

The Sumcheck Protocol

Azucena Garvia, Christoph Sprenger and Jonathan Bootle

April 18, 2024

Abstract

The sumcheck protocol, first introduced in 1992, is an interactive proof which is a key component of many probabilistic proof systems in computational complexity theory and cryptography, some of which have been deployed. We provide a formally verified security analysis of the sumcheck protocol, following a general and modular approach.

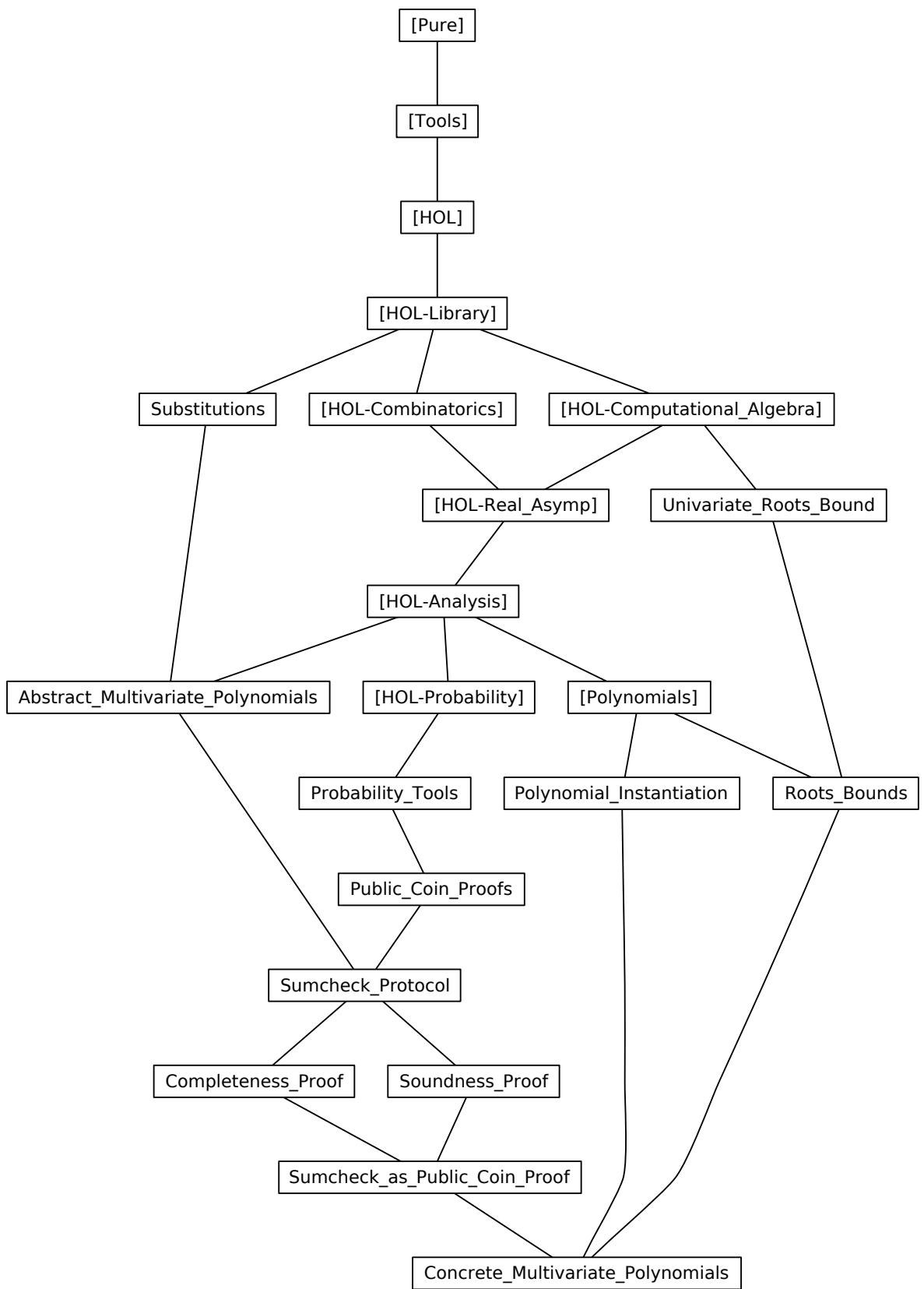
First, we give a general formalization of public-coin interactive proofs. We then define a *generalized sumcheck protocol* for which we axiomatize the underlying mathematical structure and we establish its soundness and completeness. Finally, we prove that these axioms hold for multivariate polynomials, the original setting of the sumcheck protocol. Our modular analysis will facilitate formal verification of sumcheck instances based on different mathematical structures with little effort, by simply proving that these structures satisfy the axioms. Moreover, the analysis will encourage the development and formal verification of future probabilistic proof systems using the sumcheck protocol as a building block.

