynomial*) *fmap*, 𝒱 :: (*nat*,*string*) *shared-vars*);
RETURN (*err*, 𝒱, *A*, *spec*)
}}›

lemma *set-vars-llist-s2* [*simp*]: ‹*set* (*vars-llist-s2 b*) = *vars-llist b*›
by (*induction b*)
(*auto simp*: *vars-llist-def*)

sepref-register *upper-bound-on-dom import-variablesS vars-llist-s2 memory-out-msg*

sepref-definition *import-variablesS-impl*
is ‹*uncurry import-variablesS*›
:: ‹(*list-assn string-assn*)^k $*_a$ *shared-vars-assn*^{d} $\rightarrow_a$ *memory-allocation-assn* $\times_a$ *shared-vars-assn*›
unfolding *import-variablesS-def*
by *sepref*

lemmas [*sepref-fr-rules*] =
import-variablesS-impl.refine full-normalize-poly'-impl.refine
lemma [*sepref-fr-rules*]:
‹*CONSTRAINT is-pure R* ⟹ ((*return o CFAILED*), *RETURN o CFAILED*) ∈ $R^k \rightarrow_a$ *status-assn R*›
apply *sepref-to-hoare*
apply *sep-auto*
by (*smt ent-refl-true is-pure-conv merge-pure-star pure-def*)

sepref-definition *remap-polys-l2-with-err-s-impl*
is ⟨*uncurry3 remap-polys-l2-with-err-s*⟩
:: ⟨*poly-assn*^{k} $*_a$ *poly-assn*^{k} $*_a$ *polys-assn-input*^{k} $*_a$ *shared-vars-assn*^{d} $\rightarrow_a$
status-assn raw-string-assn $\times_a$ *shared-vars-assn* $\times_a$ *polys-s-assn* $\times_a$ *poly-s-assn*⟩
supply [[*goals-limit=1*]]
supply [*split*] = *option.splits*
unfolding *remap-polys-l2-with-err-s-def pow-2-64*
in-dom-m-lookup-iff
fmlookup'-def[*symmetric*]
memory-out-msg-def[*symmetric*]
op-fmap-empty-def[*symmetric*] *while-eq-nfoldli*[*symmetric*]
unfolding
HOL-list.fold-custom-empty
apply (*subst while-upt-while-direct*)
apply *simp*
apply (*rewrite* **in** ⟨(-, ⌑, -)⟩ *annotate-assn*[**where** *A*=⟨*polys-s-assn*⟩])
apply (*rewrite at* ⟨*fmupd* ⌑⟩ *uint64-of-nat-conv-def*[*symmetric*])
by *sepref*

lemmas [*sepref-fr-rules*] =
remap-polys-l2-with-err-s-impl.refine

definition *full-checker-l-s2*
:: ⟨*llist-polynomial* ⇒ (*nat*, *llist-polynomial*) *fmap* ⇒ (-, *string*, *nat*) *pac-step list* ⇒
(*string code-status* × -) *nres*⟩
where
⟨*full-checker-l-s2 spec A st* = *do* {
spec' ← *full-normalize-poly spec*;
(*b*, $\mathcal{V}$, *A*, *spec'*) ← *remap-polys-l2-with-err-s spec' spec A* ({#}, *fmempty*, *fmempty*);
if is-cfailed b
then RETURN (*b*, $\mathcal{V}$, *A*)
else do {
PAC-checker-l-s spec' ($\mathcal{V}$, *A*) *b st*
}
}⟩

sepref-register *remap-polys-l2-with-err-s full-checker-l-s2 PAC-checker-l-s*

sepref-definition *full-checker-l-s2-impl*
is ⟨*uncurry2 full-checker-l-s2*⟩
:: ⟨*poly-assn*^{k} $*_a$ *polys-assn-input*^{k} $*_a$ (*list-assn* (*pac-step-rel-assn* (*uint64-nat-assn*) *poly-assn string-assn*))^k
$\rightarrow_a$
status-assn raw-string-assn $\times_a$ *shared-vars-assn* $\times_a$ *polys-s-assn*⟩
unfolding *full-checker-l-s2-def*
empty-shared-vars-def[*symmetric*]
by *sepref*

