A Set Reconciliation Algorithm

Paul Hofmeier and Emin Karayel

January 25, 2026

Abstract

This entry formally verifies the set reconciliation algorithm with nearly optimal communication complexity, due to Y. Minsky *et al.* [1]. The algorithm allows two communication partners, who have a similar pair of sets to reconcile them while using messages of nearly optimal size, proportional to a bound on the maximum symmetric difference between the sets.

The formalization also introduces an optimization, which reduces the communication complexity even further compared to the original publication.

Contents

1 Preliminary Results

```
theory Poly-Lemmas
  imports
    HOL−Computational-Algebra.Polynomial
    Polynomial-Interpolation.Missing-Polynomial
begin

lemma card-sub-int-diff-finite:
  assumes finite A finite B
  shows int (card A) − card B = int (card (A−B)) − card (B−A)
  using assms card-add-diff-finite by fastforce

lemma card-sub-int-diff-finite-real:
  assumes finite A finite B
  shows real (card A) − card B = real (card (A−B)) − card (B−A)
  using assms card-add-diff-finite by fastforce
```

1.1 Characteristic Polynomial

The characteristic polynomial associated to a set:

```
definition set-to-poly :: 'a::finite-field set ⇒ 'a poly where
  set-to-poly A ≡ ∏ a ∈ A. [:−a,1:]

lemma set-to-poly-correct: {x. poly (set-to-poly A) x = 0} = A
proof (induct A rule: infinite-finite-induct)
  case (infinite A)
  then show ?case by simp
next
  case empty
  then show ?case unfolding set-to-poly-def by simp
next
  case (insert x F)
  have set-to-poly (insert x F) = set-to-poly F ∗ [:−x,1:]
    unfolding set-to-poly-def by (simp add: insert.hyps(2))
  also have {xa. poly (set-to-poly F ∗ [:−x,1:]) xa = 0} =
      {xa. poly (set-to-poly F) xa = 0} ∪ {xa. poly ([:−x,1:]) xa = 0}
    by auto
  moreover have 2: {xa. poly (set-to-poly F) xa = 0} = F
    by (simp add: insert.hyps(3))
  moreover have 3: {xa. poly ([:−x,1:]) xa = 0} = {x}
    by auto
  ultimately have {xa. poly (set-to-poly (insert x F)) xa = 0} = F ∪ {x}
     by simp
  then show ?case by simp
qed

lemma in-set-to-poly: poly (set-to-poly A) x = 0 ⟷ x ∈ A
  using set-to-poly-correct
```

by *auto*

lemma *set-to-poly-not0*[*simp*]: *set-to-poly A* ≠ *0*
unfolding *set-to-poly-def* **by** *auto*

lemma *set-to-poly-empty*[*simp*]: *set-to-poly* {} = *1*
unfolding *set-to-poly-def* **by** *simp*

lemma *set-to-poly-inj*: *inj set-to-poly*
by (*metis injI set-to-poly-correct*)

lemma *rsquarefree-set-to-poly*: *rsquarefree* (*set-to-poly A*)
proof (*induct A rule*: *infinite-finite-induct*)
case (*infinite A*)
then show *?case* **by** *simp*
next
case *empty*
then show *?case*
by (*simp add*: *rsquarefree-def set-to-poly-def*)
next
case (*insert x F*)
then have *1*: *set-to-poly* (*insert x F*) = *set-to-poly F* ∗ [:−*x*,*1*:]
by (*simp add*: *set-to-poly-def*)

have *rsquarefree* [:−*x*,*1*:]
using *rsquarefree-single-root* **by** *simp*
also have *poly* (*set-to-poly F*) *x* ≠ *0*
using *insert* **by** (*simp add*: *in-set-to-poly*)
moreover have *poly* ([:−*x*,*1*:]) *x* = *0*
using *insert* **by** *simp*
ultimately have *rsquarefree*(*set-to-poly F* ∗ [:−*x*,*1*:])
using *insert*(*3*) *rsquarefree-mul* **by** *fastforce*

then show *?case* **using** *1*
by *simp*
qed

lemma *set-to-poly-insert*:
assumes*x* ∉ *A*
shows *set-to-poly* (*insert x A*) = *set-to-poly A* ∗ [:−*x*,*1*:]
using *assms set-to-poly-def* **by** (*simp add*: *set-to-poly-def*)

lemma *set-to-poly-mult*: *set-to-poly X* ∗ *set-to-poly Y* = *set-to-poly* (*X* ∪ *Y*) ∗ *set-to-poly* (*X* ∩ *Y*)
by (*simp add*: *prod.union-inter set-to-poly-def*)

lemma *set-to-poly-mult-distinct*:
assumes *X* ∩ *Y* = {}
shows *set-to-poly X* ∗ *set-to-poly Y* = *set-to-poly* (*X* ∪ *Y*)

by (*simp add*: *set-to-poly-mult assms*)

lemma *set-to-poly-degree*:
degree (*set-to-poly A*) = *card A*
proof (*induct A rule*: *infinite-finite-induct*)
case (*infinite A*)
then show *?case* **by** *auto*
next
case *empty*
then show *?case* **by** *auto*
next
case (*insert x F*)
have [:−*x*, *1*:] ≠ *0* **and** *set-to-poly F* ≠ *0*
using *set-to-poly-not0* **by** *auto*
then have *degree* (*set-to-poly F* ∗ [:−*x*, *1*:]) = *degree* (*set-to-poly F*) + *degree* [:−*x*, *1*:]
using *degree-mult-eq* **by** *blast*
also have *set-to-poly* (*insert x F*) = *set-to-poly F* ∗ [:−*x*, *1*:]
using *insert set-to-poly-insert* **by** *simp*
ultimately show *?case* **using** *insert*
by *simp*
qed

lemma *set-to-poly-order*:
order x (*set-to-poly A*) = (*if x* ∈ *A then 1 else 0*)
by (*simp add*: *in-set-to-poly order-0I rsquarefree-root-order rsquarefree-set-to-poly*)

lemma *set-to-poly-lead-coeff*: *lead-coeff* (*set-to-poly A*) = *1*
proof (*induct A rule*: *infinite-finite-induct*)
case (*infinite A*)
then show *?case* **by** *auto*
next
case *empty*
then show *?case* **by** *auto*
next
case (*insert x A*)
then have *ins*: *set-to-poly* (*insert x A*) = *set-to-poly A* ∗ [:−*x*,*1*:]
unfolding *set-to-poly-def* **by** *simp*
then show *?case*
unfolding *ins lead-coeff-mult* **using** *insert* **by** *simp*
qed

lemma *degree-sub-lead-coeff*:
assumes *degree p* > *0*
shows *degree* (*p* − *monom* (*lead-coeff p*) (*degree p*)) < *degree p*
using *assms* **by** (*simp add*: *coeff-eq-0 degree-lessI*)

lemma *remove-lead-from-monic*:
fixes *p q* :: ′*a* :: *field poly*

assumes *monic p*
assumes *degree p* $>$ *0*
shows *degree* (*p* $-$ *monom 1* (*degree p*)) $<$ *degree p*
using *degree-sub-lead-coeff*[*OF assms*(*2*)] *assms*(*1*) **by** *simp*

lemma *poly-eqI-degree-monic*:
fixes *p q* :: *'a* :: *field poly*
assumes *degree p* $=$ *degree q*
assumes *degree p* $\leq$ *card A*
assumes *monic p monic q*
assumes $\bigwedge x.\ x \in A \Longrightarrow$ *poly p x* $=$ *poly q x*
shows *p* $=$ *q*
proof (*cases degree p* $>$ *0*)
case *True*
have *degree* (*p* $-$ *monom 1* (*degree p*)) $<$ *card A*
using *remove-lead-from-monic*[*OF assms*(*3*)] *True assms*(*2*) **by** *simp*
moreover have *degree* (*q* $-$ *monom 1* (*degree q*)) $<$ *card A*
using *remove-lead-from-monic*[*OF assms*(*4*)] *True assms*(*1*,*2*) **by** *simp*
ultimately have *p* $-$ *monom 1* (*degree p*) $=$ *q* $-$ *monom 1* (*degree q*)
using *assms*(*1*,*5*) **by** (*intro poly-eqI-degree*[*of A*]) *auto*
thus *?thesis* **using** *assms*(*1*) **by** *simp*
next
case *False*
hence *degree p* $=$ *0 degree q* $=$ *0* **using** *assms*(*1*) **by** *auto*
thus *p* $=$ *q* **using** *assms*(*3*,*4*) *monic-degree-0* **by** *blast*
qed

end

2 Rational Function Interpolation

theory *Rational-Function-Interpolation*
imports
Poly-Lemmas
Gauss-Jordan.System-Of-Equations
Polynomial-Interpolation.Missing-Polynomial
begin

2.1 Definitions

General condition for rational functions interpolation

definition *interpolated-rational-function* **where**
interpolated-rational-function p_A p_B *E* f_A f_B d_A d_B $\equiv$
($\forall$ *e* $\in$ *E*. f_A *e* $*$ *poly* p_B *e* $=$ f_B *e* $*$ *poly* p_A *e*) $\wedge$
degree p_A $\leq$ (d_A::*real*) $\wedge$ *degree* p_B $\leq$ (d_B::*real*) $\wedge$
$p_A \neq 0 \wedge p_B \neq 0$

Interpolation condition with given exact degrees

definition *monic-interpolated-rational-function* **where**

monic-interpolated-rational-function p_A p_B E f_A f_B d_A d_B $\equiv$
  ($\forall$ $e \in E$. f_A e $*$ *poly* p_B e $=$ f_B e $*$ *poly* p_A e) $\wedge$
  degree $p_A = \lfloor d_A$::*real*$\rfloor$ $\wedge$ *degree* $p_B = \lfloor d_B$::*real*$\rfloor$ $\wedge$
  monic p_A $\wedge$ *monic* p_B

lemma *monic0*: $\neg$ *monic* (*0*::'*a*::*zero-neq-one poly*)
  by *simp*

lemma *monic-interpolated-rational-function-interpolated-rational-function*:
  monic-interpolated-rational-function p_A p_B E f_A f_B d_A d_B
    $\Longrightarrow$ *interpolated-rational-function* p_A p_B E f_A f_B d_A d_B $\vee$ $\neg$($p_A \neq 0 \wedge p_B \neq 0$)
  unfolding *monic-interpolated-rational-function-def interpolated-rational-function-def*
  by *linarith*

definition *rfi-coefficient-matrix* :: '*a*::*field list* $\Rightarrow$ ('*a* $\Rightarrow$ '*a*) $\Rightarrow$ *nat* $\Rightarrow$ *nat*
  $\Rightarrow$ *nat* $\Rightarrow$ *nat* $\Rightarrow$ '*a* **where**
  rfi-coefficient-matrix E f d_A d_B i j = (
    if $j < d_A$ *then*
      (E ! i) ^ j
    else if $j < d_A + d_B$ *then*
      $-$ f (E ! i) $*$ (E ! i) ^($j - d_A$)
    else 0
)

definition *rfi-constant-vector* :: '*a*::*field list* $\Rightarrow$ ('*a* $\Rightarrow$ '*a*) $\Rightarrow$ *nat* $\Rightarrow$ *nat* $\Rightarrow$ (*nat* $\Rightarrow$ '*a*) **where**
  rfi-constant-vector E f d_A d_B = (λ i. f (E ! i) $*$ (E ! i) ^ d_B $-$ (E ! i) ^ d_A)

definition *rational-function-interpolation* :: '*a*::*field list* $\Rightarrow$ ('*a* $\Rightarrow$ '*a*) $\Rightarrow$ *nat* $\Rightarrow$ *nat*
  $\Rightarrow$ '*m*::*mod-type itself* $\Rightarrow$ ('*a*,'*m*) *vec* **where**
  rational-function-interpolation E f d_A d_B m =
    (*let solved* = *solve*
      (χ (i::'*m*) (j::'*m*). *rfi-coefficient-matrix* E f d_A d_B (*to-nat* i) (*to-nat* j))
      (χ (i::'*m*). *rfi-constant-vector* E f d_A d_B (*to-nat* i))
    in fst (*the solved*))

definition *solution-to-poly* :: ('*a*::*finite-field*, '*n*::*mod-type*) *vec* $\Rightarrow$
  nat $\Rightarrow$ *nat* $\Rightarrow$ '*a poly* $\times$ '*a poly* **where**
  solution-to-poly S d_A d_B = (*let*
    p = *Abs-poly* (λi. *if* $i < d_A$ *then* S \$ (*from-nat* i) *else 0*) + *monom 1* d_A;
    q = *Abs-poly* (λi. *if* $i < d_B$ *then* S \$ (*from-nat* ($i + d_A$)) *else 0*) + *monom 1* d_B *in*
    (p, q))

definition *interpolate-rat-fun* **where**
  interpolate-rat-fun E f d_A d_B m =
  solution-to-poly (*rational-function-interpolation* E f d_A d_B m) d_A d_B

