

A Set Reconciliation Algorithm

Paul Hofmeier and Emin Karayel

December 27, 2025

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	2
1.1	On Polynomial Roots	2
1.2	On <i>rsquarefree</i>	2
1.3	On Symmetric Differences	2
1.4	Characteristic Polynomial	3
2	Rational Function Interpolation	4
2.1	Definitions	5
2.2	Preliminary Results	6
2.3	On <i>solution-to-poly</i>	6
2.4	Correctness	7
2.5	Main lemma	8
3	Factorisation of Polynomials	9
3.1	Elimination of Repeated Factors	10
3.2	Executable version of <i>proots</i>	11
3.3	Executable version of <i>order</i>	11
4	Set Reconciliation Algorithm	12
4.1	Informal Description of the Algorithm	13
4.2	Lemmas	14
4.3	Main Result	15

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]*: $\text{order } x \text{ } [-y, 1:] = (\text{if } x=y \text{ then } 1 \text{ else } 0)$
 $\langle \text{proof} \rangle$

1.1 On Polynomial Roots

lemma *proots-empty*: $\text{proots } p = \{\#\} \longleftrightarrow p = 0 \vee (\forall x. \text{poly } p \ x \neq 0)$
 $\langle \text{proof} \rangle$

lemma *proots-element*: $x \in \# \text{ proots } p \vee p = 0 \longleftrightarrow \text{poly } p \ x = 0$
 $\langle \text{proof} \rangle$

lemma *proots-diff*:

assumes $p \neq 0 \ q \neq 0$

shows $\text{set-mset } (\text{proots } p - \text{proots } q) = \{x. \text{order } x \ p > \text{order } x \ q\}$ (**is** $?L = ?R$)
 $\langle \text{proof} \rangle$

1.2 On *rsquarefree*

The following fact is an improved version of $\llbracket \text{rsquarefree } ?p; \text{poly } ?p \ ?z = 0; ?p \neq 0 \rrbracket \implies \text{order } ?z \ ?p = 1$, which does not require the assumption that $p \neq 0$.

lemma *rsquarefree-root-order'*: $\text{rsquarefree } p \implies \text{poly } p \ x = 0 \implies \text{order } x \ p = 1$
 $\langle \text{proof} \rangle$

lemma *rsquarefree-single-root[simp]*: $\text{rsquarefree } [-x, 1:]$
 $\langle \text{proof} \rangle$

lemma *rsquarefree-mul*:

assumes $\text{rsquarefree } p \ \text{rsquarefree } q$

$\forall x. \text{poly } p \ x \neq 0 \vee \text{poly } q \ x \neq 0$

shows $\text{rsquarefree}(p * q)$
 $\langle \text{proof} \rangle$

1.3 On Symmetric Differences

lemma *card-sym-diff-finite*:

assumes $\text{finite } A \ \text{finite } B$

shows $\text{card } (\text{sym-diff } A \ B) = \text{card } (A - B) + \text{card } (B - A)$
 $\langle \text{proof} \rangle$

lemma *card-add-diff-finite*:
assumes *finite A finite B*
shows $\text{card } A + \text{card } (B-A) = \text{card } B + \text{card } (A-B)$
 $\langle \text{proof} \rangle$

lemma *card-sub-int-diff-finite*:
assumes *finite A finite B*
shows $\text{int } (\text{card } A) - \text{card } B = \text{int } (\text{card } (A-B)) - \text{card } (B-A)$
 $\langle \text{proof} \rangle$

lemma *card-sub-int-diff-finite-real*:
assumes *finite A finite B*
shows $\text{real } (\text{card } A) - \text{card } B = \text{real } (\text{card } (A-B)) - \text{card } (B-A)$
 $\langle \text{proof} \rangle$