7 Correctness theorem

context *poly-embed*
begin

definition *fully-epac-assn* **where**
⟨*fully-epac-assn* = (*list-assn*
(*hr-comp* (*pac-step-rel-assn uint64-nat-assn poly-assn string-assn*)

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

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 represention of the PAC proofs.

lemma *remap-polys-l2-with-err-s-remap-polys-s-with-err*:
  assumes ‹((*spec*, *a*, *b*, *c*), (*spec'*, *a'*, *c'*, *b'*)) ∈ *Id*›
  shows ‹*remap-polys-l2-with-err-s spec a b c*
    ≤ ⇓ *Id*
  (*remap-polys-s-with-err spec' a' b' c'*)›
proof −
  have [*refine*]: ‹(*A*, *A'*) ∈ *Id* ⟹ *upper-bound-on-dom A*
    ≤ ⇓ {(*n*, *dom*). *dom* = *set* [*0*..<*n*]} (*SPEC* (λ*dom*. *set-mset* (*dom-m A'*) ⊆ *dom* ∧ *finite dom*))› **for** *A A'*
    unfolding *upper-bound-on-dom-def*
    apply (*rule RES-refine*)
    apply (*auto simp*: *upper-bound-on-dom-def*)
    done
  have *3*: ‹(*n*, *dom*) ∈ {(*n*, *dom*). *dom* = *set* [*0*..<*n*]} ⟹
    ([*0*..<*n*], *dom*) ∈ ⟨*nat-rel*⟩*list-set-rel*› **for** *n dom*
    by (*auto simp*: *list-set-rel-def br-def*)
  have *4*: ‹(*p*,*q*) ∈ *Id* ⟹
    weak-equality-l p spec ≤ ⇓*Id* (*weak-equality-l q spec*)› **for** *p q spec*
    by *auto*

  have *6*: ‹*a* = *b* ⟹ (*a*, *b*) ∈ *Id*› **for** *a b*
    by *auto*

  have *id*: ‹*f*=*g* ⟹ *f* ≤⇓*Id g*› **for** *f g*
    by *auto*
  have [*simp*]: ‹*vars-llist-s2 x* = *vars-llist-l x*› **for** *x*
    by (*induction x rule*: *vars-llist-s2.induct*) *auto*
  show *?thesis*
    supply [[*goals-limit*=*1*]]

unfolding *remap-polys-l2-with-err-s-def remap-polys-s-with-err-def*
apply (*refine-rcg*
LFOc-refine[**where** R= ‹{((a,b,c), (a′,b′,c′)). ((a,b,c), (a′,c′,b′))∈*Id*}›])
subgoal using *assms* **by** *auto*
subgoal using *assms* **by** *auto*
apply (*rule id*)
subgoal using *assms* **by** *auto*
subgoal using *assms* **by** *auto*
apply (*rule id*)
subgoal using *assms* **by** *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
apply (*rule 3*)
subgoal by *auto*
subgoal by *auto*
subgoal using *assms* **by** *auto*
apply (*rule id*)
subgoal using *assms* **by** *auto*
subgoal by *auto*
subgoal by *auto*
apply (*rule id*)
subgoal by *auto*
apply (*rule id*)
subgoal unfolding *weak-equality-l-s′-def* **by** *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
done
qed

lemma *full-checker-l-s2-full-checker-l-s*:
‹(*uncurry2 full-checker-l-s2*, *uncurry2 full-checker-l-s*) ∈ (*Id* $\times_r$ *Id*) $\times_r$ *Id* $\rightarrow_f$ ⟨*Id*⟩*nres-rel*›
proof −
have *id*: ‹*f*=*g* ⟹ *f* ≤⇓*Id g*› **for** *f g*
by *auto*
show *?thesis*
apply (*intro frefI nres-relI*)
unfolding *uncurry-def*
apply *clarify*
unfolding *full-checker-l-s2-def*
full-checker-l-s-def
apply (*refine-rcg remap-polys-l2-with-err-s-remap-polys-s-with-err*)
apply (*rule id*)
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
apply (*rule id*)
subgoal by *auto*
done
qed