2.2 Preliminary Results

lemma *consecutive-sum-combine*:
  assumes $m \geq n$
  shows $(\sum i = 0..n.\ f\ i) + (\sum i = Suc\ n\ ..m.\ f\ i) = (\sum i = 0..m.\ f\ i)$
proof −
  from *assms* **have** $\{0..n\} \cup \{Suc\ n..m\} = \{0..m\}$
    by *auto*
  moreover have $sum\ f\ (\{0..n\} \cup \{Suc\ n..m\}) =$
      $sum\ f\ (\{0..n\} - \{Suc\ n..m\}) + sum\ f\ (\{Suc\ n..m\} - \{0..n\}) + sum\ f\ (\{0..n\}$
$\cap \{Suc\ n..m\})$
    using *sum-Un2 finite-atLeastAtMost* **by** *fast*
  ultimately show *?thesis*
    by (*simp add*: *Diff-triv*)
qed

lemma *poly-altdef-Abs-poly-le*:
  fixes $x :: 'a::\{comm\text{-}semiring\text{-}0,\ semiring\text{-}1\}$
  shows $poly\ (Abs\text{-}poly\ (\lambda i.\ if\ i \leq n\ then\ f\ i\ else\ 0))\ x = (\sum i = 0..n.\ f\ i * x\ \widehat{}\ i)$
proof −
  let $?if_A 0 = (\lambda i.\ if\ i \leq n\ then\ f\ i\ else\ 0)$
  let $?p = Abs\text{-}poly\ ?if_A 0$

  have *co*: $coeff\ ?p = ?if_A 0$
    using *coeff-Abs-poly-If-le* **by** *blast*

  then have $\forall\ i{>}n.\ coeff\ ?p\ i = 0$
    by *auto*
  then have *de*: $degree\ ?p \leq n$
    using *degree-le* **by** *blast*

  have $\forall\ i{>}degree\ ?p.\ ?if_A 0\ i = 0$
    using *co coeff-eq-0* **by** *fastforce*
  then have $\forall\ i{>}degree\ ?p.\ ?if_A 0\ i * x\ \widehat{}\ i = 0$
    by *simp*
  then have $\forall\ \ i \in \{Suc\ (degree\ ?p)..n\}.\ (?if_A 0\ i * x\ \widehat{}\ i) = 0$
    using *less-eq-Suc-le* **by** *fastforce*
  then have *db*: $(\sum i = Suc\ (degree\ ?p)..n.\ ?if_A 0\ i * x\ \widehat{}\ i) = 0$
     by *simp*

  have $poly\ ?p\ x = (\sum i{\leq}degree\ ?p.\ coeff\ ?p\ i * x\ \widehat{}\ i)$
    using *poly-altdef* **by** *auto*
  also have $\ldots = (\sum i{\leq}degree\ ?p.\ ?if_A 0\ i * x\ \widehat{}\ i)$
    using *co* **by** *simp*
  also have $\ldots = (\sum i = 0..degree\ ?p.\ ?if_A 0\ i * x\ \widehat{}\ i)$
    using *atMost-atLeast0* **by** *simp*
  also have $\ldots = (\sum i = 0..degree\ ?p.\ ?if_A 0\ i * x\ \widehat{}\ i) +$
     $(\sum i = Suc\ (degree\ ?p)..n.\ ?if_A 0\ i * x\ \widehat{}\ i)$
    using *db* **by** *simp*
  also have $\ldots = (\sum i = 0..n.\ ?if_A 0\ i * x\ \widehat{}\ i)$

using *consecutive-sum-combine de* **by** *blast*
finally show *?thesis*
by *simp*
qed

lemma *poly-altdef-Abs-poly-l*:
fixes $x :: 'a::\{comm\text{-}semiring\text{-}0, semiring\text{-}1\}$
shows *poly* (*Abs-poly* ($\lambda i.$ *if* $i < n$ *then f i else 0*)) $x = (\sum i{<}n.\ f\ i * x\ \hat{}\ i)$
proof (*cases n*)
case *0*
have *p0*: *Abs-poly* ($\lambda i.$ *0*) = *0*
using *zero-poly-def* **by** *fastforce*
show *?thesis*
using *0* **by** (*simp add*: *p0*)
next
case (*Suc m*)
have *poly* (*Abs-poly* ($\lambda i.$ *if* $i \le m$ *then f i else 0*)) $x = (\sum i = 0..m.\ f\ i * x\ \hat{}\ i)$
using *poly-altdef-Abs-poly-le* **by** *blast*
moreover have *poly* (*Abs-poly* ($\lambda i.$ *if* $i \le m$ *then f i else 0*)) x = *poly* (*Abs-poly* ($\lambda i.$ *if* $i < n$ *then f i else 0*)) x
using *Suc* **using** *less-Suc-eq-le* **by** *auto*
moreover have $(\sum i = 0..m.\ f\ i * x\ \hat{}\ i) = (\sum i{<}n.\ f\ i * x\ \hat{}\ i)$
using *Suc atLeast0AtMost lessThan-Suc-atMost* **by** *presburger*
ultimately show *?thesis* **by** *argo*
qed

lemma *degree-Abs-poly-If-l*:
assumes $n \neq 0$
shows *degree* (*Abs-poly* ($\lambda i.$ *if* $i < n$ *then f i else 0*)) $< n$
proof −
have *coeff* (*Abs-poly* ($\lambda i.$ *if* $i < n$ *then f i else 0*)) $x = 0$ **if** $x \ge n$ **for** x
using *coeff-Abs-poly* [*of n* ($\lambda i.$ *if* $i < n$ *then f i else 0*)] **using** *that* **by** *simp*
then show *?thesis*
using *assms degree-lessI* **by** *blast*
qed

lemma *nth-less-length-in-set-eq*:
shows ($\forall\ i < length\ E.\ f\ (E\ !\ i) = g\ (E\ !\ i)) \longleftrightarrow (\forall\ e \in set\ E.\ f\ e = g\ e)$
proof *standard*
show $\forall i < length\ E.\ f\ (E\ !\ i) = g\ (E\ !\ i) \Longrightarrow \forall e{\in}set\ E.\ f\ e = g\ e$
using *in-set-conv-nth* **by** *metis*
next
show $\forall e{\in}set\ E.\ f\ e = g\ e \Longrightarrow \forall i{<}length\ E.\ f\ (E\ !\ i) = g\ (E\ !\ i)$
by *simp*
qed

lemma *nat-leq-real-floor*: *real* (*i*::*nat*) $\le$ (*d*::*real*) $\longleftrightarrow$ *real* $i \le \lfloor d \rfloor$ (**is** *?l* = *?r*)
proof
assume *?l*

then show *?r*
  using *floor-mono* **by** *fastforce*
next
  assume *?r*
  then show *?l*
    by *linarith*
qed

lemma *mod-type-less-function-eq*:
  fixes $i :: 'a::mod\text{-}type$
  assumes $\forall\ i < CARD('a)\ .\ f\ i = g\ i$
  shows $f\ (to\text{-}nat\ i) = g\ (to\text{-}nat\ i)$
  using *assms* **by** (*simp add*: *to-nat-less-card*)

2.3 On *solution-to-poly*

lemma *fst-solution-to-poly-nz*:
  $fst\ (solution\text{-}to\text{-}poly\ S\ d_A\ d_B) \neq 0$
proof
  assume $fst\ (solution\text{-}to\text{-}poly\ S\ d_A\ d_B) = 0$
  hence $coeff\ (Abs\text{-}poly\ (\lambda i.\ if\ i < d_A\ then\ S\ \$\ (from\text{-}nat\ i)\ else\ 0) + monom\ 1\ d_A)\ d_A = 0$
    unfolding *solution-to-poly-def* **by** *simp*
  hence $coeff\ (Abs\text{-}poly\ (\lambda i.\ if\ i < d_A\ then\ S\ \$\ (from\text{-}nat\ i)\ else\ 0))\ d_A + 1 = 0$ **by** *simp*
  thus *False* **by** (*subst* (*asm*) *coeff-Abs-poly*[**where** $n{=}d_A$]) *auto*
qed

lemma *snd-solution-to-poly-nz*:
  $snd\ (solution\text{-}to\text{-}poly\ S\ d_A\ d_B) \neq 0$
proof
  assume $snd\ (solution\text{-}to\text{-}poly\ S\ d_A\ d_B) = 0$
  hence $coeff\ (Abs\text{-}poly\ (\lambda i.\ if\ i < d_B\ then\ S\ \$\ (from\text{-}nat\ (i{+}d_A))\ else\ 0) + monom\ 1\ d_B)\ d_B = 0$
    unfolding *solution-to-poly-def* **by** *simp*
  hence $coeff\ (Abs\text{-}poly\ (\lambda i.\ if\ i < d_B\ then\ S\ \$\ (from\text{-}nat\ (i{+}d_A))\ else\ 0))\ d_B + 1 = 0$ **by** *simp*
  thus *False* **by** (*subst* (*asm*) *coeff-Abs-poly*[**where** $n{=}d_B$]) *auto*
qed

lemma *degree-Abs0p1*: $degree\ (Abs\text{-}poly\ (\lambda i.\ 0) + 1) = 0$
    by (*metis add-0 degree-1 zero-poly-def*)

lemma *degree-solution-to-poly-fst*:
  $degree\ (fst\ (solution\text{-}to\text{-}poly\ S\ d_A\ d_B)) = d_A$
proof ($cases\ d_A$)
  case *0*
  then show *?thesis* **unfolding** *solution-to-poly-def*
    using *degree-Abs0p1* **by** (*simp add*: *one-pCons*)

next
  case (*Suc nat*)
  then have *degree* (*Abs-poly* (λi. *if* $i < d_A$ *then S* \$ *from-nat i else 0*)) $< d_A$
    using *degree-Abs-poly-If-l* **by** *fast*
  moreover have ... = *degree* (*monom* (*1*::'*a*) d_A)
    by (*simp add*: *degree-monom-eq*)
  ultimately show *?thesis*
    unfolding *solution-to-poly-def*
    by (*simp add*: *degree-add-eq-right*)
qed

lemma *degree-solution-to-poly-snd*:
  degree (*snd* (*solution-to-poly S* d_A d_B)) = d_B
proof (*cases* d_B)
  case *0*
  then show *?thesis* **unfolding** *solution-to-poly-def*
    using *degree-Abs0p1* **by** (*simp add*: *one-pCons*)
next
  case (*Suc nat*)
  then have *degree* (*Abs-poly* (λi. *if* $i < d_B$ *then S* \$ *from-nat* ($i + d_A$) *else 0*))
$< d_B$
    using *degree-Abs-poly-If-l* **by** *fast*
  moreover have ... = *degree* (*monom* (*1*::'*a*) d_B)
    by (*simp add*: *degree-monom-eq*)
  ultimately show *?thesis*
    unfolding *solution-to-poly-def*
    by (*simp add*: *degree-add-eq-right*)
qed

