# A Set Reconciliation Algorithm

Paul Hofmeier and Emin Karayel

February 6, 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**

Taken from Budan-Fourier.BF-Misc

**lemma** *order-linear*[*simp*]: *order x* [:−*y*, *1*:] = (*if x*=*y then 1 else 0*)
⟨*proof*⟩

## 1.1 On Polynomial Roots

**lemma** *proots-empty*: *proots p* = {#} ⟷ *p* = *0* ∨ (∀ *x*. *poly p x* ≠ *0*)
⟨*proof*⟩

**lemma** *proots-element*: *x* ∈# *proots p* ∨ *p* = *0* ⟷ *poly p x* = *0*
⟨*proof*⟩

**lemma** *proots-diff*:
**assumes** *p* ≠ *0 q* ≠ *0*
**shows** *set-mset* (*proots p* − *proots q*) = {*x*. *order x p* > *order x q*} (**is** *?L* = *?R*)
⟨*proof*⟩

## 1.2 On *rsquarefree*

The following fact is an improved version of ⟦*rsquarefree ?p*; *poly ?p ?z* = *0*; *?p* ≠ *0*⟧ ⟹ *order ?z ?p* = *1*, which does not require the assumption that *p* ≠ *0*.

**lemma** *rsquarefree-root-order′*: *rsquarefree p* ⟹ *poly p x* = *0* ⟹ *order x p* = *1*
⟨*proof*⟩

**lemma** *rsquarefree-single-root*[*simp*]: *rsquarefree* [:−*x*,*1*:]
⟨*proof*⟩

**lemma** *rsquarefree-mul*:
**assumes** *rsquarefree p rsquarefree q*
∀ *x*. *poly p x* ≠ *0* ∨ *poly q x* ≠ *0*
**shows** *rsquarefree*(*p* ∗ *q*)
⟨*proof*⟩

## 1.3 On Symmetric Differences

**lemma** *card-sym-diff-finite*:
**assumes** *finite A finite B*
**shows** *card* (*sym-diff A B*) = *card* (*A*−*B*) + *card* (*B*−*A*)
⟨*proof*⟩

**lemma** *card-add-diff-finite*:
**assumes** *finite A finite B*
**shows** *card A + card (B−A) = card B + card (A−B)*
⟨*proof*⟩

**lemma** *card-sub-int-diff-finite*:
**assumes** *finite A finite B*
**shows** *int (card A) − card B = int (card (A−B)) − card (B−A)*
⟨*proof*⟩

**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)*
⟨*proof*⟩

## 1.4 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*⟩

**lemma** *in-set-to-poly*: *poly* (*set-to-poly A*) *x* = *0* ⟷ *x* ∈ *A*
⟨*proof*⟩

**lemma** *set-to-poly-not0*[*simp*]: *set-to-poly A* ≠ *0*
⟨*proof*⟩

**lemma** *set-to-poly-empty*[*simp*]: *set-to-poly* {} = *1*
⟨*proof*⟩

**lemma** *set-to-poly-inj*: *inj set-to-poly*
⟨*proof*⟩

**lemma** *rsquarefree-set-to-poly*: *rsquarefree* (*set-to-poly A*)
⟨*proof*⟩

**lemma** *set-to-poly-insert*:
**assumes***x* ∉ *A*
**shows** *set-to-poly* (*insert x A*) = *set-to-poly A* ∗ [:−*x*,*1*:]
⟨*proof*⟩

**lemma** *set-to-poly-mult*: *set-to-poly X* ∗ *set-to-poly Y* = *set-to-poly* (*X* ∪ *Y*) ∗ *set-to-poly* (*X* ∩ *Y*)
⟨*proof*⟩

**lemma** *set-to-poly-mult-distinct*:

**assumes** $X \cap Y = \{\}$
**shows** *set-to-poly* $X * $ *set-to-poly* $Y =$ *set-to-poly* $(X \cup Y)$
⟨*proof*⟩

**lemma** *set-to-poly-degree*:
*degree* (*set-to-poly* $A$) = *card* $A$
⟨*proof*⟩

**lemma** *set-to-poly-order*:
*order* $x$ (*set-to-poly* $A$) = (*if* $x \in A$ *then 1 else 0*)
⟨*proof*⟩

**lemma** *set-to-poly-lead-coeff*: *lead-coeff* (*set-to-poly* $A$) = *1*
⟨*proof*⟩

**lemma** *degree-sub-lead-coeff*:
**assumes** *degree* $p > 0$
**shows** *degree* $(p -$ *monom* (*lead-coeff* $p$) (*degree* $p)) <$ *degree* $p$
⟨*proof*⟩

**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* $p$
⟨*proof*⟩

**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*⟩

**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* $(0::'a::zero\text{-}neq\text{-}one\ poly)$
⟨*proof*⟩

**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* $p_A$ $p_B$ $E$ $f_A$ $f_B$ $d_A$ $d_B \vee \neg(p_A \neq 0 \wedge p_B \neq 0)$
⟨*proof*⟩