lemma *full-poly-input-assn-alt-def*:
‹*full-poly-input-assn* = (*hr-comp*
(*hr-comp* (*hr-comp* *polys-assn-input* (⟨*nat-rel*, *Id*⟩*fmap-rel*))
(⟨*nat-rel*, *fully-unsorted-poly-rel* *O* *mset-poly-rel*⟩*fmap-rel*))
polys-rel)›
proof −
have [*simp*]: ‹⟨*nat-rel*, *Id*⟩*fmap-rel* = *Id*›
apply (*auto simp*: *fmap-rel-def*)
by (*metis* (*no-types*, *hide-lams*) *fmap-ext-fmdom fmlookup-dom-iff fset-eqI option.sel*)
show *?thesis*
unfolding *full-poly-input-assn-def*
by *auto*
qed

lemma *PAC-full-correctness*:
‹(*uncurry2 full-checker-l-s2-impl*,
uncurry2 (λ*spec A* -. *PAC-checker-specification spec A*))
∈ *full-poly-assn*^{k} $*_a$ *full-poly-input-assn*^{k} $*_a$
fully-epac-assn^k $\rightarrow_a$ *hr-comp* (*status-assn raw-string-assn* $\times_a$ *shared-vars-assn* $\times_a$ *polys-s-assn*)
{((*err*, -), *err*′, -). (*err*, *err*′) ∈ *code-status-status-rel*}›
proof −
have *1*: ‹(*uncurry2 full-checker-l-s2*, *uncurry2* (λ*spec A* -. *PAC-checker-specification spec A*))
∈ (⟨⟨*Id*⟩*list-rel* $\times_r$ *int-rel*⟩*list-rel* *O* *fully-unsorted-poly-rel* *O* *mset-poly-rel* $\times_r$
(⟨*nat-rel*, *Id*⟩*fmap-rel* *O* ⟨*nat-rel*, *fully-unsorted-poly-rel* *O* *mset-poly-rel*⟩*fmap-rel*) *O*
polys-rel) $\times_r$
⟨*p2rel*
(⟨*nat-rel*, *fully-unsorted-poly-rel* *O* *mset-poly-rel*,
var-rel⟩*EPAC-Checker.pac-step-rel-raw*)⟩*list-rel* $\rightarrow_f$ ⟨((({((*err*, -), *err*′, -).
(*err*, *err*′) ∈ *Id*} *O*
{((*b*, *A*, *st*), *b*′, *A*′, *st*′).
(¬ *is-cfailed b* ⟶ (*A*, *A*′) ∈ {(*x*, *y*). *y* = *set-mset x*} ∧ (*st*, *st*′) ∈ *Id*) ∧
(*b*, *b*′) ∈ *Id*}) *O*
{((*err*, 𝒱, *A*), *err*′, 𝒱′, *A*′).
((*err*, 𝒱, *A*), *err*′, 𝒱′, *A*′)
∈ *code-status-status-rel* $\times_r$
vars-rel2 err $\times_r$
{(*xs*, *ys*).
¬ *is-cfailed err* ⟶
(*xs*, *ys*) ∈ ⟨*nat-rel*, *sorted-poly-rel* *O* *mset-poly-rel*⟩*fmap-rel* ∧
(∀ *i*∈#*dom-m xs*. *vars-llist* (*xs* ∝ *i*) ⊆ 𝒱)}}) *O*
{((*st*, *G*), *st*′, *G*′).
(*st*, *st*′) ∈ *status-rel* ∧ (*st* ≠ *FAILED* ⟶ (*G*, *G*′) ∈ *Id* $\times_r$ *polys-rel*)}⟩*nres-rel*›
using *full-checker-l-s2-full-checker-l-s*[
FCOMP full-checker-l-s-full-checker-l-prep′,
FCOMP full-checker-l-prep-full-checker-l2′,
FCOMP full-checker-l-full-checker′,
FCOMP full-checker-spec′,
unfolded full-poly-assn-def[*symmetric*]
full-poly-input-assn-def[*symmetric*]
fully-epac-assn-def[*symmetric*]
code-status-assn-def[*symmetric*]
full-vars-assn-def[*symmetric*]
polys-rel-full-polys-rel
hr-comp-prod-conv
full-polys-assn-def[*symmetric*]

```
    full-poly-input-assn-alt-def[symmetric]] by auto
  have 2: ‹A ⊆ B ⟹ ⟨A⟩nres-rel ⊆ ⟨B⟩nres-rel› for A B
    by (auto simp: nres-rel-def conc-fun-R-mono conc-trans-additional(6))