lemma *monic-solution-to-poly-snd*:
  monic (*snd* (*solution-to-poly S* d_A d_B))
proof (*cases* d_B)
  case *0*
  then show *?thesis* **unfolding** *solution-to-poly-def*
    by (*simp add*: *coeff-Abs-poly degree-Abs0p1*)
next
  case (*Suc x*)
  have *1*: *coeff* (*Abs-poly* (λi. *if* $i <$ *Suc x then S* \$ *from-nat* ($i + d_A$) *else 0*))
(*Suc x*) = *0*
    by (*simp add*: *coeff-eq-0 degree-Abs-poly-If-l*)
  have *degree* (*Abs-poly* (λi. *if* $i < d_B$ *then S* \$ *from-nat* ($i + d_A$) *else 0*) +
monom 1 d_B) = d_B
    using *degree-solution-to-poly-snd* **unfolding** *solution-to-poly-def* **by** *auto*
  then show *?thesis*
      unfolding *solution-to-poly-def* **using** *1 Suc* **by** *simp*
  qed

lemma *monic-solution-to-poly-fst*:
  monic (*fst* (*solution-to-poly S* d_A d_B))

```
proof (cases d_A)
  case 0
  then show ?thesis
    unfolding solution-to-poly-def by (simp add: coeff-Abs-poly degree-Abs0p1)
next
  case (Suc x)
  have 1: coeff (Abs-poly (λi. if i < d_A then S $ (from-nat i) else 0)) (Suc x) = 0
    by (simp add: Suc coeff-eq-0 degree-Abs-poly-If-l)
  have degree (Abs-poly (λi. if i < d_A then S $ (from-nat i) else 0) + monom 1
d_A) = d_A
    using degree-solution-to-poly-fst unfolding solution-to-poly-def by auto
  then show ?thesis
      unfolding solution-to-poly-def using 1 Suc by simp
qed
```

2.4 Correctness

Needs the assumption that the system is consistent, because a solution exists.

```
lemma rational-function-interpolation-correct-poly:
  assumes
    ∀ x ∈ set E. f x = f_A x / f_B x ∀ x ∈ set E. f_B x ≠ 0
    d_A + d_B ≤ length E
    CARD('m::mod-type) = length E
    consistent (χ (i::'m) (j::'m). rfi-coefficient-matrix E f d_A d_B (to-nat i) (to-nat
j))
                (χ (i::'m). rfi-constant-vector E f d_A d_B (to-nat i))
    S = rational-function-interpolation E f d_A d_B TYPE('m)
    p_A = fst (solution-to-poly S d_A d_B)
    p_B = snd (solution-to-poly S d_A d_B)
  shows
    ∀ e ∈ set E. f_A e * poly p_B e = f_B e * poly p_A e
proof −

  let ?coeff = rfi-coefficient-matrix E f d_A d_B
  let ?const = rfi-constant-vector E f d_A d_B
  let ?coeff' = (χ (i::'m) (j::'m). ?coeff (to-nat i) (to-nat j))
  let ?const' = (χ (i::'m). ?const (to-nat i))

  have is-solution S ?coeff' ?const'
   by (simp add: assms(5,6) consistent-imp-is-solution-solve rational-function-interpolation-def)
  then have sol: ?coeff' *v S = ?const'
    by (simp add: is-solution-def)

  have const: ?const i = ?const' $ from-nat i if i < length E for i
    by (simp add: assms(4) that to-nat-from-nat-id)

  have coeff: ?coeff i j = ?coeff' $ from-nat i $ from-nat j
    if i < length E j < length E for i j
  proof −
```

```
    have to-nat (from-nat i ::'m) = i
      using that assms(4)
      by (intro to-nat-from-nat-id) simp
    moreover have to-nat (from-nat j ::'m) = j
      using that assms(4,3)
      by (intro to-nat-from-nat-id) simp
    ultimately show ?thesis
      unfolding rfi-coefficient-matrix-def
      by (simp add: Let-def)
  qed

  have x: (∑j < d_A + d_B. (?coeff i j) * S $ (from-nat j)) = ?const i
    (is ?l = ?r) if i < length E for i
  proof -
    have ?l = (∑j < length E. ?coeff i j * S $ (from-nat j))
    using assms(3) by (intro sum.mono-neutral-cong-left) (auto simp add:rfi-coefficient-matrix-def)
     also have ... = (∑j < length E. ?coeff' $ (from-nat i) $ (from-nat j) * S $
(from-nat j))
      using coeff that by auto
    also have ... = (∑j ∈ {0..< length E}. ?coeff' $ (from-nat i) $ (from-nat j)
* S $ (from-nat j))
      by (intro sum.reindex-bij-betw [symmetric] bij-betwI [where g = id]) auto
    also have ... = (∑j∈(UNIV::'m set). ?coeff' $ (from-nat i) $ j * S $ j)
    using bij-from-nat [where 'a = 'm] assms(3,4) by (intro sum.reindex-bij-betw)
simp
    also have ... = (?coeff' *v S) $ (from-nat i)
      unfolding matrix-vector-mult-def by simp
    also have ... = ?const' $ (from-nat i)
      using sol by simp
    finally show ?l = ?r using const that by simp
  qed

  let ?p-lam = λi. if i < d_A then S $ from-nat i else 0
  let ?q-lam = λi. if i < d_B then S $ from-nat (i + d_A) else 0
  let ?p' = Abs-poly ?p-lam + monom 1 d_A
  let ?q' = Abs-poly ?q-lam + monom 1 d_B
  have pq: p_A = ?p' p_B = ?q'
    using assms(7,8) unfolding solution-to-poly-def by auto

  have (∑j<d_A. S $ from-nat j * E ! i ^ j) − f (E ! i) * (∑j<d_B. S $ from-nat
(j + d_A) * E ! i ^ j)
    = f (E ! i) * E ! i ^ d_B − E ! i ^ d_A if i < length E for i
  proof -
    let ?pq-lam = (λj. (if j < d_A then E ! i ^ j else
       if j < d_A + d_B then − f (E ! i) * E ! i ^ (j − d_A) else 0) * S $ from-nat j)

    have reindex: (∑j ∈ {d_A..< d_A + d_B}. − f (E ! i) * E ! i ^ (j − d_A) * S $
from-nat j) =
      (∑j ∈ {0..< d_B}. − f (E ! i) * E ! i ^ (j) * S $ from-nat (j+d_A))
```

by (*rule sum.reindex-bij-witness* [*of* - $\lambda i.\ i + d_A\ \lambda i.\ i - d_A$]) *auto*

from x **have** $f\ (E\ !\ i) * E\ !\ i\ \hat{}\ d_B - E\ !\ i\ \hat{}\ d_A = (\sum j{<}d_A + d_B.\ ?pq\text{-}lam\ j$
$)$
unfolding *rfi-coefficient-matrix-def rfi-constant-vector-def* **using** *that* **by** *simp*
also have $\ldots = (\sum j \in \{0..< d_A + d_B\}.\ ?pq\text{-}lam\ j)$
using *atLeast0LessThan* **by** *presburger*
also have $\ldots = (\sum j \in \{0..< d_A\}.\ ?pq\text{-}lam\ j) + (\sum j \in \{d_A..< d_A + d_B\}.$
$?pq\text{-}lam\ j)$
by (*subst sum.atLeastLessThan-concat*) *auto*
also have $\ldots = (\sum j \in \{0..< d_A\}.\ E\ !\ i\ \hat{}j * S\ \$\ from\text{-}nat\ j) +$
$(\sum j \in \{d_A..< d_A + d_B\}.\ - f\ (E\ !\ i) * E\ !\ i\ \hat{}(j - d_A) * S\ \$\ from\text{-}nat\ j)$
by *auto*
also have $\ldots = (\sum j \in \{0..< d_A\}.\ E\ !\ i\ \hat{}j * S\ \$\ from\text{-}nat\ j) +$
$(\sum j \in \{0..< d_B\}.\ - f\ (E\ !\ i) * E\ !\ i\ \hat{}(j) * S\ \$\ from\text{-}nat\ (j{+}d_A))$
using *reindex* **by** *simp*
also have $\ldots = (\sum j \in \{0..< d_A\}.\ E\ !\ i\ \hat{}j * S\ \$\ from\text{-}nat\ j) +$
$- f\ (E\ !\ i) * (\sum j \in \{0..< d_B\}.\ E\ !\ i\ \hat{}(j) * S\ \$\ from\text{-}nat\ (j{+}d_A))$
by (*simp add*: *sum-distrib-left mult.commute mult.left-commute*)
finally have $f\ (E\ !\ i) * E\ !\ i\ \hat{}\ d_B - E\ !\ i\ \hat{}\ d_A = \ldots$
by *argo*

moreover have $(\sum j \in \{0..< d_A\}.\ E\ !\ i\ \hat{}j * S\ \$\ from\text{-}nat\ j) =$
$(\sum j{<}d_A.\ S\ \$\ from\text{-}nat\ j * E\ !\ i\ \hat{}j)$
by (*subst atLeast0LessThan*) (*meson mult.commute*)
moreover have $(\sum j \in \{0..< d_B\}.\ E\ !\ i\ \hat{}(j) * S\ \$\ from\text{-}nat\ (j{+}d_A)) =$
$(\sum j{<}d_B.\ S\ \$\ from\text{-}nat\ (j + d_A) * E\ !\ i\ \hat{}j)$
by (*subst atLeast0LessThan*) (*meson mult.commute*)
ultimately show *?thesis*
by *simp*
qed

then have $\forall e \in set\ E.\ (\sum j{<}d_A.\ S\ \$\ from\text{-}nat\ j * e\ \hat{}j) - f\ e * (\sum j{<}d_B.\ S\ \$$
$from\text{-}nat\ (j + d_A) * e\ \hat{}j)$
$= f\ e * e\ \hat{}\ d_B - e\ \hat{}\ d_A$
by (*subst nth-less-length-in-set-eq* [*symmetric*]) *auto*

then have $(\sum i{<}d_A.\ S\ \$\ from\text{-}nat\ i * e\ \hat{}\ i) - f\ e * (\sum i{<}d_B.\ S\ \$\ from\text{-}nat\ (i$
$+ d_A) * e\ \hat{}\ i)$
$= f\ e * e\ \hat{}\ d_B - e\ \hat{}\ d_A$ **if** $e \in set\ E$ **for** e
using *that* **by** *blast*

then have $(\sum i{<}d_A.\ S\ \$\ from\text{-}nat\ i * e\ \hat{}\ i) + e\ \hat{}\ d_A$
$= f\ e * e\ \hat{}\ d_B + f\ e * (\sum i{<}d_B.\ S\ \$\ from\text{-}nat\ (i + d_A) * e\ \hat{}\ i)$ **if** $e \in set\ E$
for e
using *that* **by** (*simp add:field-simps*)

then have $f\ e * ((\sum i{<}d_B.\ S\ \$\ from\text{-}nat\ (i + d_A) * e\ \hat{}\ i) + e\ \hat{}\ d_B) =$
$(\sum i{<}d_A.\ S\ \$\ from\text{-}nat\ i * e\ \hat{}\ i) + e\ \hat{}\ d_A$ **if** $e \in set\ E$ **for** e

using *that* **by** (*simp add*: *ring-class.ring-distribs*(*1*))

then have $f\ e * (poly\ (Abs\text{-}poly\ ?q\text{-}lam)\ e + poly\ (monom\ 1\ d_B)\ e) =$
$poly\ (Abs\text{-}poly\ ?p\text{-}lam)\ e + poly\ (monom\ 1\ d_A)\ e$ **if** $e \in set\ E$ **for** e
unfolding *poly-altdef-Abs-poly-l poly-monom* **using** *that* **by** *auto*