The paper presenting this formalization is to appear at CSF 2024 under the title “Formal Verification of the Sumcheck Protocol”.

Contents

1	Auxiliary Lemmas Related to Probability Theory	3
1.1	Tuples	3
1.2	Congruence and monotonicity	3
1.3	Some simple derived lemmas	3
1.4	Intersection and union lemmas	4
1.5	Independent probabilities for head and tail of a tuple	4
2	Generic Public-coin Interactive Proofs	7
2.1	Generic definition	7
2.2	Generic soundness and completeness	7
3	Substitutions	9
4	Abstract Multivariate Polynomials	11
4.1	Arity: definition and some lemmas	11
4.2	Lemmas about evaluation, degree, and variables of finite sums	12
4.3	Lemmas combining eval, sum, and inst	13
4.4	Merging sums over substitutions	13

5	Sumcheck Protocol	15
5.1	The sumcheck problem	15
5.2	The sumcheck protocol	15
5.3	The sumcheck protocol as a public-coin proof instance	16
6	Completeness Proof for the Sumcheck Protocol	18
7	Soundness Proof for the Sumcheck Protocol	20
8	Sumcheck Protocol as Public-coin Proof	26
8.1	Property-related definitions	26
8.2	Public coin proof locale interpretation	26
9	Instantiation for Multivariate Polynomials	28
9.1	Instantiation of monomials	28
9.2	Instantiation of polynomials	28
9.3	Full instantiation corresponds to evaluation	29
10	Roots Bound for Univariate Polynomials	30
10.1	Basic lemmas	30
10.2	Univariate roots bound	30
11	Roots Bound for Multivariate Polynomials of Arity at Most One	32
11.1	Lemmas connecting univariate and multivariate polynomials	32
11.1.1	Basic lemmas	32
11.1.2	Total degree corresponds to degree for polynomials of arity at most one	32
11.2	Roots bound for univariate polynomials of type <i>'a mpoly</i>	33
12	Multivariate Polynomials: Instance	34
12.1	Auxiliary lemmas	34
12.2	Proving the assumptions of the locale	34
12.2.1	Variables	34
12.2.2	Degree	34
12.2.3	Evaluation	35
12.2.4	Roots assumption	38
12.3	Locale interpretation	39



3
Figure 1: Theory dependencies

1 Auxiliary Lemmas Related to Probability Theory

```
theory Probability-Tools
  imports HOL-Probability.Probability
begin
```

1.1 Tuples

```
definition tuples :: ‹'a set  $\Rightarrow$  nat  $\Rightarrow$  'a list set› where
  ‹tuples S n = {xs. set xs  $\subseteq$  S  $\wedge$  length xs = n}›
```

```
lemma tuplesI: ‹[ set xs  $\subseteq$  S; length xs = n ]  $\Longrightarrow$  xs  $\in$  tuples S n›
  by (simp add: tuples-def)
```

```
lemma tuplesE [elim]: ‹[ xs  $\in$  tuples S n; [ set xs  $\subseteq$  S; length xs = n ]  $\Longrightarrow$  P ]  $\Longrightarrow$  P›
  by (simp add: tuples-def)
```

```
lemma tuples-Zero: ‹tuples S 0 = {[]}›
  by (auto simp add: tuples-def)
```

```
lemma tuples-Suc: ‹tuples S (Suc n) = ( $\lambda(x, xs). x \# xs$ ) ‘(S  $\times$  tuples S n)›
  by (fastforce simp add: tuples-def image-def Suc-length-conv dest: sym)
```

```
lemma tuples-non-empty [simp]: ‹S  $\neq$  {}  $\Longrightarrow$  tuples S n  $\neq$  {}›
  by (induction n) (auto simp add: tuples-Zero tuples-Suc)
```

```
lemma tuples-finite [simp]: ‹[ finite (S::'a set); S  $\neq$  {} ]  $\Longrightarrow$  finite (tuples S n :: 'a list set)›
  by (auto simp add: tuples-def dest: finite-lists-length-eq)
```