1.4 Characteristic Polynomial

The characteristic polynomial associated to a set:

definition *set-to-poly* :: *'a::finite-field set \Rightarrow 'a poly* **where**
 $\text{set-to-poly } A \equiv \prod_{a \in A} [-a, 1:]$

lemma *set-to-poly-correct*: $\{x. \text{poly } (\text{set-to-poly } A) \ x = 0\} = A$
 $\langle \text{proof} \rangle$

lemma *in-set-to-poly*: $\text{poly } (\text{set-to-poly } A) \ x = 0 \longleftrightarrow x \in A$
 $\langle \text{proof} \rangle$

lemma *set-to-poly-not0[simp]*: $\text{set-to-poly } A \neq 0$
 $\langle \text{proof} \rangle$

lemma *set-to-poly-empty[simp]*: $\text{set-to-poly } \{\} = 1$
 $\langle \text{proof} \rangle$

lemma *set-to-poly-inj*: *inj set-to-poly*
 $\langle \text{proof} \rangle$

lemma *rsquarefree-set-to-poly*: *rsquarefree (set-to-poly A)*
 $\langle \text{proof} \rangle$

lemma *set-to-poly-insert*:
assumes $x \notin A$
shows $\text{set-to-poly } (\text{insert } x \ A) = \text{set-to-poly } A * [-x, 1:]$
 $\langle \text{proof} \rangle$

lemma *set-to-poly-mult*: $\text{set-to-poly } X * \text{set-to-poly } Y = \text{set-to-poly } (X \cup Y) * \text{set-to-poly } (X \cap Y)$
 $\langle \text{proof} \rangle$