then have $f\ e * (poly\ (Abs\text{-}poly\ ?q\text{-}lam)\ e + poly\ (monom\ 1\ d_B)\ e) =$
$poly\ (Abs\text{-}poly\ ?p\text{-}lam)\ e + poly\ (monom\ 1\ d_A)\ e$ **if** $e \in set\ E$ **for** e
using *that* **by** *simp*
then have $f\ e * poly\ (Abs\text{-}poly\ ?q\text{-}lam + monom\ 1\ d_B)\ e =$
$poly\ (Abs\text{-}poly\ ?p\text{-}lam + monom\ 1\ d_A)\ e$ **if** $e \in set\ E$ **for** e
by (*simp add*: *that*)
then have $(f_A\ e\ /\ f_B\ e) * poly\ ?q'\ e = poly\ ?p'\ e$ **if** $e \in set\ E$ **for** e
using *that assms*(*1*) **by** *simp*
then have $f_A\ e * poly\ ?q'\ e = f_B\ e * poly\ ?p'\ e$ **if** $e \in set\ E$ **for** e
using *that* **by** (*simp add*: *assms*(*2*) *nonzero-divide-eq-eq*)
then have $\forall e \in set\ E.\ f_A\ e * poly\ (snd\ (solution\text{-}to\text{-}poly\ S\ d_A\ d_B))\ e =$
$f_B\ e * poly\ (fst\ (solution\text{-}to\text{-}poly\ S\ d_A\ d_B))\ e$
unfolding *solution-to-poly-def* **by** *auto*
then show $\forall e \in set\ E.\ f_A\ e * poly\ p_B\ e = f_B\ e * poly\ p_A\ e$
using *assms*(*8*,*7*) **by** *simp*
qed

lemma *poly-lead-coeff-extract*:
$poly\ p\ x = (\sum i{<}degree\ p.\ coeff\ p\ i * x\ \hat{}\ i) + lead\text{-}coeff\ p * x\ \hat{}\ degree\ p$
for $x :: 'a::\{comm\text{-}semiring\text{-}0, semiring\text{-}1\}$
unfolding *poly-altdef* **using** *lessThan-Suc-atMost sum.lessThan-Suc* **by** *auto*

lemma d_A-d_B-*helper*:
assumes
finite A finite B
$int\ d_A = \lfloor(real\ (length\ E) + card\ A - card\ B)/2\rfloor$
$int\ d_B = \lfloor(real\ (length\ E) + card\ B - card\ A)/2\rfloor$
$card\ (sym\text{-}diff\ A\ B) \le length\ E$
shows
$d_A + d_B \le length\ E$
$card\ (A - B) \le d_A$ $card\ (B - A) \le d_B$
$d_B - card\ (B - A) = d_A - card\ (A - B)$
proof −
have a: $real\ d_A = of\text{-}int\ \lfloor(real\ (length\ E) + card\ A - card\ B)/2\rfloor$
using *assms*(*3*) **by** *simp*
have b: $real\ d_B = of\text{-}int\ \lfloor(real\ (length\ E) + card\ B - card\ A)/2\rfloor$
using *assms*(*4*) **by** *simp*

have $real\ d_A + real\ d_B \le (real\ (length\ E){+}card\ A{-}card\ B)/2 + (real\ (length\ E){+}card\ B{-}card\ A)/2$
unfolding *a b* **by** (*intro add-mono*) *linarith+*
also have $\ldots = real\ (length\ E)$ **by** *argo*
finally have $real\ d_A + real\ d_B \le length\ E$ **by** *simp*

thus $d_A + d_B \leq$ *length E* **by** *simp*

have *real* (*card* ($A - B$)) = (*real* (*card* (*sym-diff A B*)) + *real* (*card A*) − *real* (*card B*))/*2*
unfolding *card-sym-diff-finite*[*OF assms*(*1*,*2*)] **using** *card-sub-int-diff-finite*[*OF assms*(*1*,*2*)]
by *simp*
also have ... ≤ (*real* (*length E*) + *real* (*card A*) − *real* (*card B*))/*2*
using *assms*(*5*) **by** *simp*
finally have *real* (*card* ($A{-}B$)) $\leq d_A$
unfolding *a* **using** *nat-leq-real-floor* **by** *blast*
thus *c*:*card* ($A{-}B$) $\leq d_A$ **by** *auto*

have *real* (*card* ($B - A$)) = (*real* (*card* (*sym-diff A B*)) + *real* (*card B*) − *real* (*card A*))/*2*
unfolding *card-sym-diff-finite*[*OF assms*(*1*,*2*)] **using** *card-sub-int-diff-finite*[*OF assms*(*1*,*2*)]
by *simp*
also have ... ≤ (*real* (*length E*) + *real* (*card B*) − *real* (*card A*))/*2*
using *assms*(*5*) **by** *simp*
finally have *real* (*card* ($B{-}A$)) $\leq d_B$
unfolding *b* **using** *nat-leq-real-floor* **by** *blast*
thus *d*:*card* ($B{-}A$) $\leq d_B$ **by** *auto*

have *real* d_B − *real* d_A =
of-int ⌊(*real* (*length E*) − *card B* − *card A*)/*2* + *real* (*card B*)⌋ −
of-int ⌊(*real* (*length E*) − *card A* − *card B*)/*2* + *real* (*card A*)⌋
unfolding *a b* **by** *argo*
also have ... = *real* (*card B*) − *real* (*card A*)
by (*simp add*:*algebra-simps*)
also have ... = *real* (*card* ($B - A$)) − *card* ($A - B$)
using *card-sub-int-diff-finite*[*OF assms*(*1*,*2*)] **by** *simp*
finally have *real* d_B − *real* d_A = *real* (*card* ($B - A$)) − *card* ($A - B$)
by *simp*

thus d_B − *card* ($B - A$) = d_A − *card* ($A - B$)
using *c d* **by** *simp*
qed

Insert the solution we know that must exist to show it's consistent

lemma *rational-function-interpolation-consistent*:
fixes *A B* :: *'a*::*finite-field set*
assumes
∀ $x \in$ (*set E*). *f x* = f_A *x* / f_B *x*
CARD(*'m*::*mod-type*) = *length E*
$d_A + d_B \leq$ *length E*
card ($A - B$) $\leq d_A$
card ($B - A$) $\leq d_B$
d_B − *card* ($B - A$) = d_A − *card* ($A - B$)

$\forall\ x \in set\ E.\ x \notin A$ $\forall\ x \in set\ E.\ x \notin B$
$f_A = (\lambda\ x \in set\ E.\ poly\ (set\text{-}to\text{-}poly\ A)\ x)$
$f_B = (\lambda\ x \in set\ E.\ poly\ (set\text{-}to\text{-}poly\ B)\ x)$
shows
consistent $(\chi\ (i::'m)\ (j::'m).\ rfi\text{-}coefficient\text{-}matrix\ E\ f\ d_A\ d_B\ (to\text{-}nat\ i)\ (to\text{-}nat\ j))$
$(\chi\ (i::'m).\ rfi\text{-}constant\text{-}vector\ E\ f\ d_A\ d_B\ (to\text{-}nat\ i))$
proof −
let $?coeff = rfi\text{-}coefficient\text{-}matrix\ E\ f\ d_A\ d_B$
let $?const = rfi\text{-}constant\text{-}vector\ E\ f\ d_A\ d_B$
let $?coeff' = (\chi\ (i::'m)\ (j::'m).\ ?coeff\ (to\text{-}nat\ i)\ (to\text{-}nat\ j))$
let $?const' = (\chi\ (i::'m).\ ?const\ (to\text{-}nat\ i))$

define *sp* **where** $sp = set\text{-}to\text{-}poly\ (A{-}B) * monom\ 1\ (d_A - card\ (A{-}B))$
define *sq* **where** $sq = set\text{-}to\text{-}poly\ (B{-}A) * monom\ 1\ (d_B - card\ (B{-}A))$

let $?x = (\chi\ (i::'m).\ if\ (to\text{-}nat\ i) < d_A\ then\ coeff\ sp\ (to\text{-}nat\ i)\ else\ coeff\ sq\ (to\text{-}nat\ i - d_A))$

have *poly-mul-eq*: $f_A\ x * poly\ sq\ x = f_B\ x * poly\ sp\ x$ **if** $x \in set\ E$ **for** x
proof −
have $set\text{-}to\text{-}poly\ A * set\text{-}to\text{-}poly\ (B - A) = (set\text{-}to\text{-}poly\ B) * set\text{-}to\text{-}poly\ (A - B)$
by (*simp add*: *Un-commute set-to-poly-mult-distinct*)
then have $(set\text{-}to\text{-}poly\ A * set\text{-}to\text{-}poly\ (B - A) * monom\ 1\ (d_B - card\ (B - A)) =$
$(set\text{-}to\text{-}poly\ B) * set\text{-}to\text{-}poly\ (A - B) * monom\ 1\ (d_A - card\ (A - B)))$
using *assms*(*6*) **by** *argo*
hence $poly\ (set\text{-}to\text{-}poly\ A)\ x * poly\ (set\text{-}to\text{-}poly\ (B - A) * monom\ 1\ (d_B - card\ (B - A)))\ x =$
$poly\ (set\text{-}to\text{-}poly\ B)\ x * poly\ (set\text{-}to\text{-}poly\ (A - B) * monom\ 1\ (d_A - card\ (A - B)))\ x$
by (*metis* (*no-types*, *lifting*) *mult.commute mult.left-commute poly-mult*)
thus *?thesis*
using *that* **unfolding** *assms sp-def sq-def* **by** *simp*
qed

have *x-sol-raw*: $(\sum j \in \{0..{<}d_A\}.\ e\ \hat{}\ j * coeff\ sp\ j\ ) + (\sum j \in \{0..{<}d_B\}.\ - f\ e * e\ \hat{}\ j * (coeff\ sq\ j))$
$= f\ e * e\ \hat{}\ d_B - e\ \hat{}\ d_A$ **if** $e \in set\ E$ **for** e
proof −
have f_Az: $f_A\ e \neq 0$
using *assms* (*7*,*9*) *in-set-to-poly that* **by** *auto*
moreover have f_Bz: $f_B\ e \neq 0$
using *assms* (*8*,*10*) *in-set-to-poly that* **by** *auto*
ultimately have *fz*: $f\ e \neq 0$
using *that assms*(*1*) **by** *simp*

have ff_B: $f\ e = f_A\ e\ /\ f_B\ e$
using *that assms*(*1*) **by** *simp*

have *lead-coeff sp* = *1*
unfolding *sp-def lead-coeff-mult* **using** *set-to-poly-lead-coeff lead-coeff-monom*
by (*auto simp add*: *degree-monom-eq*)
moreover have *degree sp* = d_A
unfolding *sp-def* **using** *assms*(*4*)
by (*simp add*: *add-diff-inverse-nat degree-monom-eq degree-mult-eq order-less-imp-not-less set-to-poly-degree*)
ultimately have *poly-sp*: *poly sp e* = ($\sum$ $i<d_A$. *coeff sp i* ∗ *e* ^ *i*) + *e* ^ d_A **for** *e*
unfolding *poly-lead-coeff-extract* **by** *simp*

have *lead-coeff sq* = *1*
unfolding *sq-def lead-coeff-mult* **using** *set-to-poly-lead-coeff lead-coeff-monom*
by (*auto simp add*: *degree-monom-eq*)
moreover have *degree sq* = d_B
using *assms*(*5*) **unfolding** *sq-def*
by (*simp add*: *degree-monom-eq degree-mult-eq le-eq-less-or-eq set-to-poly-degree*)
ultimately have *poly-sq*: *poly sq e* = ($\sum$ $i<d_B$. *coeff sq i* ∗ *e* ^ *i*) + *e* ^ d_B **for** *e*
unfolding *poly-lead-coeff-extract* **by** *simp*