**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*
$(E\ !\ i)\ \hat{}\ j$
*else if* $j < d_A + d_B$ *then*
$-\ f\ (E\ !\ i) * (E\ !\ i)\ \hat{}\ (j - d_A)$
*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* $E$ $f$ $d_A$ $d_B$ = $(\lambda\ i.\ f\ (E\ !\ i) * (E\ !\ i)\ \hat{}\ d_B - (E\ !\ i)\ \hat{}\ d_A)$

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

**definition** *solution-to-poly* :: $('a::finite\text{-}field,\ 'n::mod\text{-}type)\ vec \Rightarrow nat \Rightarrow nat \Rightarrow 'a\ poly \times 'a\ poly$ **where**

*solution-to-poly S* $d_A$ $d_B$ = (*let*
*p* = *Abs-poly* (λ*i*. *if i* < $d_A$ *then S* $ (*from-nat i*) *else 0*) + *monom 1* $d_A$;
*q* = *Abs-poly* (λ*i*. *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* ≥ *n*
**shows** (∑ *i* = *0*..*n*. *f i*) + (∑ *i* = *Suc n* ..*m*. *f i*) = (∑ *i* = *0*..*m*. *f i*)
⟨*proof*⟩

**lemma** *poly-altdef-Abs-poly-le*:
**fixes** *x* :: *'a*::{*comm-semiring-0*, *semiring-1*}
**shows** *poly* (*Abs-poly* (λ*i*. *if i* ≤ *n then f i else 0*)) *x* = (∑ *i* = *0*..*n*. *f i* ∗ *x* ^ *i*)
⟨*proof*⟩

**lemma** *poly-altdef-Abs-poly-l*:
**fixes** *x* :: *'a*::{*comm-semiring-0*,*semiring-1*}
**shows** *poly* (*Abs-poly* (λ*i*. *if i* < *n then f i else 0*)) *x* = (∑ *i*<*n*. *f i* ∗ *x* ^ *i*)
⟨*proof*⟩

**lemma** *degree-Abs-poly-If-l*:
**assumes** *n* ≠ *0*
**shows** *degree* (*Abs-poly* (λ*i*. *if i* < *n then f i else 0*)) < *n*
⟨*proof*⟩

**lemma** *nth-less-length-in-set-eq*:
**shows** (∀ *i* < *length E*. *f* (*E* ! *i*) = *g* (*E* ! *i*)) ⟷ (∀ *e* ∈ *set E*. *f e* = *g e*)
⟨*proof*⟩

**lemma** *nat-leq-real-floor*: *real* (*i*::*nat*) ≤ (*d*::*real*) ⟷ *real i* ≤ ⌊*d*⌋ (**is** *?l* = *?r*)
⟨*proof*⟩

**lemma** *mod-type-less-function-eq*:
**fixes** *i* :: *'a*::*mod-type*
**assumes** ∀ *i* < *CARD*(*'a*) . *f i* = *g i*
**shows** *f* (*to-nat i*) = *g* (*to-nat i*)
⟨*proof*⟩

## 2.3 On *solution-to-poly*

**lemma** *fst-solution-to-poly-nz*:
*fst* (*solution-to-poly S* $d_A$ $d_B$) ≠ *0*
⟨*proof*⟩

**lemma** *snd-solution-to-poly-nz*:
*snd* (*solution-to-poly S* $d_A$ $d_B$) $\neq$ *0*
⟨*proof*⟩

**lemma** *degree-Abs0p1*: *degree* (*Abs-poly* (λ*i*. *0*) + *1*) = *0*
⟨*proof*⟩

**lemma** *degree-solution-to-poly-fst*:
*degree* (*fst* (*solution-to-poly S* $d_A$ $d_B$)) = $d_A$
⟨*proof*⟩

**lemma** *degree-solution-to-poly-snd*:
*degree* (*snd* (*solution-to-poly S* $d_A$ $d_B$)) = $d_B$
⟨*proof*⟩

**lemma** *monic-solution-to-poly-snd*:
*monic* (*snd* (*solution-to-poly S* $d_A$ $d_B$))
⟨*proof*⟩

**lemma** *monic-solution-to-poly-fst*:
*monic* (*fst* (*solution-to-poly S* $d_A$ $d_B$))
⟨*proof*⟩

## 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*⟩

**lemma** *poly-lead-coeff-extract*:
*poly p x* = (∑ *i*<*degree p*. *coeff p i* ∗ *x* ^ *i*) + *lead-coeff p* ∗ *x* ^ *degree p*
**for** *x* :: '*a*::{*comm-semiring-0*,*semiring-1*}
⟨*proof*⟩

**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* $d_B = \lfloor (real\ (length\ E) + card\ B - card\ A)/2 \rfloor$
$card\ (sym\text{-}diff\ A\ B) \leq length\ E$
**shows**
$d_A + d_B \leq length\ E$
$card\ (A - B) \leq d_A\ card\ (B - A) \leq d_B$
$d_B - card\ (B - A) = d_A - card\ (A - B)$
⟨*proof*⟩

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**
$\forall\ x \in (set\ E).\ f\ x = f_A\ x\ /\ f_B\ x$
$CARD('m$::*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*⟩