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

```

assumes  $X \cap Y = \{\}$ 
shows  $\text{set-to-poly } X * \text{set-to-poly } Y = \text{set-to-poly } (X \cup Y)$ 
 $\langle \text{proof} \rangle$ 

lemma set-to-poly-degree:
   $\text{degree } (\text{set-to-poly } A) = \text{card } A$ 
 $\langle \text{proof} \rangle$ 

lemma set-to-poly-order:
   $\text{order } x \ (\text{set-to-poly } A) = (\text{if } x \in A \text{ then } 1 \text{ else } 0)$ 
 $\langle \text{proof} \rangle$ 

lemma set-to-poly-lead-coeff:  $\text{lead-coeff } (\text{set-to-poly } A) = 1$ 
 $\langle \text{proof} \rangle$ 

lemma degree-sub-lead-coeff:
  assumes  $\text{degree } p > 0$ 
  shows  $\text{degree } (p - \text{monom } (\text{lead-coeff } p) \ (\text{degree } p)) < \text{degree } p$ 
 $\langle \text{proof} \rangle$ 

lemma remove-lead-from-monic:
  fixes  $p \ q :: 'a :: \text{field poly}$ 
  assumes monic  $p$ 
  assumes  $\text{degree } p > 0$ 
  shows  $\text{degree } (p - \text{monom } 1 \ (\text{degree } p)) < \text{degree } p$ 
 $\langle \text{proof} \rangle$ 

lemma poly-eqI-degree-monic:
  fixes  $p \ q :: 'a :: \text{field poly}$ 
  assumes  $\text{degree } p = \text{degree } q$ 
  assumes  $\text{degree } p \leq \text{card } A$ 
  assumes monic  $p$  monic  $q$ 
  assumes  $\bigwedge x. x \in A \implies \text{poly } p \ x = \text{poly } q \ x$ 
  shows  $p = q$ 
 $\langle \text{proof} \rangle$ 

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 $p_A p_B E f_A f_B d_A d_B \equiv$
 $(\forall e \in E. f_A e * \text{poly } p_B e = f_B e * \text{poly } p_A e) \wedge$
 $\text{degree } p_A \leq (d_A::\text{real}) \wedge \text{degree } p_B \leq (d_B::\text{real}) \wedge$
 $p_A \neq 0 \wedge p_B \neq 0$

Interpolation condition with given exact degrees

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

lemma *monic0*: $\neg \text{monic } (0::'a::\text{zero-neq-one poly})$
 $\langle \text{proof} \rangle$

lemma *monic-interpolated-rational-function-interpolated-rational-function*:
monic-interpolated-rational-function $p_A p_B E f_A f_B d_A d_B$
 $\implies \text{interpolated-rational-function } p_A p_B E f_A f_B d_A d_B \vee \neg(p_A \neq 0 \wedge p_B \neq 0)$
 $\langle \text{proof} \rangle$

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

definition *rfi-constant-vector* $:: 'a::\text{field list} \Rightarrow ('a \Rightarrow 'a) \Rightarrow \text{nat} \Rightarrow \text{nat} \Rightarrow (\text{nat} \Rightarrow 'a)$ **where**
rfi-constant-vector $E f d_A d_B = (\lambda i. f (E ! i) * (E ! i) \wedge d_B - (E ! i) \wedge d_A)$

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

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

$\text{solution-to-poly } S \ d_A \ d_B = (\text{let}$
 $\quad p = \text{Abs-poly } (\lambda i. \text{ if } i < d_A \text{ then } S \ \$ (\text{from-nat } i) \text{ else } 0) + \text{monom } 1 \ d_A;$
 $\quad q = \text{Abs-poly } (\lambda i. \text{ if } i < d_B \text{ then } S \ \$ (\text{from-nat } (i+d_A)) \text{ else } 0) + \text{monom } 1$
 $\quad d_B \text{ in}$
 $\quad (p, q))$

definition *interpolate-rat-fun* **where**

$\text{interpolate-rat-fun } E \ f \ d_A \ d_B \ m =$
 $\text{solution-to-poly } (\text{rational-function-interpolation } E \ f \ d_A \ d_B \ m) \ d_A \ d_B$

2.2 Preliminary Results

lemma *consecutive-sum-combine*:

assumes $m \geq n$
shows $(\sum i = 0..n. f \ i) + (\sum i = \text{Suc } n ..m. f \ i) = (\sum i = 0..m. f \ i)$
 $\langle \text{proof} \rangle$