1.2 Congruence and monotonicity

```
lemma prob-cong: — adapted from Joshua
  assumes ‹ $\bigwedge x. x \in \text{set-pmf } M \Longrightarrow x \in A \longleftrightarrow x \in B$ ›
  shows ‹measure-pmf.prob M A = measure-pmf.prob M B›
  using assms
  by (simp add: measure-pmf.finite-measure-eq-AE AE-measure-pmf-iff)
```

```
lemma prob-mono:
  assumes ‹ $\bigwedge x. x \in \text{set-pmf } M \Longrightarrow x \in A \Longrightarrow x \in B$ ›
  shows ‹measure-pmf.prob M A  $\leq$  measure-pmf.prob M B›
  using assms
  by (simp add: measure-pmf.finite-measure-mono-AE AE-measure-pmf-iff)
```

1.3 Some simple derived lemmas

```
lemma prob-empty:
  assumes ‹A = {}›
  shows ‹measure-pmf.prob M A = 0›
  using assms
  by (simp) — uses measure-empty. Sigma-Algebra.measure ?M {} = 0
```

```
lemma prob-pmf-of-set-geq-1:
  assumes finite S and S  $\neq$  {}
```

shows $\text{measure-pmf.prob (pmf-of-set } S) A \geq 1 \iff S \subseteq A$ **using** *assms*
by (*auto simp add: measure-pmf.measure-ge-1-iff measure-pmf.prob-eq-1 AE-measure-pmf-iff*)

1.4 Intersection and union lemmas

lemma *prob-disjoint-union*:

assumes $\langle A \cap B = \{\} \rangle$
shows $\langle \text{measure-pmf.prob } M (A \cup B) = \text{measure-pmf.prob } M A + \text{measure-pmf.prob } M B \rangle$
using *assms*
by (*fact measure-pmf.finite-measure-Union[simplified]*)

lemma *prob-finite-Union*:

assumes $\langle \text{disjoint-family-on } A I \rangle \langle \text{finite } I \rangle$
shows $\langle \text{measure-pmf.prob } M (\bigcup_{i \in I} A i) = (\sum_{i \in I} \text{measure-pmf.prob } M (A i)) \rangle$
using *assms*
by (*intro measure-pmf.finite-measure-finite-Union (simp-all)*)

lemma *prob-disjoint-cases*:

assumes $\langle B \cup C = A \rangle \langle B \cap C = \{\} \rangle$
shows $\langle \text{measure-pmf.prob } M A = \text{measure-pmf.prob } M B + \text{measure-pmf.prob } M C \rangle$
proof –
have $\langle \text{measure-pmf.prob } M A = \text{measure-pmf.prob } M (B \cup C) \rangle$ **using** $\langle B \cup C = A \rangle$
by (*auto intro: prob-cong*)
also have $\langle \dots = \text{measure-pmf.prob } M B + \text{measure-pmf.prob } M C \rangle$ **using** $\langle B \cap C = \{\} \rangle$
by (*simp add: prob-disjoint-union*)
finally show *?thesis* .

qed

lemma *prob-finite-disjoint-cases*:

assumes $\langle (\bigcup_{i \in I} B i) = A \rangle \langle \text{disjoint-family-on } B I \rangle \langle \text{finite } I \rangle$
shows $\langle \text{measure-pmf.prob } M A = (\sum_{i \in I} \text{measure-pmf.prob } M (B i)) \rangle$
proof –
have $\langle \text{measure-pmf.prob } M A = \text{measure-pmf.prob } M (\bigcup_{i \in I} B i) \rangle$ **using** *assms(1)*
by (*auto intro: prob-cong*)
also have $\langle \dots = (\sum_{i \in I} \text{measure-pmf.prob } M (B i)) \rangle$ **using** *assms(2,3)*
by (*intro prob-finite-Union*)
finally show *?thesis* .

qed

1.5 Independent probabilities for head and tail of a tuple

lemma *pmf-of-set-Times*: — by Andreas Lochbihler

$\text{pmf-of-set } (A \times B) = \text{pair-pmf } (\text{pmf-of-set } A) (\text{pmf-of-set } B)$
if *finite* *A* *finite* *B* $A \neq \{\}$ $B \neq \{\}$
by(*rule pmf-eqI*)(*auto simp add: that pmf-pair indicator-def*)

lemma *prob-tuples-hd-tl-indep*:

assumes $\langle S \neq \{\} \rangle$
shows
 $\langle \text{measure-pmf.prob } (\text{pmf-of-set } (\text{tuples } S (\text{Suc } n))) \{ (r::'a::\text{finite}) \# rs \mid r \text{ rs. } P r \wedge Q rs \}$
 $= \text{measure-pmf.prob } (\text{pmf-of-set } (S::'a \text{ set})) \{ r. P r \} * \text{measure-pmf.prob } (\text{pmf-of-set } (\text{tuples } S n)) \{ rs. Q rs \} \rangle$