  have 3: ‹(uncurry2 full-checker-l-s2, uncurry2 (λspec A -. PAC-checker-specification spec A))
    ∈ (⟨⟨Id⟩list-rel ×r int-rel⟩list-rel O fully-unsorted-poly-rel O mset-poly-rel ×r
    (⟨nat-rel, Id⟩fmap-rel O ⟨nat-rel, fully-unsorted-poly-rel O mset-poly-rel⟩fmap-rel) O
    polys-rel) ×r
    ⟨p2rel
    (⟨nat-rel, fully-unsorted-poly-rel O mset-poly-rel,
    var-rel⟩EPAC-Checker.pac-step-rel-raw)⟩list-rel →f
    ⟨{((err, -), err', -). (err, err') ∈ code-status-status-rel}⟩nres-rel›
    apply (rule set-mp[OF - 1])
    unfolding fref-param1[symmetric]
    apply (rule fun-rel-mono)
    apply auto[]
    apply (rule 2)
    apply auto
    done

  have 4: ‹⟨nat-rel, Id⟩fmap-rel = Id›
    apply (auto simp: fmap-rel-def)
    by (metis (no-types, hide-lams) fmap-ext-fmdom fmlookup-dom-iff fset-eqI option.sel)
  have H: ‹full-poly-assn = (hr-comp poly-assn
    (⟨⟨Id⟩list-rel ×r int-rel⟩list-rel O fully-unsorted-poly-rel O mset-poly-rel))›
    ‹full-poly-input-assn = hr-comp polys-assn-input
   ((Id O ⟨nat-rel, fully-unsorted-poly-rel O mset-poly-rel⟩fmap-rel) O polys-rel)›
    unfolding full-poly-assn-def fully-epac-assn-def full-poly-input-assn-def
      hr-comp-assoc O-assoc
    by auto
  show ?thesis
    using full-checker-l-s2-impl.refine[FCOMP 3]
    unfolding full-poly-assn-def[symmetric]
      full-poly-input-assn-def[symmetric]
      fully-epac-assn-def[symmetric]
      code-status-assn-def[symmetric]
      full-vars-assn-def[symmetric]
      polys-rel-full-polys-rel
      hr-comp-prod-conv
      full-polys-assn-def[symmetric]
      full-poly-input-assn-alt-def[symmetric]
      4 H[symmetric]
    by auto
qed
```

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
end

theory *EPAC-Checker-MLton*
  imports *EPAC-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 CL 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
    ‹*code/no-sharing/parser.sml*›
    ‹*code/no-sharing/pasteque.sml*›
    ‹*code/no-sharing/pasteque.mlb*›
  where ‹*fn dir* =>
  let
    val exec = *Generated-Files.execute* (*Path.append dir* (*Path.basic code*));
     val - = *exec* ‹*Copy files*›
    (*cp checker.ML* ^((*File.bash-path* **path** ‹$*ISAFOL*›) ^/*PAC-Checker2/code/no-sharing/checker.ML*));
     val - = *exec* ‹*Copy files*›
      (*cp no-sharing/∗ .*);
    val - = *exec* ‹*Copy files*›
      (*ls .*) |> @{*print*};
    val - =
        exec ‹*Compilation*›
          (*File.bash-path* **path** ‹$*ISABELLE-MLTON*› ^ ^
            −*const 'MLton.safe false'* −*verbose 1* −*default*−*type int64* −*output pasteque* ^
            −*codegen native* −*inline 700* −*cc*−*opt* −*O3 pasteque.mlb*);
    in () *end*›