lemma *poly-altdef-Abs-poly-le*:

fixes $x :: 'a :: \{\text{comm-semiring-0, semiring-1}\}$
shows $\text{poly } (\text{Abs-poly } (\lambda i. \text{ if } i \leq n \text{ then } f \ i \text{ else } 0)) \ x = (\sum i = 0..n. f \ i * x ^ i)$
 $\langle \text{proof} \rangle$

lemma *poly-altdef-Abs-poly-l*:

fixes $x :: 'a :: \{\text{comm-semiring-0, semiring-1}\}$
shows $\text{poly } (\text{Abs-poly } (\lambda i. \text{ if } i < n \text{ then } f \ i \text{ else } 0)) \ x = (\sum i < n. f \ i * x ^ i)$
 $\langle \text{proof} \rangle$

lemma *degree-Abs-poly-If-l*:

assumes $n \neq 0$
shows $\text{degree } (\text{Abs-poly } (\lambda i. \text{ if } i < n \text{ then } f \ i \text{ else } 0)) < n$
 $\langle \text{proof} \rangle$

lemma *nth-less-length-in-set-eq*:

shows $(\forall i < \text{length } E. f \ (E ! i) = g \ (E ! i)) \longleftrightarrow (\forall e \in \text{set } E. f \ e = g \ e)$
 $\langle \text{proof} \rangle$

lemma *nat-leq-real-floor*: $\text{real } (i::\text{nat}) \leq (d::\text{real}) \longleftrightarrow \text{real } i \leq \lfloor d \rfloor$ (**is** $?l = ?r$)

$\langle \text{proof} \rangle$

lemma *mod-type-less-function-eq*:

fixes $i :: 'a :: \text{mod-type}$
assumes $\forall i < \text{CARD}('a). f \ i = g \ i$
shows $f \ (\text{to-nat } i) = g \ (\text{to-nat } i)$
 $\langle \text{proof} \rangle$

2.3 On solution-to-poly

lemma *fst-solution-to-poly-nz*:

$\text{fst } (\text{solution-to-poly } S \ d_A \ d_B) \neq 0$
 $\langle \text{proof} \rangle$

lemma *snd-solution-to-poly-nz*:
 $\text{snd } (\text{solution-to-poly } S \ d_A \ d_B) \neq 0$
 $\langle \text{proof} \rangle$

lemma *degree-Abs0p1*: $\text{degree } (\text{Abs-poly } (\lambda i. 0) + 1) = 0$
 $\langle \text{proof} \rangle$

lemma *degree-solution-to-poly-fst*:
 $\text{degree } (\text{fst } (\text{solution-to-poly } S \ d_A \ d_B)) = d_A$
 $\langle \text{proof} \rangle$

lemma *degree-solution-to-poly-snd*:
 $\text{degree } (\text{snd } (\text{solution-to-poly } S \ d_A \ d_B)) = d_B$
 $\langle \text{proof} \rangle$

lemma *monic-solution-to-poly-snd*:
 $\text{monic } (\text{snd } (\text{solution-to-poly } S \ d_A \ d_B))$
 $\langle \text{proof} \rangle$

lemma *monic-solution-to-poly-fst*:
 $\text{monic } (\text{fst } (\text{solution-to-poly } S \ d_A \ d_B))$
 $\langle \text{proof} \rangle$

2.4 Correctness

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

lemma *rational-function-interpolation-correct-poly*:
assumes
 $\forall x \in \text{set } E. f \ x = f_A \ x / f_B \ x \ \forall x \in \text{set } E. f_B \ x \neq 0$
 $d_A + d_B \leq \text{length } E$
 $\text{CARD}('m::\text{mod-type}) = \text{length } E$
 $\text{consistent } (\chi \ (i::'m) \ (j::'m). \text{rfi-coefficient-matrix } E \ f \ d_A \ d_B \ (\text{to-nat } i) \ (\text{to-nat } j))$
 $(\chi \ (i::'m). \text{rfi-constant-vector } E \ f \ d_A \ d_B \ (\text{to-nat } i))$
 $S = \text{rational-function-interpolation } E \ f \ d_A \ d_B \ \text{TYPE}('m)$
 $p_A = \text{fst } (\text{solution-to-poly } S \ d_A \ d_B)$
 $p_B = \text{snd } (\text{solution-to-poly } S \ d_A \ d_B)$
shows
 $\forall e \in \text{set } E. f_A \ e * \text{poly } p_B \ e = f_B \ e * \text{poly } p_A \ e$
 $\langle \text{proof} \rangle$

lemma *poly-lead-coeff-extract*:
 $\text{poly } p \ x = (\sum i < \text{degree } p. \text{coeff } p \ i * x^i) + \text{lead-coeff } p * x^{\text{degree } p}$
for $x :: 'a::\{\text{comm-semiring-0}, \text{semiring-1}\}$
 $\langle \text{proof} \rangle$

lemma *d_A-d_B-helper*:
assumes

$finite\ A\ finite\ B$
 $int\ d_A = \lfloor (real\ (length\ E) + card\ A - card\ B)/2 \rfloor$
 $int\ d_B = \lfloor (real\ (length\ E) + card\ B - card\ A)/2 \rfloor$
 $card\ (sym-diff\ A\ B) \leq length\ E$
shows
 $d_A + d_B \leq length\ E$
 $card\ (A - B) \leq d_A\ card\ (B - A) \leq d_B$
 $d_B - card\ (B - A) = d_A - card\ (A - B)$
 $\langle proof \rangle$

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

$\forall\ x \in (set\ E). f\ x = f_A\ x / f_B\ x$
 $CARD('m::mod-type) = length\ E$
 $d_A + d_B \leq length\ E$
 $card\ (A - B) \leq d_A$
 $card\ (B - A) \leq d_B$
 $d_B - card\ (B - A) = d_A - card\ (A - B)$
 $\forall\ x \in set\ E. x \notin A\ \forall\ x \in set\ E. x \notin B$
 $f_A = (\lambda\ x \in set\ E. poly\ (set-to-poly\ A)\ x)$
 $f_B = (\lambda\ x \in set\ E. poly\ (set-to-poly\ B)\ x)$

shows

$consistent\ (\chi\ (i::'m)\ (j::'m). rfi-coefficient-matrix\ E\ f\ d_A\ d_B\ (to-nat\ i)\ (to-nat\ j))$
 $(\chi\ (i::'m). rfi-constant-vector\ E\ f\ d_A\ d_B\ (to-nat\ i))$
 $\langle proof \rangle$

2.5 Main lemma

lemma *rational-function-interpolation-correct:*

assumes

$int\ d_A = \lfloor (real\ (length\ E) + card\ A - card\ B)/2 \rfloor$
 $int\ d_B = \lfloor (real\ (length\ E) + card\ B - card\ A)/2 \rfloor$
 $card\ (sym-diff\ A\ B) \leq length\ E$

$\forall\ x \in set\ E. x \notin A\ \forall\ x \in set\ E. x \notin B$
 $f_A = (\lambda\ x \in set\ E. poly\ (set-to-poly\ A)\ x)$
 $f_B = (\lambda\ x \in set\ E. poly\ (set-to-poly\ B)\ x)$
 $CARD('m::mod-type) = length\ E$

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

$monic-interpolated-rational-function\ (fst\ sol)\ (snd\ sol)\ (set\ E)\ f_A\ f_B\ d_A\ d_B$
 $\langle proof \rangle$

lemma *interpolated-rational-function-floor-eq:*

$\text{interpolated-rational-function } p_A \ p_B \ E \ f_A \ f_B \ d_A \ d_B \longleftrightarrow$
 $\text{interpolated-rational-function } p_A \ p_B \ E \ f_A \ f_B \ \lfloor d_A \rfloor \ \lfloor d_B \rfloor$
 $\langle \text{proof} \rangle$

lemma *sym-diff-bound-div2-ge0*:
fixes $A \ B :: 'a :: \text{finite set}$
assumes $\text{card } (\text{sym-diff } A \ B) \leq \text{length } E$
shows $(\text{real } (\text{length } E) + \text{card } A - \text{card } B) / 2 \geq 0$
 $\langle \text{proof} \rangle$

If the degrees are reals we take the floor first

lemma *rational-function-interpolation-correct-real*:
fixes $d'_A \ d'_B :: \text{real}$
assumes
 $\text{card } (\text{sym-diff } A \ B) \leq \text{length } E$
 $\forall x \in \text{set } E. x \notin A \ \forall x \in \text{set } E. x \notin B$
 $f_A = (\lambda x \in \text{set } E. \text{poly } (\text{set-to-poly } A) \ x)$
 $f_B = (\lambda x \in \text{set } E. \text{poly } (\text{set-to-poly } B) \ x)$
 $\text{CARD}('m :: \text{mod-type}) = \text{length } E$
defines $d'_A \equiv (\text{real } (\text{length } E) + \text{card } A - \text{card } B) / 2$
defines $d'_B \equiv (\text{real } (\text{length } E) + \text{card } B - \text{card } A) / 2$
defines $d_A \equiv \text{nat } \lfloor d'_A \rfloor$
defines $d_B \equiv \text{nat } \lfloor d'_B \rfloor$
defines $\text{sol-poly} \equiv \text{interpolate-rat-fun } E \ (\lambda e. f_A \ e \ / \ f_B \ e) \ d_A \ d_B \ \text{TYPE}('m)$
shows
 $\text{monic-interpolated-rational-function } (\text{fst } \text{sol-poly}) \ (\text{snd } \text{sol-poly}) \ (\text{set } E) \ f_A \ f_B$
 $d'_A \ d'_B$
 $\langle \text{proof} \rangle$

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
 $\text{square-free } f$