have f_B *e* ∗ (($\sum$ *i* = *0*..<d_A. *coeff sp i* ∗ *e* ^ *i*) + *e* ^ d_A) =
f_A *e* ∗ (($\sum$ *i* = *0*..<d_B. *coeff sq i* ∗ *e* ^ *i*) + *e* ^ d_B)
using *that poly-mul-eq* **unfolding** *poly-sq poly-sp lessThan-atLeast0* **by** *simp*
then have f_B *e* ∗ (($\sum$ *j* = *0*..<d_A. *e* ^ *j* ∗ *coeff sp j*) + *e* ^ d_A) =
f_A *e* ∗ (($\sum$ *j* = *0*..<d_B. *e* ^ *j* ∗ *coeff sq j*) + *e* ^ d_B)
by (*metis* (*lifting*) *Finite-Cartesian-Product.sum-cong-aux mult.commute*)
then have ($\sum$ *j* = *0*..<d_A. *e* ^ *j* ∗ *coeff sp j*) + *e* ^ d_A =
f e ∗ (($\sum$ *j* = *0*..<d_B. *e* ^ *j* ∗ *coeff sq j*) + *e* ^ d_B)
unfolding ff_B **using** f_Bz
by (*metis* (*no-types*, *lifting*) f_Bz *nonzero-mult-div-cancel-left times-divide-eq-left*)
also have ... = *f e* ∗ ($\sum$ *j* = *0*..<d_B. *e* ^ *j* ∗ *coeff sq j*) + *f e* ∗ *e* ^ d_B
by *algebra*
also have ... = ($\sum$ *j* = *0*..<d_B. *f e* ∗ *e* ^ *j* ∗ *coeff sq j*) + *f e* ∗ *e* ^ d_B
by (*metis* (*no-types*, *lifting*) *Finite-Cartesian-Product.sum-cong-aux mult.assoc sum-distrib-left*)
finally have ($\sum$ *j* = *0*..<d_A. *e* ^ *j* ∗ *coeff sp j*) + *e* ^ d_A =
($\sum$ *j* = *0*..<d_B. *f e* ∗ *e* ^ *j* ∗ *coeff sq j*) + *f e* ∗ *e* ^ d_B
by *argo*
then have ($\sum$ *j* = *0*..<d_A. *e* ^ *j* ∗ *coeff sp j*) =
($\sum$ *j* = *0*..<d_B. *f e* ∗ *e* ^ *j* ∗ *coeff sq j*) + *f e* ∗ *e* ^ d_B − *e* ^ d_A
using *add-implies-diff* **by** *blast*
then have ($\sum$ *j* = *0*..<d_A. *e* ^ *j* ∗ *coeff sp j*) + (− ($\sum$ *j* = *0*..<d_B. *f e* ∗ *e* ^ *j* ∗ *coeff sq j*)) =
f e ∗ *e* ^ d_B − *e* ^ d_A
by *auto*

moreover have $-(\sum j = 0..<d_B.\ f\ e * e\ \hat{}\ j * (coeff\ sq\ j)) =$
$(\sum j = 0..<d_B.\ - f\ e * e\ \hat{}\ j * coeff\ sq\ j)$
using *sum-negf* [*symmetric*] **by** *auto*
ultimately show *?thesis*
by *argo*
qed

let *?const-lam* $= \lambda e.\ f\ e * e\ \hat{}\ d_B - e\ \hat{}\ d_A$
let *?const-lam'* $= \lambda i.$ *?const-lam* $(E\ !\ i)$
let *?coeff-lam* $= \lambda e\ j.(if\ j < d_A\ then\ e\ \hat{}\ j$
$else\ if\ j < d_A + d_B$
$then - f\ e * e\ \hat{}\ (j - d_A)\ else\ 0) *$
$(if\ j < d_A\ then\ coeff\ sp\ j\ else\ coeff\ sq\ (j - d_A))$
let *?coeff-lam'* $= \lambda i$. *?coeff-lam* $(E\ !\ i)$

have $(\sum j \in \{0..<length\ E\}.$ *?coeff-lam* $e\ j) =$ *?const-lam* e **if** $e \in set\ E$ **for** e
proof −
have $(\sum j \in \{0..<length\ E\}.$ *?coeff-lam* $e\ j) = (\sum j \in \{0..<d_A + d_B\}.$ *?coeff-lam* $e\ j)$
using *assms(3)* **by** (*intro sum.mono-neutral-cong-right*) *auto*
also have $\ldots = (\sum j\in\{0..<d_A\}.\ e\hat{}j * coeff\ sp\ j) + (\sum j \in \{0..<d_B\}. -f\ e * e\hat{}j*(coeff\ sq\ j))$
proof −
have $(\sum j \in \{0..<d_A + d_B\}.$ *?coeff-lam* $e\ j) =$
$(\sum j \in \{0..<d_A\}.$ *?coeff-lam* $e\ j) + (\sum j \in \{d_A..<d_A + d_B\}.$ *?coeff-lam* $e\ j)$
by (*intro sum.atLeastLessThan-concat* [*symmetric*]) *auto*
also have $\ldots = (\sum j \in \{0..<d_A\}.\ e\ \hat{}\ j * coeff\ sp\ j) +$
$(\sum j \in \{d_A..<d_A + d_B\}. - f\ e * e\ \hat{}(j - d_A) * (coeff\ sq\ (j-d_A)))$
by *simp*
moreover have $(\sum j \in \{d_A..<d_A + d_B\}. - f\ e * e\ \hat{}(j - d_A) * (coeff\ sq\ (j-d_A))) =$
$(\sum j \in \{0..<d_B\}. - f\ e * e\ \hat{}(j) * (coeff\ sq\ j))$
by (*rule sum.reindex-bij-witness* [*of* - $\lambda i.\ i + d_A\ \lambda i.\ i - d_A$]) *auto*
ultimately show *?thesis*
by *simp*
qed
also have ... = *?const-lam* e
using *that x-sol-raw* **by** *simp*
finally show *?thesis* **by** *simp*
qed
then have $(\sum j \in \{0..<length\ E\}.$ *?coeff-lam'* $i\ j) =$ *?const-lam'* i **if** $i < length\ E$ **for** i
using *that* **by** *simp*
moreover have $(\sum j\in(UNIV::'m\ set).$ *?coeff-lam* $i\ (to\text{-}nat\ j)) = (\sum j \in \{0..<CARD('m)\}.$ *?coeff-lam* $i\ j)$ **for** i

using *bij-to-nat* **by** (*intro sum.reindex-bij-betw*) *blast*
ultimately have ($\sum j \in (UNIV::'m\ set)$. *?coeff-lam'* *i* (*to-nat j*)) = *?const-lam'* *i* **if** $i <$ *length E* **for** *i*
using *that assms* **using** *of-nat-eq-iff*[*of card top length E*] *assms*(*3*) **by** *force*
then have (λi. $\sum j \in (UNIV::'m\ set)$. *?coeff-lam'* *i* (*to-nat j*)) (*to-nat* (*i*::'*m*)) = *?const-lam'* (*to-nat i*) **for** *i*
using *mod-type-less-function-eq* [*of* (λi. $\sum j \in (UNIV::'m\ set)$. *?coeff-lam'* *i* (*to-nat j*)) *?const-lam'* *i*]
using *assms*(*2*) *assms*(*3*) **by** *auto*
then have *eval*: (λi. $\sum j \in (UNIV::'m\ set)$. *?coeff-lam'* (*to-nat* (*i*::'*m*)) (*to-nat j*)) =
(λ *i*. *?const-lam'* (*to-nat i*))
by *simp*

have *?coeff'* $*v$ *?x* = *?const'*
unfolding *matrix-vector-mult-def*
rfi-coefficient-matrix-def
rfi-constant-vector-def
using *eval* **by** *simp*
then show *?thesis*
unfolding *consistent-def is-solution-def* **by** *auto*
qed

2.5 Main lemma

lemma *rational-function-interpolation-correct*:
assumes
int $d_A = \lfloor (real\ (length\ E) + card\ A - card\ B)/2 \rfloor$
int $d_B = \lfloor (real\ (length\ E) + card\ B - card\ A)/2 \rfloor$
card (*sym-diff A B*) $\leq$ *length E*

$\forall\ x \in set\ E.\ x \notin A$ $\forall\ x \in set\ E.\ x \notin B$
$f_A = (\lambda\ x \in set\ E.\ poly\ (set\text{-}to\text{-}poly\ A)\ x)$
$f_B = (\lambda\ x \in set\ E.\ poly\ (set\text{-}to\text{-}poly\ B)\ x)$
CARD('*m*::*mod-type*) = *length E*
defines
sol $\equiv$ *solution-to-poly* (*rational-function-interpolation E* ($\lambda e.\ f_A\ e\ /\ f_B\ e$) d_A d_B *TYPE*('*m*)) d_A d_B
shows
monic-interpolated-rational-function (*fst sol*) (*snd sol*) (*set E*) f_A f_B d_A d_B
proof −
let *?f* = ($\lambda e.\ f_A\ e\ /\ f_B\ e$)
let *?S* = *rational-function-interpolation E* ($\lambda e.\ f_A\ e\ /\ f_B\ e$) d_A d_B *TYPE*('*m*)
let *?p* = *fst* (*solution-to-poly ?S* d_A d_B)
let *?q* = *snd* (*solution-to-poly ?S* d_A d_B)

have *f*:*finite A finite B*
using *finite* **by** *blast+*
note *pd-pq-props* = d_A-d_B-*helper*[*OF f assms*(*1*−*3*)]

have *consistent* (χ (*i*::'*m*) (*j*::'*m*). *rfi-coefficient-matrix E ?f* d_A d_B (*to-nat i*) (*to-nat j*))
(χ (*i*::'*m*). *rfi-constant-vector E ?f* d_A d_B (*to-nat i*))
using *assms pd-pq-props*
by (*intro rational-function-interpolation-consistent* [**where** $A = A$ **and** $B = B$ **and** $f_A = f_A$ **and** $f_B = f_B$])
auto
then have $\forall$ *e*$\in$*set E*. f_A *e* $*$ *poly ?q e* = f_B *e* $*$ *poly ?p e*
using *assms pd-pq-props*(*1*) *in-set-to-poly*
by (*intro rational-function-interpolation-correct-poly* [**where** *f* = *?f* **and** $d_A = d_A$ **and** $d_B = d_B$ **and** *S* = *?S*])
auto
moreover have *real* (*degree ?p*) = *real* d_A
using *degree-solution-to-poly-fst* **by** *auto*
moreover have *real* (*degree ?q*) = *real* d_B
using *degree-solution-to-poly-snd* **by** *auto*
moreover have *monic ?q*
using *monic-solution-to-poly-snd* **by** *auto*
moreover have *monic ?p*
using *monic-solution-to-poly-fst* **by** *auto*
ultimately show *?thesis* **using** *fst-solution-to-poly-nz snd-solution-to-poly-nz*
unfolding *monic-interpolated-rational-function-def sol-def* **by** *force*
qed

lemma *interpolated-rational-function-floor-eq*:
interpolated-rational-function p_A p_B *E* f_A f_B d_A d_B $\longleftrightarrow$
interpolated-rational-function p_A p_B *E* f_A f_B $\lfloor d_A \rfloor$ $\lfloor d_B \rfloor$
unfolding *interpolated-rational-function-def* **using** *nat-leq-real-floor* **by** *simp*

lemma *sym-diff-bound-div2-ge0*:
fixes *A B* :: '*a* :: *finite set*
assumes *card* (*sym-diff A B*) $\leq$ *length E*
shows (*real* (*length E*) + *card A* $-$ *card B*)/*2* $\geq$ *0*
proof $-$
have $*$: *finite A finite B* **using** *finite* **by** *auto*

have *0* $\leq$ *real* (*card* (*sym-diff A B*)) + *real* (*card* (*A*$-$*B*)) $-$ (*card* (*B*$-$*A*))
unfolding *card-sym-diff-finite*[*OF* $*$] **by** *simp*
also have ... $\leq$ *real* (*length E*) + *real* (*card* (*A*$-$*B*)) $-$ (*card* (*B*$-$*A*))
using *assms*(*1*) **by** *simp*
also have ... = (*real* (*length E*) + *card A* $-$ *card B*)
using *card-sub-int-diff-finite* [*OF* $*$] **by** *simp*
finally show *?thesis* **by** *simp*
qed