end

theory *EPAC-Efficient-Checker-MLton*
  imports *EPAC-Efficient-Checker-Synthesis*
begin
local-setup ‹

```
  let
    val version =
      trim-line (#1 (Isabelle-System.bash-output (cd $ISAFOL/ && git rev−parse −−short HEAD ||
echo unknown)))
  in
    Local-Theory.define
      ((binding ‹version›, NoSyn),
        ((binding ‹version-def›, []), HOLogic.mk-literal version)) #> #2
  end
›
```

declare *version-def* [*code*]

definition *uint32-of-uint64* :: ‹*uint64* ⇒ *uint32*› **where**
‹*uint32-of-uint64 n* = *uint32-of-nat* (*nat-of-uint64 n*)›

lemma [*code*]: ‹*hashcode n* = *uint32-of-uint64* (*n AND 4294967295*)› **for** *n* :: *uint64*
unfolding *hashcode-uint64-def uint32-of-uint64-def* **by** *auto*

code-printing code-module *Uint64* ⇀ (*SML*) ‹(∗ *Test that words can handle numbers between 0 and 63* ∗)

```
val - = if 6 <= Word.wordSize then () else raise (Fail (wordSize less than 6));

structure Uint64 : sig
  eqtype uint64;
  val zero : uint64;
  val one : uint64;
  val fromInt : IntInf.int −> uint64;
  val toInt : uint64 −> IntInf.int;
  val toFixedInt : uint64 −> Int.int;
  val toLarge : uint64 −> LargeWord.word;
  val fromLarge : LargeWord.word −> uint64
  val fromFixedInt : Int.int −> uint64
  val toWord32: uint64 −> Word32.word
  val plus : uint64 −> uint64 −> uint64;
  val minus : uint64 −> uint64 −> uint64;
  val times : uint64 −> uint64 −> uint64;
  val divide : uint64 −> uint64 −> uint64;
  val modulus : uint64 −> uint64 −> uint64;
  val negate : uint64 −> uint64;
  val less-eq : uint64 −> uint64 −> bool;
  val less : uint64 −> uint64 −> bool;
  val notb : uint64 −> uint64;
  val andb : uint64 −> uint64 −> uint64;
  val orb : uint64 −> uint64 −> uint64;
  val xorb : uint64 −> uint64 −> uint64;
  val shiftl : uint64 −> IntInf.int −> uint64;
  val shiftr : uint64 −> IntInf.int −> uint64;
  val shiftr-signed : uint64 −> IntInf.int −> uint64;
  val set-bit : uint64 −> IntInf.int −> bool −> uint64;
  val test-bit : uint64 −> IntInf.int −> bool;
end = struct

type uint64 = Word64.word;
```

```
val zero = (0wx0 : uint64);

val one = (0wx1 : uint64);

fun fromInt x = Word64.fromLargeInt (IntInf.toLarge x);

fun toInt x = IntInf.fromLarge (Word64.toLargeInt x);

fun toFixedInt x = Word64.toInt x;

fun fromLarge x = Word64.fromLarge x;

fun fromFixedInt x = Word64.fromInt x;

fun toLarge x = Word64.toLarge x;

fun toWord32 x = Word32.fromLarge x

fun plus x y = Word64.+(x, y);

fun minus x y = Word64.-(x, y);

fun negate x = Word64.~(x);

fun times x y = Word64.*(x, y);

fun divide x y = Word64.div(x, y);

fun modulus x y = Word64.mod(x, y);

fun less-eq x y = Word64.<=(x, y);

fun less x y = Word64.<(x, y);

fun set-bit x n b =
  let val mask = Word64.<< (0wx1, Word.fromLargeInt (IntInf.toLarge n))
  in if b then Word64.orb (x, mask)
     else Word64.andb (x, Word64.notb mask)
  end

fun shiftl x n =
  Word64.<< (x, Word.fromLargeInt (IntInf.toLarge n))

fun shiftr x n =
  Word64.>> (x, Word.fromLargeInt (IntInf.toLarge n))

fun shiftr-signed x n =
  Word64.~>> (x, Word.fromLargeInt (IntInf.toLarge n))

fun test-bit x n =
  Word64.andb (x, Word64.<< (0wx1, Word.fromLargeInt (IntInf.toLarge n))) <> Word64.fromInt 0

val notb = Word64.notb
```