## 2.5 Main lemma

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

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

**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* $p_A$ $p_B$ $E$ $f_A$ $f_B$ $\lfloor d_A \rfloor$ $\lfloor d_B \rfloor$
⟨*proof*⟩

**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*⟩

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* $A$ $B$) $\leq$ *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-to-poly* $A$) $x)$
$f_B = (\lambda\ x \in$ *set* $E$. *poly* (*set-to-poly* $B$) $x)$
*CARD*($'m$::*mod-type*) = *length* $E$
**defines** $d'_A \equiv$ (*real* (*length* $E$) + *card* $A$ − *card* $B$)/2
**defines** $d'_B \equiv$ (*real* (*length* $E$) + *card* $B$ − *card* $A$)/2
**defines** $d_A \equiv$ *nat* $\lfloor d'_A \rfloor$
**defines** $d_B \equiv$ *nat* $\lfloor d'_B \rfloor$
**defines** *sol-poly* ≡ *interpolate-rat-fun* $E$ ($\lambda e.\ f_A\ e$ / $f_B\ e$) $d_A$ $d_B$ *TYPE*($'m$)
**shows**
*monic-interpolated-rational-function* (*fst sol-poly*) (*snd sol-poly*) (*set* $E$) $f_A$ $f_B$ $d'_A$ $d'_B$
⟨*proof*⟩

**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*⟩

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*
⟨*proof*⟩

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*⟩

**lemma** *extract-root*:
**assumes** *monic p irreducible p*
**shows** *extract-root p = proots p*
⟨*proof*⟩

**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*⟩

**lemma** *proots-extract-roots-factorized*:
**assumes** *squarefree p*
**shows** *proots p = extract-roots (snd (finite-field-factorization p))*
⟨*proof*⟩

## 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)*

⟨*proof*⟩

**lemma** *degree-zero-iff-no-factors*:
**fixes** $p :: 'a :: \{factorial\text{-}ring\text{-}gcd, semiring\text{-}gcd\text{-}mult\text{-}normalize, field\}\ poly$
**assumes** $p \neq 0$
**shows** $prime\text{-}factors\ p = \{\} \longleftrightarrow degree\ p = 0$
⟨*proof*⟩

**lemma** *ERF'-termination*:
**assumes** $degree\ p > 0$
**shows** $degree\ (p\ div\ prod\text{-}list\ (ERF\ p)) < degree\ p$
⟨*proof*⟩

**termination**
⟨*proof*⟩

**lemma** *ERF'-squarefree*:
**assumes** $x \in set\ (ERF'\ p)$
**shows** *squarefree x* ⟨*proof*⟩

**lemma** *ERF-not0*: $p \neq 0 \Longrightarrow 0 \notin set\ (ERF\ p)$
⟨*proof*⟩

**lemma** *ERF'-not0*: $0 \notin set\ (ERF'\ p)$
⟨*proof*⟩

**lemma** *ERF'-proots*: $proots\ (\prod x \leftarrow ERF'\ p.\ x) = proots\ p$
⟨*proof*⟩

## 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*⟩

## 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*]: $p \neq 0 \Longrightarrow order\ x\ p = order\text{-}eff\ x\ p$
⟨*proof*⟩

**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*, $\lambda$ $x \in$ *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 (λx. 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:

$$\begin{aligned} p_A &:= \chi_{A-B}(x)x^r \\ p_B &:= \chi_{B-A}(x)x^r \end{aligned}$$

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 $\mathrm{ord}_x$), we have:

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

which implies:

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

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

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

## 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**
$\forall\ x \in set\ E.\ poly\ (set\text{-}to\text{-}poly\ A)\ x * poly\ p_B\ x = poly\ (set\text{-}to\text{-}poly\ B)\ x * poly\ p_A\ x$
$degree\ p_A \leq (real\ (length\ E) + card\ A - card\ B)/2$
$degree\ p_B \leq (real\ (length\ E) + card\ B - card\ A)/2$
$card\ (sym\text{-}diff\ A\ B) < length\ E$
$set\ E \cap A = \{\}\ set\ E \cap B = \{\}$
**shows** $set\text{-}to\text{-}poly\ (A-B) * p_B = set\text{-}to\text{-}poly\ (B-A) * p_A$
⟨*proof*⟩

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**
$\forall\ x \in set\ E.\ poly\ (set\text{-}to\text{-}poly\ A)\ x * poly\ p_B\ x = poly\ (set\text{-}to\text{-}poly\ B)\ x * poly\ p_A\ x$
$degree\ p_A = \lfloor (real\ (length\ E) + card\ A - card\ B)/2 \rfloor$
$degree\ p_B = \lfloor (real\ (length\ E) + card\ B - card\ A)/2 \rfloor$
$card\ (sym\text{-}diff\ A\ B) \leq length\ E$
$set\ E \cap A = \{\}\ set\ E \cap B = \{\}$
$monic\ p_A\ monic\ p_B$
**shows** $set\text{-}to\text{-}poly\ (A-B) * p_B = set\text{-}to\text{-}poly\ (B-A) * p_A$ (**is** *?lhs = ?rhs*)
⟨*proof*⟩

### 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*⟩

**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.