If the degrees are reals we take the floor first

lemma *rational-function-interpolation-correct-real*:

```
  fixes d'_A d'_B:: real
  assumes
    card (sym-diff A B) ≤ length E
    ∀ x ∈ set E. x ∉ A ∀ x ∈ set E. x ∉ B
    f_A = (λ x ∈ set E. poly (set-to-poly A) x)
    f_B = (λ x ∈ set E. poly (set-to-poly B) x)
    CARD('m::mod-type) = length E
  defines d'_A ≡ (real (length E) + card A − card B)/2
  defines d'_B ≡ (real (length E) + card B − card A)/2
  defines d_A ≡ nat ⌊d'_A⌋
  defines d_B ≡ nat ⌊d'_B⌋
  defines sol-poly ≡ interpolate-rat-fun E (λe. f_A e / f_B e) d_A d_B TYPE('m)
  shows
    monic-interpolated-rational-function (fst sol-poly) (snd sol-poly) (set E) f_A f_B
d'_A d'_B
proof −
  have e: d'_A ≥ 0
    unfolding d'_A-def using sym-diff-bound-div2-ge0 assms(1) by auto

  hence a: int d_A = ⌊(real (length E) + real (card A) − real (card B)) / 2⌋
    using d'_A-def unfolding d_A-def by simp

  have f: d'_B ≥ 0
   unfolding d'_B-def using sym-diff-bound-div2-ge0 assms(1) by (metis Un-commute)

  hence b: int d_B = ⌊(real (length E) + real (card B) − real (card A)) / 2⌋
    using d'_B-def unfolding d_B-def by simp

  have c: monic-interpolated-rational-function (fst sol-poly) (snd sol-poly) (set E)
f_A f_B d_A d_B
    unfolding sol-poly-def interpolate-rat-fun-def
    by (intro rational-function-interpolation-correct [OF a b assms(1−6)])
  moreover have ⌊d'_A⌋ = real (nat ⌊d'_A⌋)
    using e by (intro of-nat-nat[symmetric]) simp
  moreover have ⌊d'_B⌋ = real (nat ⌊d'_B⌋)
    using f by (intro of-nat-nat[symmetric]) simp
  ultimately have
    monic-interpolated-rational-function (fst sol-poly) (snd sol-poly) (set E) f_A f_B
(nat ⌊d'_A⌋) (nat ⌊d'_B⌋)
    unfolding d_A-def d_B-def
    by simp
  thus ?thesis unfolding monic-interpolated-rational-function-def
    using assms(9,10) a b d'_A-def d'_B-def floor-of-nat by simp
qed

end
```

3 Factorisation of Polynomials

theory *Factorisation*
  imports
    Berlekamp-Zassenhaus.Finite-Field
    Berlekamp-Zassenhaus.Finite-Field-Factorization
    Elimination-Of-Repeated-Factors.ERF-Perfect-Field-Factorization
    Elimination-Of-Repeated-Factors.ERF-Algorithm
begin

hide-const (**open**) *Coset.order*
hide-const (**open**) *module.smult*
hide-const (**open**) *UnivPoly.coeff*
hide-const (**open**) *Formal-Power-Series.radical*

lemma *proots-finite-field-factorization*:
  assumes
    square-free f
    finite-field-factorization f = (*c*, *us*)
  shows *proots f* = *sum-list* (*map proots us*)
proof −
  have *fffp*: *f* = *smult c* (*prod-list us*) (∀ *u* ∈ *set us*. *monic u* ∧ *irreducible u*)
    using *finite-field-factorization-explicit assms* **by** *auto*
  then have *0* ∉ *set us*
    by *blast*
  then have *proots* (∏ *u*←*us*. *u*) = (∑ *u*←*us*. *proots u*)
    using *proots-prod-list fffp* **by** *auto*
  then show *?thesis* **using** *assms*
    by (*simp add*: *fffp square-free-def*)
qed

The following fact is an improved version of *?x* ≠ *0* ⟹ *squarefree ?x* = *square-free ?x*, which does not require the assumtion that *p* ≠ *0*.

lemma *squarefree-square-free'*:
  fixes *p* :: *'a*:: *field poly*
  shows *squarefree p* = *square-free p*
  by (*metis not-squarefree-0 square-free-def squarefree-square-free*)

This function returns the roots of an irreducible polynomial:

fun *extract-root* :: *'a*::*prime-card mod-ring poly* ⇒ *'a mod-ring multiset* **where**
  extract-root p = (*if degree p* = *1 then* {# − *coeff p 0* #} *else* {#})

lemma *degree1-monic*:
  assumes *degree p* = *1*
  assumes *monic p*
  obtains *c* **where** *p* = [:*c*,*1*:]
proof −
  obtain *a b* **where** *op*: *p* = [: *b*, *a* :]
    using *degree1-coeffs assms*(*1*) **by** *meson*

then have *a* = *1*
  using *assms* **by** *simp*
then show *?thesis*
  using *op* **using** *that* **by** *simp*
qed

lemma *extract-root*:
  assumes *monic p irreducible p*
  shows *extract-root p* = *proots p*
proof −
  consider (*A*) *degree p* = *0* | (*B*) *degree p* = *1* | (*C*) *degree p* > *1*
    by *linarith*
  thus *?thesis*
  proof (*cases*)
    case *A*
    hence *extract-root p* = {#} **by** *simp*
    also have ... = *proots 1* **by** *simp*
    also have ... = *proots p* **using** *A assms*(*1*) *monic-degree-0* **by** *blast*
    finally show *?thesis* **by** *simp*
  next
    case *B*
    obtain *c* **where** *p-def*: *p* = [:*c*,*1*:]
      using *assms*(*1*) *B degree1-monic* **by** *blast*

    hence *proots p* = {#−*c*#}
      using *proots-linear-factor* **by** *blast*
    also have ... = *extract-root p*
      unfolding *p-def* **by** *simp*
    finally show *?thesis* **by** *simp*
  next
    case *C*
    have *False* **if** *x* ∈# *proots p* **for** *x*
    proof −
      have *p* ≠ *0* **using** *C* **by** *auto*
      hence *poly p x* = *0* **using** *set-count-proots that* **by** *simp*
      thus *False* **using** *C assms root-imp-reducible-poly* **by** *blast*
    qed
    hence *proots p* = {#} **by** *auto*
    also have ... = *extract-root p*
      using *C* **by** *simp*
    finally show *?thesis* **by** *simp*
  qed
qed

fun *extract-roots* :: *'a*::*prime-card mod-ring poly list* ⇒ *'a mod-ring multiset* **where**
  extract-roots [] = {#}
| *extract-roots* (*p*#*ps*) = *extract-root p* + *extract-roots ps*

lemma *extract-roots*:

```
  ∀ p ∈ set ps. monic p ∧ irreducible p ⟹
    sum-list (map proots ps) = extract-roots ps
proof (induction ps)
  case Nil
  then show ?case by simp
next
  case (Cons p ps)
  have sum-list (map proots (p # ps)) = proots p + sum-list (map proots ps) by
simp
  also have ... = extract-root p + sum-list (map proots ps)
    using Cons(2) by (subst extract-root) auto
  also have ... =  extract-roots (p # ps) using Cons by simp
  finally show ?case by simp
qed

lemma proots-extract-roots-factorized:
  assumes squarefree p
  shows proots p = extract-roots (snd (finite-field-factorization p))
proof -
  have sf:square-free p
    using squarefree-square-free' assms by blast

  have proots p = sum-list (map proots (snd (finite-field-factorization p)))
    using proots-finite-field-factorization[OF sf] by (metis prod.collapse)
  also have ... = extract-roots (snd (finite-field-factorization p))
    using finite-field-factorization-explicit[OF sf]
    by (intro extract-roots) (metis prod.collapse)
  finally show ?thesis  by simp
qed
```

3.1 Elimination of Repeated Factors

Wrapper around the ERF algorithm, which returns each factor with multiplicity in the input polynomial

```
function ERF' where
  ERF' p = (
    if degree p = 0 then [] else
      let factors = ERF p in
        ERF' (p div (prod-list factors)) @ factors)
  by auto

lemma degree-zero-iff-no-factors:
  fixes p :: 'a :: {factorial-ring-gcd,semiring-gcd-mult-normalize,field} poly
  assumes p ≠ 0
  shows prime-factors p = {} ⟷ degree p = 0
proof
  assume prime-factors p = {}
  hence is-unit p using assms
    by (meson prime-factorization-empty-iff set-mset-eq-empty-iff)
```

```
  thus degree p = 0
    using poly-dvd-1 by blast
next
  assume degree p  = 0
  thus prime-factors p = {} using assms prime-factors-degree0 by metis
qed

lemma ERF'-termination:
  assumes degree p > 0
  shows degree (p div prod-list (ERF p)) < degree p
proof (intro degree-div-less)
  show p-ne-0: p ≠ 0 using assms by auto

  have a:radical p = prod-list (ERF p)
    using p-ne-0 ERF-correct(1) by metis

  show prod-list (ERF p) dvd p unfolding a[symmetric] by (rule radical-dvd)

  have prime-factors p ≠ {}
    using p-ne-0 assms(1) degree-zero-iff-no-factors[OF p-ne-0] by simp
  hence prime-factors (radical p) ≠ {}
    using p-ne-0 prime-factors-radical by metis
  moreover have radical p ≠ 0
    using radical-eq-0-iff p-ne-0 by auto
  ultimately have degree (radical p) > 0
    using degree-zero-iff-no-factors by blast

  thus degree (prod-list (ERF p)) ≠ 0
    using a by simp
qed

termination
  using ERF'-termination
  by (relation measure degree) auto

lemma ERF'-squarefree:
  assumes x ∈ set (ERF' p)
  shows squarefree x using assms
proof (induct p rule: ERF'.induct)
  case (1 p)
  define factors where factors = ERF p
  show ?case
  proof (cases degree p > 0)
    case True
    hence a: ERF' p = ERF' (p div prod-list factors) @ factors
      unfolding factors-def
      by (subst ERF'.simps) (simp add:Let-def)
    hence x ∈ set (ERF' (p div prod-list factors)) ∨ x ∈ set (factors)
      using 1(2) unfolding a  by simp
```

moreover have *x* ∈ *set* (*factors*) ⟹ *squarefree x*
using *ERF-correct*(*2*) *True factors-def*
by (*metis degree-0 order-less-irrefl*)
ultimately show *?thesis*
using *1*(*1*)[*OF - factors-def*] *True* **by** *auto*
next
case *False*
hence *ERF′ p* = [] **by** *simp*
thus *?thesis* **using** *1*(*2*) **by** *simp*
qed
qed

lemma *ERF-not0*: *p* ≠ *0* ⟹ *0* ∉ *set* (*ERF p*)
using *ERF-correct*(*2*) *not-squarefree-0* **by** *blast*

lemma *ERF′-not0*: *0* ∉ *set* (*ERF′ p*)
using *ERF′-squarefree not-squarefree-0* **by** *blast*

lemma *ERF′-proots*: *proots* (∏ *x*← *ERF′ p*. *x*) = *proots p*
proof (*induct p rule*: *ERF′.induct*)
case (*1 p*)
show *?case*
proof (*cases degree p* > *0*)
case *True*
let *?prod* = *prod-list* (*ERF p*)

have *a*:*ERF′ p* = *ERF′* (*p div ?prod*) @ (*ERF p*)
unfolding *factors-def*
by (*subst ERF′.simps*) (*simp add*:*Let-def*)