fun andb x y = *Word64.andb*(*x*, *y*);

fun orb x y = *Word64.orb*(*x*, *y*);

fun xorb x y = *Word64.xorb*(*x*, *y*);

end (∗*struct Uint64*∗)
›

code-printing constant *arl-get-u′* ⇀ (*SML*) (*fn/* ()*/* =>*/* *Array.sub/* ((*fn/* (*a*,*b*)*/* =>*/* *a*) ((-)),*/* *Word64.toInt* (*Uint64.toLarge* ((-)))))

definition *uint32-of-uint64′* **where**
[*symmetric*, *code*]: *uint32-of-uint64′* = *uint32-of-uint64*
code-printing constant *uint32-of-uint64′* ⇀ (*SML*) *Uint64.toWord32* ((-))
thm *hashcode-literal-def*[*unfolded hashcode-list-def*]

definition *string-nth* **where**
‹*string-nth s x* = *literal.explode s* ! *x*›

definition *string-nth′* **where**
‹*string-nth′ s x* = *literal.explode s* ! *nat x*›

lemma [*code*]: ‹*string-nth s x* = *string-nth′ s* (*int x*)›
unfolding *string-nth-def string-nth′-def*
by *auto*

definition *string-size* :: ‹*String.literal*⇒*nat*› **where**
‹*string-size s* = *size s*›

definition *string-size′* **where**
[*symmetric*,*code*]: ‹*string-size′* = *string-size*›

lemma [*code*]: ‹*size* = *string-size*›
unfolding *string-size-def* **..**

code-printing constant *string-nth′* ⇀ (*SML*) (*String.sub/* ((-),*/* *IntInf.toInt* ((*integer′-of′-int* ((-))))))
code-printing constant *string-size′* ⇀ (*SML*) *nat′-of′-integer* ((*IntInf.fromInt* ((*String.size* ((-))))))

function *hashcode-eff* **where**
[*simp del*]: ‹*hashcode-eff s h i* = (*if i* ≥ *size s then h else hashcode-eff s* (*h* ∗ *33* + *hashcode* (*s* ! *i*)) (*i*+*1*))›
by *auto*
termination
by (*relation* ‹*measure* (λ(*s*,*h*,*i*). *size s* −*i*)›)
auto

definition *hashcode-eff′* **where**
‹*hashcode-eff′ s h i* = *hashcode-eff* (*String.explode s*) *h i*›

lemma *hashcode-eff′-code*[*code*]:
‹*hashcode-eff′ s h i* = (*if i* ≥ *size s then h else hashcode-eff′ s* (*h* ∗ *33* + *hashcode* (*string-nth s i*)) (*i*+*1*))›
unfolding *hashcode-eff′-def string-nth-def hashcode-eff.simps*[*symmetric*] *size-literal.rep-eq*
..