```

    finite-field-factorization f = (c, us)
  shows roots f = sum-list (map roots us)
<proof>

```

The following fact is an improved version of $?x \neq 0 \implies \text{squarefree } ?x = \text{square-free } ?x$, which does not require the assumption that $p \neq 0$.

```

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  $\Rightarrow$  '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 = roots p
<proof>

```

```

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

```

```

lemma extract-roots:
   $\forall p \in \text{set } ps. \text{monic } p \wedge \text{irreducible } p \implies$ 
  sum-list (map roots ps) = extract-roots ps
<proof>

```

```

lemma roots-extract-roots-factorized:
  assumes squarefree p
  shows roots 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)

```

$\langle \text{proof} \rangle$

lemma *degree-zero-iff-no-factors*:

fixes $p :: 'a :: \{\text{factorial-ring-gcd}, \text{semiring-gcd-mult-normalize}, \text{field}\}$ *poly*

assumes $p \neq 0$

shows $\text{prime-factors } p = \{\} \iff \text{degree } p = 0$

$\langle \text{proof} \rangle$

lemma *ERF'-termination*:

assumes $\text{degree } p > 0$

shows $\text{degree } (p \text{ div } \text{prod-list } (\text{ERF } p)) < \text{degree } p$

$\langle \text{proof} \rangle$

termination

$\langle \text{proof} \rangle$

lemma *ERF'-squarefree*:

assumes $x \in \text{set } (\text{ERF}' p)$

shows $\text{squarefree } x$ $\langle \text{proof} \rangle$

lemma *ERF-not0*: $p \neq 0 \implies 0 \notin \text{set } (\text{ERF } p)$

$\langle \text{proof} \rangle$

lemma *ERF'-not0*: $0 \notin \text{set } (\text{ERF}' p)$

$\langle \text{proof} \rangle$

lemma *ERF'-proots*: $\text{proots } (\prod x \leftarrow \text{ERF}' p. x) = \text{proots } p$

$\langle \text{proof} \rangle$

3.2 Executable version of *proots*

fun *proots-eff* :: $'a::\text{prime-card mod-ring poly} \Rightarrow 'a \text{ mod-ring multiset}$ **where**

$\text{proots-eff } p = \text{sum-list } (\text{map } (\text{extract-roots} \circ \text{snd} \circ \text{finite-field-factorization}) (\text{ERF}' p))$

lemma *proots-eff-correct* [code-unfold]: $\text{proots } p = \text{proots-eff } p$

$\langle \text{proof} \rangle$

3.3 Executable version of *order*

fun *order-eff* :: $'a \text{ mod-ring} \Rightarrow 'a::\text{prime-card mod-ring poly} \Rightarrow \text{nat}$ **where**

$\text{order-eff } x p = \text{count } (\text{proots-eff } p) x$

lemma *order-eff-code* [code-unfold]: $p \neq 0 \implies \text{order } x p = \text{order-eff } x p$

$\langle \text{proof} \rangle$

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 chosen 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  $E :: 'a :: \text{prime-card mod-ring list}$ 
  fixes  $\text{phantom-}m :: 'm::\text{mod-type itself}$ 
  assumes  $\text{type-}m: \text{phantom-}m = \text{TYPE}('m)$ 
  assumes  $\text{distinct-}E: \text{distinct } E$ 
  assumes  $\text{card-}m: \text{CARD}('m) = \text{length } E$ 
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
   $\text{encode } A = (\text{card } A, \lambda x \in \text{set } E. \text{poly } (\text{set-to-poly } A) x)$ 

```

definition *decode* where

decode $B \ R =$
 (let
 $(n, f_A) = R;$
 $f_B = (\lambda x \in \text{set } E. \text{poly } (\text{set-to-poly } B) \ x);$
 $d_A = \text{nat } \lfloor (\text{real } (\text{length } E) + n - \text{card } B) / 2 \rfloor;$
 $d_B = \text{nat } \lfloor (\text{real } (\text{length } E) + \text{card } B - n) / 2 \rfloor;$
 $(p_A, p_B) = \text{interpolate-rat-fun } E \ (\lambda x. f_A \ x / f_B \ x) \ d_A \ d_B \ \text{phantom-}m;$
 $r_A = \text{proots-eff } p_A;$
 $r_B = \text{proots-eff } p_B$
 in
 $\text{set-mset } (r_A - r_B) \cup (B - (\text{set-mset } (r_B - r_A)))$)

4.1 Informal Description of the Algorithm

The protocol works as follows:

We associate 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:

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

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:

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

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

$\forall x \in \text{set } E. \text{ poly } (\text{set-to-poly } A) \ x * \text{ poly } p_B \ x = \text{ poly } (\text{set-to-poly } B) \ x * \text{ poly } p_A \ x$

$\text{degree } p_A \leq (\text{real } (\text{length } E) + \text{card } A - \text{card } B) / 2$

$\text{degree } p_B \leq (\text{real } (\text{length } E) + \text{card } B - \text{card } A) / 2$

$\text{card } (\text{sym-diff } A \ B) < \text{length } E$

$\text{set } E \cap A = \{\} \ \text{set } E \cap B = \{\}$

shows $\text{set-to-poly } (A-B) * p_B = \text{set-to-poly } (B-A) * p_A$

<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

$\forall x \in \text{set } E. \text{ poly } (\text{set-to-poly } A) \ x * \text{ poly } p_B \ x = \text{ poly } (\text{set-to-poly } B) \ x * \text{ poly } p_A \ x$

$\text{degree } p_A = \lfloor (\text{real } (\text{length } E) + \text{card } A - \text{card } B) / 2 \rfloor$

$\text{degree } p_B = \lfloor (\text{real } (\text{length } E) + \text{card } B - \text{card } A) / 2 \rfloor$

$\text{card } (\text{sym-diff } A \ B) \leq \text{length } E$

$\text{set } E \cap A = \{\} \ \text{set } E \cap B = \{\}$

$\text{monic } p_A \ \text{monic } p_B$

shows $\text{set-to-poly } (A-B) * p_B = \text{set-to-poly } (B-A) * p_A$ (**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

$\text{card } (\text{sym-diff } A \ B) \leq \text{length } E$

$\text{set } E \cap A = \{\}$ $\text{set } E \cap B = \{\}$

shows $\text{decode } B \ (\text{encode } A) = A$

<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.