have *h*: *proots* (∏ *x*←*ERF′* (*p div ?prod*). *x*) = *proots* (*p div ?prod*)
using *1 True* **by** *simp*

have *p0*: *p* ≠ *0*
using *True* **by** *force*
then have *l0*: *?prod* ≠ *0*
using *ERF-not0* **by** *simp*

have *radical p dvd p*
by *simp*
then have *pdvd*: *?prod dvd p*
using *ERF-correct*(*1*) *p0* **by** *metis*
then have *d0*: (*p div ?prod*) ≠ *0*
using *p0* **using** *dvd-div-eq-0-iff* **by** *blast*

have *proots* (*p div ?prod*) + *proots ?prod* =
proots (*p div ?prod* ∗ *?prod*)
using *proots-mult l0 d0* **by** *metis*

then have *1*: *proots p* = *proots* (*p div ?prod*) + *proots ?prod*
using *pdvd* **by** *simp*

have ($\prod$ *x*←*ERF′* (*p div ?prod*). *x*) ≠ *0*
using *ERF′-not0* **by** *force*
then have *proots* ($\prod$ *x*←*ERF′* (*p div ?prod*). *x*) + *proots ?prod*
= *proots* (($\prod$ *x*←*ERF′* (*p div ?prod*). *x*) ∗ *?prod*)
using *proots-mult l0* **by** *metis*
also have ... = *proots* ($\prod$ *x*←*ERF′ p*. *x*)
using *a* **by** *force*
finally have *proots* ($\prod$ *x*←*ERF′ p*. *x*) = *proots* (*p div ?prod*) + *proots ?prod*
using *h* **by** *argo*

then show *?thesis* **using** *1* **by** *argo*
next
case *False*
then have *deg*: *degree p* = *0*
by *simp*
then have *ERF′ p* = []
by (*subst ERF′.simps*) *simp*
then have *1*: *proots* ($\prod$ *x*←*ERF′ p*. *x*) = {#}
by *simp*
from *deg* **obtain** *x* **where** *p* = [:*x*:]
using *degree-eq-zeroE* **by** *blast*
then have *proots p* = {#}
by *simp*
thus *?thesis* **using** *1* **by** *simp*
qed
qed

3.2 Executable version of *proots*

fun *proots-eff* :: *′a*::*prime-card mod-ring poly* ⇒ *′a mod-ring multiset* **where**
proots-eff p = *sum-list* (*map* (*extract-roots* ∘ *snd* ∘ *finite-field-factorization*) (*ERF′ p*))

lemma *proots-eff-correct* [*code-unfold*]: *proots p* = *proots-eff p*
proof −
have *proots p* = *proots* ($\prod$ *x*← *ERF′ p*. *x*)
using *ERF′-proots* **by** *metis*
also have ... = *sum-list* (*map proots* (*ERF′ p*))
using *ERF′-squarefree not-squarefree-0* **by** (*intro proots-prod-list*) *blast*
also have ... = *sum-list* (*map* (*extract-roots* ∘ *snd* ∘ *finite-field-factorization*) (*ERF′ p*))
using *proots-extract-roots-factorized*[*OF ERF′-squarefree*]
by (*intro arg-cong*[**where** *f*=*sum-list*] *map-cong refl*) (*auto simp add*:*comp-def*)
finally show *?thesis* **by** *simp*
qed

3.3 Executable version of *order*

fun *order-eff* :: *'a mod-ring* ⇒ *'a::prime-card mod-ring poly* ⇒ *nat* **where**
  order-eff x p = *count* (*proots-eff p*) *x*

lemma *order-eff-code* [*code-unfold*]: $p \neq 0 \Longrightarrow$ *order x p* = *order-eff x p*
  unfolding *order-eff.simps proots-eff-correct* [*symmetric*] *count-proots*
  by *auto*

end

4 Set Reconciliation Algorithm

theory *Set-Reconciliation*
  imports
    HOL−Library.FuncSet
    HOL−Computational-Algebra.Polynomial
    Factorisation
    Rational-Function-Interpolation
begin

hide-const (**open**) *up-ring.monom*

The following locale introduces the context for the reconciliation algorithm. It fixes parameters that are assumed to be known in advance, in particular:

- a bound m on the symmetric difference: represented using the type variable $'m$
- the finite field used to represent the elements of the sets: represented using the type variable $'a$
- the evaluation points used (which must be choosen outside of the domain used to represent the elements of the sets): represented using the variable E

To preserve generality as much as possible, we only present an interaction protocol that allows one party Alice to send a message to the second party Bob, who can reconstruct the set Alice has, assuming Bob holds a set himself, whose symmetric difference does not exceed m.

Note that using this primitive, it is possible for Bob to compute the union of the sets, and of course the algorithm can also be used to send a message from Bob to Alice, such that Alice can do so as well. However, the primitive we describe can be used in many other scenarios.

locale *set-reconciliation-algorithm* =
  fixes *E* :: *'a* :: *prime-card mod-ring list*
  fixes *phantom-m* :: *'m::mod-type itself*

assumes *type-m*: *phantom-m* = *TYPE*('*m*)
assumes *distinct-E*: *distinct E*
assumes *card-m*: *CARD*('*m*) = *length E*
begin

The algorithm—or, more precisely the protocol—is represented using a pair of algorithms. The first is the encoding function which Alice used to create the message she sends. The second is the decoding algorithm, which Bob can use to reconstruct the set Alice has.

definition *encode* **where**
encode A = (*card A*, λ *x* ∈ *set E*. *poly* (*set-to-poly A*) *x*)

definition *decode* **where**
decode B R =
(*let*
(*n*, f_A) = *R*;
f_B = (λ *x* ∈ *set E*. *poly* (*set-to-poly B*) *x*);
d_A = *nat* ⌊(*real* (*length E*) + *n* − *card B*) / *2*⌋;
d_B = *nat* ⌊(*real* (*length E*) + *card B* − *n*) / *2*⌋;
(p_A,p_B) = *interpolate-rat-fun E* (λ*x*. f_A *x* / f_B *x*) d_A d_B *phantom-m*;
r_A = *proots-eff* p_A;
r_B = *proots-eff* p_B
in
set-mset (r_A − r_B) ∪ (*B* − (*set-mset* (r_B − r_A))))

4.1 Informal Description of the Algorithm

The protocol works as follows:

We association with each set A a polynomial $\chi_A(x) := \prod_{s \in A}(x - s)$ in the finite field F. As mentioned before we reserve a set of m evaluation points E, which can be arbitrary prearranged points, as long as they are field elements not used to represent set elements.

Then Alice sends the size of its set |A| and the evaluation of its characteristic polynomial on E.

Bob computes

$$\begin{aligned} d_A &:= \left\lfloor \frac{|E| + |A| - |B|}{2} \right\rfloor \\ d_B &:= \left\lfloor \frac{|E| + |B| - |A|}{2} \right\rfloor \end{aligned}$$

Then Bob finds monic polynomials p_A, p_B of degree d_A and d_B fulfilling the condition:

$$p_A(x)\chi_B(x) = p_B(x)\chi_A(X) \text{ for all } x \in E \tag{1}$$

The above results in a system of linear equations, which can be solved using Gaussian elimination. It is easy to show that the system is solvable since:

$$p_A := \chi_{A-B}(x)x^r$$

$$p_B \quad := \quad \chi_{B-A}(x)x^r$$

is a solution, where $r := d_A - |A - B| = d_B - |B - A|$.

The equation (Eq. 1) implies also:

$$p_A(x)\chi_{B-A}(x) = p_B(x)\chi_{A-B}(x) \text{ for all } x \in E \tag{2}$$

since $\chi_A(x) = \chi_{A-B}(x)\chi_{A\cap B}(x)$, $\chi_B(x) = \chi_{B-A}(x)\chi_{A\cap B}(x)$, and $\chi_{A\cap B}(x) \neq 0$, because of our constraint that E is outside of the universe of the set elements. Btw. in general

$$\chi_{U\cup V} = \chi_U \chi_V \text{ for any disjoint } U, V.$$

Because the polynomials on both sides of Eq. 2 are *monic* polynomials of the same degree m', where $m' \leq m$, and agree on m points, they must be equal.

This implies in particular, that for the order of any root x (denoted by ord_x), we have:

$$\text{ord}_x(p_A\chi_{B-A}) = \text{ord}_x(p_B\chi_{A-B})$$

which implies:

$$\text{ord}_x(p_A) - \text{ord}_x(p_B) = \text{ord}_x(\chi_{B-A}) - \text{ord}_x(\chi_{A-B}).$$

Note that by definition the right hand side is equal to $+1$ if $x \in B - A$, -1 if $x \in A - B$ and 0 otherwise. Thus Bob can compute A using

$$A := \{x|\text{ord}_x(p_A) - \text{ord}_x(p_B) > 0\} \cup (B - \{x|\text{ord}_x(p_A) - \text{ord}_x(p_B) < 0\}).$$

4.2 Lemmas

This is no longer used, but it will be needed if you implement decode using an interpolation algorithm that does not return monic polynomials.

lemma *interpolated-rational-function-eq*:
  assumes
    $\forall$ *x* $\in$ *set E. poly* (*set-to-poly A*) *x* $*$ *poly* p_B *x* = *poly* (*set-to-poly B*) *x* $*$ *poly* p_A *x*
    degree $p_A \leq$ (*real* (*length E*) + *card A* − *card B*)/*2*
    degree $p_B \leq$ (*real* (*length E*) + *card B* − *card A*)/*2*
    card (*sym-diff A B*) < *length E*
    set E $\cap$ *A* = {} *set E* $\cap$ *B* = {}
  shows *set-to-poly* (*A*−*B*) $*$ p_B = *set-to-poly* (*B*−*A*) $*$ p_A
proof −
  have *fin*: *finite A finite B*
    by *simp+*

  have *dA*: *degree* $p_A \leq$ (*real* (*length E*) + *card* (*A*−*B*) − *card* (*B*−*A*))/*2*

using *assms*(*2*) *card-sub-int-diff-finite*[*OF fin*] **by** *simp*
have *dB*: *degree* $p_B \leq$ (*real* (*length E*) + *card* (*B*−*A*) − *card* (*A*−*B*))/*2*
using *assms card-sub-int-diff-finite*[*OF fin*] **by** *simp*

have *set-to-poly A* = *set-to-poly* (*A*−*B*) ∗ *set-to-poly* (*A* ∩ *B*)
using *set-to-poly-mult-distinct*
by (*metis Int-Diff-Un Int-Diff-disjoint mult.commute*)
moreover have *set-to-poly B* = *set-to-poly* (*B*−*A*) ∗ *set-to-poly* (*A* ∩ *B*)
using *set-to-poly-mult-distinct*
by (*metis Int-Diff-Un Int-Diff-disjoint Int-commute mult.commute*)
ultimately have *inE*: *poly* (*set-to-poly* (*A*−*B*) ∗ p_B) *x* = *poly* (*set-to-poly* (*B*−*A*) ∗ p_A) *x*
if *x* ∈ *set E* **for** *x*
using *that assms* **by** (*auto simp*: *in-set-to-poly*)

have *real* (*degree* (*set-to-poly* (*A*−*B*) ∗ p_B)) ≤ *real* (*card* (*A*−*B*)) + *degree* p_B
by (*metis of-nat-add of-nat-le-iff degree-mult-le set-to-poly-degree*)
also have ... ≤ (*real* (*length E*) + (*real*(*card* (*B*−*A*)) + *card* (*A*−*B*)))/*2*
using *dB* **by** *simp*
also have ... < (*length E* + *length E*) / *2*
using *assms*(*4*) *card-sym-diff-finite*[*OF fin*] **by** *simp*
also have ... = *length E* **by** *simp*
finally have *l*: *degree* (*set-to-poly* (*A*−*B*) ∗ p_B) < *length E*
by *simp*

have *real* (*degree* (*set-to-poly* (*B*−*A*) ∗ p_A)) ≤ *real* (*card* (*B*−*A*)) + *degree* p_A
by (*metis of-nat-add of-nat-le-iff degree-mult-le set-to-poly-degree*)
also have ... ≤ (*length E* + (*card* (*B*−*A*) + *card* (*A*−*B*)))/*2*
using *dA* **by** *simp*
also have ... < (*length E* + *length E*) / *2*
using *assms*(*4*) *card-sym-diff-finite*[*OF fin*] **by** *simp*
also have ... = *length E* **by** *simp*
finally have *r*: *degree* (*set-to-poly* (*B*−*A*) ∗ p_A) < *length E*
by *simp*

have *set-to-poly* (*A*−*B*) ∗ p_B = *set-to-poly* (*B*−*A*) ∗ p_A
using *l r inE poly-eqI-degree distinct-card*[*OF distinct-E*]
by (*intro poly-eqI-degree*[**where** *A*=*set E*]) *auto*
then show *?thesis* **.**
qed