(is ?lhs = ?rhs)

proof – — mostly by Andreas Lochbihler

Step 1: Split the random variable *pmf-of-set* (*tuples S (Suc n)*) into two independent (*pair-pmf*) random variables, one producing the head and one producing the tail of the list, and then (#) the two random variables using *map-pmf*.

have *: *pmf-of-set* (*tuples S (Suc n)*)
 = *map-pmf* ($\lambda(x :: 'a, xs). x \# xs$) (*pair-pmf* (*pmf-of-set S*) (*pmf-of-set* (*tuples S n*)))
unfolding *tuples-Suc* **using** $\langle S \neq \{\} \rangle$
by (*auto simp add: map-pmf-of-set-inj[symmetric] inj-on-def pmf-of-set-Times*)

Step 2: Transform the event by move the (#) from the random variable into the event. This corresponds to using *distr* on measures.

have ?lhs = *measure-pmf.prob* (*pair-pmf* (*pmf-of-set S*) (*pmf-of-set* (*tuples S n*)))
 ($\lambda(x :: 'a, xs). x \# xs$) – $\{r \# rs \mid r rs. P r \wedge Q rs\}$
unfolding * *measure-map-pmf* **by** (*rule refl*)

Step 3: Rewrite the event as a pair of events. Then we apply independence of the head from the tail.

also have ($\lambda(x, xs). x \# xs$) – $\{r \# rs \mid r rs. P r \wedge Q rs\} = \{r. P r\} \times \{rs. Q rs\}$ **by** *auto*
also have *measure-pmf.prob* (*pair-pmf* (*pmf-of-set S*) (*pmf-of-set* (*tuples S n*))) ... =
measure-pmf.prob (*pmf-of-set S*) $\{r. P r\}$
 * *measure-pmf.prob* (*pmf-of-set* (*tuples S n*)) $\{rs. Q rs\}$
by(*rule measure-pmf-prob-product simp-all*)

finally show ?thesis .

qed

lemma *prob-tuples-fixed-hd*:

$\langle \text{measure-pmf.prob } (\text{pmf-of-set } (\text{tuples UNIV } (\text{Suc } n))) \{rs :: 'a \text{ list}. P rs\}$
 = $(\sum a \in \text{UNIV}. \text{measure-pmf.prob } (\text{pmf-of-set } (\text{tuples UNIV } n)) \{rs. P (a \# rs)\}) / \text{real}(\text{CARD}('a :: \text{finite})) \rangle$
 (is ?lhs = ?rhs)

proof –

{
fix *a*
have $\langle \text{measure-pmf.prob } (\text{pmf-of-set } (\text{tuples UNIV } (\text{Suc } n))) (\{rs. P rs\} \cap \{rs. \text{hd } rs = a\})$
 = $\text{measure-pmf.prob } (\text{pmf-of-set } (\text{tuples UNIV } (\text{Suc } n))) (\{r \# rs \mid r rs. r = a \wedge P (a \# rs)\}) \rangle$
by (*intro prob-cong*) (*auto simp add: tuples-Suc*)
also have $\langle \dots = \text{measure-pmf.prob } (\text{pmf-of-set } (\text{UNIV} :: 'a \text{ set})) \{r. r = a\} * \text{measure-pmf.prob } (\text{pmf-of-set } (\text{tuples UNIV } n)) \{rs. P (a \# rs)\} \rangle$
by (*intro prob-tuples-hd-tl-indep simp*)
also have $\langle \dots = \text{measure-pmf.prob } (\text{pmf-of-set } (\text{tuples UNIV } n)) \{rs. P (a \# rs)\} / \text{real} (\text{CARD} ('a)) \rangle$
by (*simp add: measure-pmf-single*)
finally
have $\langle \text{measure-pmf.prob } (\text{pmf-of-set } (\text{tuples UNIV } (\text{Suc } n))) (\{rs. P rs\} \cap \{rs. \text{hd } rs = a\})$
 = $\text{measure-pmf.prob } (\text{pmf-of-set } (\text{tuples UNIV } n)) \{rs. P (a \# rs)\} / \text{real} (\text{CARD} ('a)) \rangle$.
 }

note *A1 = this*

have $\langle ?lhs = (\sum a \in \text{UNIV}. \text{measure-pmf.prob } (\text{pmf-of-set } (\text{tuples UNIV } (\text{Suc } n))) (\{rs. P rs\} \cap \{rs. \text{hd } rs = a\})) \rangle$

```
    by (intro prob-finite-disjoint-cases) (auto simp add: disjoint-family-on-def)
  also have  $\langle \dots = ?rhs \rangle$  using A1
    by (simp add: sum-divide-distrib)
  finally show ?thesis .
qed
```

```
end
```