lemma [*simp*]: ‹*length s* ≤ *i* ⟹ *hashcode-eff s h i* = *h*›
  by (*subst hashcode-eff.simps*)
   auto
lemma [*simp*]: ‹*hashcode-eff* (*a* # *s*) *h* (*Suc i*) = *hashcode-eff* (*s*) *h* (*i*)›
  apply (*induction s h i rule*: *hashcode-eff.induct*)
  subgoal
    apply (*subst* (*2*) *hashcode-eff.simps*)
    apply (*subst* (*1*) *hashcode-eff.simps*)
    apply *auto*
    done
  done

lemma *hashcode-eff-def*[*unfolded hashcode-eff'-def*[*symmetric*], *code*]:
  ‹*hashcode s* = *hashcode-eff* (*String.explode s*)*5381 0*› **for** *s*::*String.literal*
proof −
  have *H*: ‹*length* (*literal.explode s*) = *size s*›
    by (*simp add*: *size-literal.rep-eq*)
  have [*simp*]: ‹*foldl* (λ*h xa. h* ∗ *33* + *hashcode* ((*xs* @ [*x*]) ! *xa*)) *5381* [*0*..<*length xs*] =
    foldl (λ*h x. h* ∗ *33* + *hashcode* (*xs* ! *x*)) *5381* [*0*..<*length xs*]› **for** *xs x*
    by (*rule foldl-cong*) *auto*
  have ‹*foldl* (λ*h x. h* ∗ *33* + *hashcode x*) *5381* (*s*) =
    foldl (λ*h x. h* ∗ *33* + *hashcode* (*s* ! *x*)) *5381*
     [*0*..<*length* (*s*)]› **for** *s*
    by (*induction s rule*: *rev-induct*) *auto*
  then have *0*: ‹*hashcode s* = *foldl* (λ*h x. h* ∗ *33* + *hashcode* (*string-nth s x*)) *5381* [*0*..<*size s*]›
    unfolding *string-nth-def*
    unfolding *hashcode-literal-def*[*unfolded hashcode-list-def*] *size-literal.rep-eq*
    by *blast*

  have *upt*: ‹¬ *Suc* (*length s*) ≤ *i* ⟹ [*i*..<*Suc* (*length s*)] = *i* # [*Suc i*..<*Suc* (*length s*)]› **for** *i s*
    by (*meson leI upt-rec*)

  have [*simp*]: ‹*foldl* (λ*h x. h* ∗ *33* + *hashcode* ((*a* # *s*) ! *x*)) *h* [*Suc i*..<*Suc* (*length s*)] =
    foldl (λ*h x. h* ∗ *33* + *hashcode* (*s* ! *x*)) *h* [*i*..<(*length s*)]› **for** *a s i h*
  proof −
    have ‹*foldl* (λ*h x. h* ∗ *33* + *hashcode* ((*a* # *s*) ! *x*)) *h* [*Suc i*..<*Suc* (*length s*)] =
      foldl (λ*h x. h* ∗ *33* + *hashcode* ((*a* # *s*) ! *x*)) *h* (*map Suc* [*i*..<(*length s*)])›
      using *map-Suc-upt* **by** *presburger*
    also have ‹. . . = *foldl* (λ*aa x. aa* ∗ *33* + *hashcode* (*s* ! *x*)) *h* [*i*..<*length s*]›
      unfolding *foldl-map* **by** (*rule foldl-cong*) *auto*
    finally show *?thesis* **.**
  qed

  have *H'*: ‹*foldl* (λ*h x. h* ∗ *33* + *hashcode* (*s* ! *x*)) *h* [*i*..<*length s*] =
    hashcode-eff s h i› **for** *i h s*
    unfolding *string-nth-def H*[*symmetric*]
    supply [*simp del*] = *upt.simps*
    apply (*induction* ‹*s*› *arbitrary*: *h*)
    subgoal
      by (*subst hashcode-eff.simps*)
       auto
    subgoal
      by (*subst hashcode-eff.simps*)

```
    (auto simp: upt)
  done
 show ?thesis
  unfolding 0 H[symmetric] string-nth-def H′
  ..
qed

export-code hashcode :: String.literal ⇒ -
 in SML-imp module-name PAC-Checker

code-printing code-module array-blit ⇀ (SML)
  ‹
 fun array-blit src si dst di len = (
    src=dst andalso raise Fail (array-blit: Same arrays);
    ArraySlice.copy {
     di = IntInf.toInt di,
     src = ArraySlice.slice (src,IntInf.toInt si,SOME (IntInf.toInt len)),
     dst = dst})

  fun array-nth-oo v a i () = if IntInf.toInt i >= Array.length a then v
     else Array.sub(a,IntInf.toInt i) handle Overflow => v
  fun array-upd-oo f i x a () =
    if IntInf.toInt i >= Array.length a then f ()
    else
     (Array.update(a,IntInf.toInt i,x); a) handle Overflow => f ()

›
```