This is a specialized version of interpolated-rational-function-eq. Here the interpolated function are monic with exact degrees.

lemma *monic-interpolated-rational-function-eq*:
assumes
∀ *x* ∈ *set E*. *poly* (*set-to-poly A*) *x* ∗ *poly* p_B *x* = *poly* (*set-to-poly B*) *x* ∗ *poly* p_A *x*
degree p_A = ⌊(*real* (*length E*) + *card A* − *card B*)/*2*⌋
degree p_B = ⌊(*real* (*length E*) + *card B* − *card A*)/*2*⌋

card (*sym-diff* *A* *B*) ≤ *length* *E*
set *E* ∩ *A* = {} *set* *E* ∩ *B* = {}
monic p_A *monic* p_B
shows *set-to-poly* (*A*−*B*) ∗ p_B = *set-to-poly* (*B*−*A*) ∗ p_A (**is** *?lhs* = *?rhs*)
proof −
have *fin*: *finite* *A* *finite* *B*
by *simp*+
have *p0*: $p_A \neq 0$ $p_B \neq 0$
using *assms*(*7*, *8*) **by** *auto*

define *m′* **where** *m′* = ⌊(*real* (*length* *E*) + *card* (*B*−*A*) + *card* (*A*−*B*))/*2*⌋

note *s1* = *card-sub-int-diff-finite-real*[*OF* *fin*]
note *s2* = *card-sub-int-diff-finite-real*[*OF* *fin*(*2*,*1*)]

have *int* (*degree* *?lhs*) = *int* (*card* (*A*−*B*)) + *degree* p_B
using *set-to-poly-degree* *p0* *set-to-poly-not0* **by** (*subst* *degree-mult-eq*) *auto*
also have . . . = ⌊*card* (*A*−*B*) + (*real* (*length* *E*) + *card* (*B*−*A*) − *card* (*A*−*B*))/*2*⌋
using *assms*(*3*) *s2* **by** (*simp* *add*: *group-cancel.sub1*)
also have . . . = *m′* **unfolding** *m′-def* **by** *argo*
finally have *a*:*int* (*degree* *?lhs*) = *m′* **by** *simp*

have *int* (*degree* *?rhs*) = *int* (*card* (*B*−*A*)) + *degree* p_A
using *set-to-poly-degree* *p0* *set-to-poly-not0* **by** (*subst* *degree-mult-eq*) *auto*
also have . . . = ⌊*card* (*B*−*A*) + (*real* (*length* *E*) + *card* (*A*−*B*) − *card* (*B*−*A*))/*2*⌋
using *assms*(*2*) *s1* **by** (*simp* *add*: *group-cancel.sub1*)
also have . . . = *m′* **unfolding** *m′-def* **by** *argo*
finally have *b*:*int* (*degree* *?rhs*) = *m′* **by** *simp*

have *of-int* *m′* ≤ (*real* (*length* *E*) + *card* (*B*−*A*) + *card* (*A*−*B*))/*2*
unfolding *m′-def* **by** *linarith*
also have . . . ≤ (*real* (*length* *E*) + *real* (*length* *E*))/*2*
using *assms*(*4*) *card-sym-diff-finite*[*OF* *fin*] **by** *simp*
also have . . . ≤ *real* (*length* *E*) **by** *simp*
also have . . . = *real* (*card* (*set* *E*)) **using** *distinct-E* **by** (*simp* *add*: *distinct-card*)
finally have *c*: *m′* ≤ *card* (*set* *E*) **by** *simp*

have *t1*: *set-to-poly* *A* = *set-to-poly* (*A*−*B*) ∗ *set-to-poly* (*A* ∩ *B*)
by (*subst* *set-to-poly-mult-distinct*) (*auto* *intro*!:*arg-cong*[**where** *f*=*set-to-poly*])

have *t2*: *set-to-poly* *B* = *set-to-poly* (*B*−*A*) ∗ *set-to-poly* (*A* ∩ *B*)
by (*subst* *set-to-poly-mult-distinct*) (*auto* *intro*!:*arg-cong*[**where** *f*=*set-to-poly*])

have *d*: *poly* (*set-to-poly* (*A*−*B*) ∗ p_B) *x* = *poly* (*set-to-poly* (*B*−*A*) ∗ p_A) *x* **if** *x* ∈ *set* *E* **for** *x*
proof −
have *poly* (*set-to-poly* (*A* ∩ *B*)) *x* ≠ *0*
using *in-set-to-poly* *assms*(*5*,*6*) *that* **by** (*metis* *IntE* *disjoint-iff*)
thus *?thesis* **using** *that* *assms*(*1*) **unfolding** *t1* *t2* **by** *auto*

qed

show *?thesis*
apply (*intro poly-eqI-degree-monic*[**where** *A= set E*])
subgoal using *a b* **by** *simp*
subgoal using *a c* **by** *simp*
subgoal using *set-to-poly-lead-coeff monic-mult assms*(*8*) **by** *auto*
subgoal using *set-to-poly-lead-coeff monic-mult assms*(*7*) **by** *auto*
using *d* **by** *auto*
qed

4.3 Main Result

This is the main result of the entry. We show that the decoding algorithm, Bob uses, can reconstruct the set Alice has, if she has encoded with the encoding algorithm. Assuming the symmetric difference between the sets does not exceed the given bound.

theorem *decode-encode-correct*:
assumes
card (*sym-diff A B*) $\leq$ *length E*
set E $\cap$ *A* = {} *set E* $\cap$ *B* = {}
shows *decode B* (*encode A*) = *A*
proof −
let $?f_A$ = (λ *x* $\in$ *set E. poly* (*set-to-poly A*) *x*)
let $?f_B$ = (λ *x* $\in$ *set E. poly* (*set-to-poly B*) *x*)
let $?d_A$ = (*real* (*length E*) + *card A* − *card B*) / *2*
let $?d_B$ = (*real* (*length E*) + *card B* − *card A*) / *2*

define *p* **where** *def-pq*: *p* = *interpolate-rat-fun E* (λ*x.* $?f_A$ *x* / $?f_B$ *x*) (*nat* $\lfloor ?d_A \rfloor$) (*nat* $\lfloor ?d_B \rfloor$) *TYPE*('*m*)
define p_A p_B **where** *def-p-q*: p_A = *fst p* p_B = *snd p*

have *monic-interpolated-rational-function* (*fst p*) (*snd p*) (*set E*) $?f_A$ $?f_B$ $?d_A$ $?d_B$
unfolding *def-pq*
using *assms card-m* **by** (*intro rational-function-interpolation-correct-real*) *auto*
then have *monic-interpolated-rational-function* p_A p_B (*set E*) $?f_A$ $?f_B$ $?d_A$ $?d_B$
using *def-p-q* **by** *simp*

then have *irf*: $\forall$ *e* $\in$ *set E.* $?f_A$ *e* ∗ *poly* p_B *e* = $?f_B$ *e* ∗ *poly* p_A *e*
degree p_A = *floor* $?d_A$ *degree* p_B = *floor* $?d_B$
monic p_A *monic* p_B
unfolding *monic-interpolated-rational-function-def* **by** *auto*

have *n0*: $p_A \neq 0$ $p_B \neq 0$
using *monic0 irf* **by** *auto*

have $\forall$*x*$\in$ *set E. poly* (*set-to-poly A*) *x* ∗ *poly* p_B *x* = *poly* (*set-to-poly B*) *x* ∗ *poly* p_A *x*

using *irf*(*1*) **by** *simp*
then have *ieq*: *set-to-poly* (*A*−*B*) ∗ p_B = *set-to-poly* (*B*−*A*) ∗ p_A
using *assms irf* **by** (*intro monic-interpolated-rational-function-eq*) *auto*

have *order x* (*set-to-poly* (*A*−*B*) ∗ p_B) = *order x* (*set-to-poly* (*A*−*B*)) + *order x* p_B **for** *x*
using *irf*(*5*) *n0* **by** (*simp add*: *order-mult*)
moreover have *order x* (*set-to-poly* (*B*−*A*) ∗ p_A) = *order x* (*set-to-poly* (*B*−*A*)) + *order x* p_A **for** *x*
using *irf*(*4*) *n0* **by** (*simp add*: *order-mult*)
ultimately have *order x* (*set-to-poly* (*A*−*B*)) + *order x* p_B =
order x (*set-to-poly* (*B*−*A*)) + *order x* p_A **for** *x*
using *ieq* **by** *simp*
hence *int* (*order x* (*set-to-poly* (*A*−*B*))) + *int* (*order x* p_B) =
int (*order x* (*set-to-poly* (*B*−*A*))) + *int* (*order x* p_A) **for** *x*
using *of-nat-add* **by** *metis*
then have *oif*: *int* (*order x* (*set-to-poly* (*A*−*B*))) − *int* (*order x* (*set-to-poly* (*B*−*A*))) =
int (*order x* p_A) − *int* (*order x* p_B) **for** *x*
by (*simp add*:*field-simps*)

have *int* (*order x* p_A) − *int* (*order x* p_B) ≥ *1* ⟷ *x* ∈ (*A*−*B*) **for** *x*
unfolding *oif*[*symmetric*] *set-to-poly-order* **by** *simp*
hence *a-minus-b*: {*x*. *order x* p_A > *order x* p_B} = *A*−*B* **by** *force*

have *int* (*order x* p_A) − *int* (*order x* p_B) ≤ −*1* ⟷ *x* ∈ (*B*−*A*) **for** *x*
unfolding *oif*[*symmetric*] *set-to-poly-order* **by** *simp*
hence *b-minus-a*: {*x*. *order x* p_B > *order x* p_A} = *B*−*A* **by** *force*

have {*x*. *order x* p_A > *order x* p_B} ∪ (*B* − {*x*. *order x* p_A < *order x* p_B}) = *A*
unfolding *a-minus-b b-minus-a* **by** *auto*

moreover have *decode B* (*encode A*) =
set-mset (*proots-eff* p_A − *proots-eff* p_B) ∪ (*B* − (*set-mset* (*proots-eff* p_B − *proots-eff* p_A)))
unfolding *decode-def encode-def Let-def def-p-q def-pq*
using *type-m* **by** (*simp add*:*case-prod-beta del*:*proots-eff*.*simps*)
moreover have . . . = {*x*. *order x* p_A > *order x* p_B} ∪ (*B* − {*x*. *order x* p_B > *order x* p_A})
unfolding *proots-eff-correct* [*symmetric*]
using *irf*(*4*,*5*) *n0* **by** (*auto simp*: *set-mset-diff*)

ultimately show *?thesis* **by** *argo*
qed

end

end

References

[1] Y. Minsky, A. Trachtenberg, and R. Zippel. Set reconciliation with nearly optimal communication complexity. *IEEE Transactions on Information Theory*, 49(9):2213–2218, 2003.