This is a hack for performance. There is no need to recheck that that a char is valid when working on chars coming from strings... It is not that important in most cases, but in our case the preformance difference is really large.

```
definition unsafe-asciis-of-literal :: ‹-› where
 ‹unsafe-asciis-of-literal xs = String.asciis-of-literal xs›

definition unsafe-asciis-of-literal′ :: ‹-› where
 [simp, symmetric, code]: ‹unsafe-asciis-of-literal′ = unsafe-asciis-of-literal›

code-printing
 constant unsafe-asciis-of-literal′ ⇀
  (SML) !(List.map (fn c => let val k = Char.ord c in IntInf.fromInt k end) /o String.explode)
```

Now comes the big and ugly and unsafe hack.

Basically, we try to avoid the conversion to IntInf when calculating the hash. The performance gain is roughly 40%, which is a LOT and definitively something we need to do. We are aware that the SML semantic encourages compilers to optimise conversions, but this does not happen here, corroborating our early observation on the verified SAT solver IsaSAT.x

```
definition raw-explode where
 [simp]: ‹raw-explode = String.explode›
code-printing
 constant raw-explode ⇀
  (SML) String.explode

lemmas [code] =
 hashcode-literal-def[unfolded String.explode-code
```

unsafe-asciis-of-literal-def[*symmetric*]]

definition *uint32-of-char* **where**
[*symmetric*, *code-unfold*]: ‹*uint32-of-char x* = *uint32-of-int* (*int-of-char x*)›

code-printing
constant *uint32-of-char* ⇀
(*SML*) !(*Word32.fromInt* /*o* (*Char.ord*))

lemma [*code*]: ‹*hashcode s* = *hashcode-literal′ s*›
unfolding *hashcode-literal-def hashcode-list-def*
apply (*auto simp*: *unsafe-asciis-of-literal-def hashcode-list-def*
String.asciis-of-literal-def hashcode-literal-def hashcode-literal′-def)
done

export-code
full-checker-l-s2-impl int-of-integer Del CL nat-of-integer String.implode remap-polys-l2-with-err-s-impl
PAC-update-impl PAC-empty-impl the-error is-cfailed is-cfound
fully-normalize-poly-impl empty-shared-vars-int-impl
PAC-checker-l-s-impl PAC-checker-l-step-s-impl version
in *SML-imp* **module-name** *PAC-Checker*
file-prefix *checker*

compile-generated-files -
external-files
‹*code*/*parser.sml*›
‹*code*/*pasteque.sml*›
‹*code*/*pasteque.mlb*›
where ‹*fn dir* =>
let

val exec = *Generated-Files.execute* (*Path.append dir* (*Path.basic code*));
val - = *exec* ‹*Copy files*›
(*cp checker.ML* ^((*File.bash-path* ***path***‹$*ISAFOL*›) ^/*PAC-Checker2*/*code*/*checker.ML*));
val - =
exec ‹*Compilation*›
(*File.bash-path* ***path***‹$*ISABELLE-MLTON*› ^ ^
−*const* ′*MLton.safe false*′ −*verbose 1* −*default*−*type int64* −*output pasteque* ^
−*codegen native* −*inline 700* −*cc*−*opt* −*O3 pasteque.mlb*);
in () *end*›

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.