2 Generic Public-coin Interactive Proofs

```
theory Public-Coin-Proofs
  imports Probability-Tools
begin
```

2.1 Generic definition

```
type-synonym ('i, 'r, 'a, 'resp, 'ps) prv = 'i ⇒ 'a ⇒ 'a list ⇒ 'r ⇒ 'ps ⇒ 'resp × 'ps
```

```
locale public-coin-proof =
  fixes ver0 :: 'i ⇒ 'vs ⇒ bool
  and ver1 :: 'i ⇒ 'resp ⇒ 'r ⇒ 'a ⇒ 'a list ⇒ 'vs ⇒ bool × 'i × 'vs
begin
```

```
fun prove :: 'vs ⇒ ('i, 'r, 'a, 'resp, 'ps) prv ⇒ 'ps ⇒ 'i ⇒ 'r ⇒ ('a × 'r) list ⇒ bool where
  prove vs prv ps I r [] ⟷ ver0 I vs |
  prove vs prv ps I r ((x, r')#rm) ⟷
    (let (resp, ps') = prv I x (map fst rm) r ps in
     let (ok, I', vs') = ver1 I resp r' x (map fst rm) vs in
     ok ∧ prove vs' prv ps' I' r' rm)
```

The parameters are

- ($ver0$, $ver1$) and vs are the verifier and its current state,
- prv and ps are the prover and its current state,
- $I \in S$ is the problem instance,
- r is the verifier's randomness for the current round.
- rs is the (list of) randomness for the remaining rounds, and
- xs is a list of public per-round information/

We assume that rs and xs have the same length.

```
end
```

2.2 Generic soundness and completeness

```
locale public-coin-proof-security =
  public-coin-proof ver0 ver1
  for ver0 :: 'i ⇒ 'vs ⇒ bool
  and ver1 :: 'i ⇒ 'resp ⇒ 'r ⇒ 'a ⇒ 'a list ⇒ 'vs ⇒ bool × 'i × 'vs +
  fixes S :: 'i set — problem specification
  and honest-pr :: ('i, 'r, 'a, 'resp, 'ps) prv
  and compl-err :: 'i ⇒ real
  and sound-err :: 'i ⇒ real
  and compl-assm :: 'vs ⇒ 'ps ⇒ 'i ⇒ 'a list ⇒ bool
  and sound-assm :: 'vs ⇒ 'ps ⇒ 'i ⇒ 'a list ⇒ bool
  assumes
    completeness:
      [ I ∈ S; compl-assm vs ps I xs ] ⇒
```


measure-pmf.prob
 (*pmf-of-set (tuples UNIV (length xs))*)
 {*rs. prove vs honest-pr ps I r (zip xs rs)*} $\geq 1 - \text{compl-err } I$ **and**

soundness:

$\llbracket I \notin S; \text{sound-assm } vs \text{ ps } I \text{ xs} \rrbracket \implies$
measure-pmf.prob
 (*pmf-of-set (tuples UNIV (length xs))*)
 {*rs. prove vs pr ps I r (zip xs rs)*} $\leq \text{sound-err } I$

locale *public-coin-proof-strong-props* =
public-coin-proof ver0 ver1
for *ver0* :: '*i* \Rightarrow '*vs* \Rightarrow *bool*
and *ver1* :: '*i* \Rightarrow '*resp* \Rightarrow '*r::finite* \Rightarrow '*a* \Rightarrow '*a list* \Rightarrow '*vs* \Rightarrow *bool* \times '*i* \times '*vs* +
fixes *S* :: '*i set* — problem specification
and *honest-pr* :: ('*i*, '*r*, '*a*, '*resp*, '*ps*) *prv*
and *sound-err* :: '*i* \Rightarrow *real*
and *compl-assm* :: '*vs* \Rightarrow '*ps* \Rightarrow '*i* \Rightarrow '*a list* \Rightarrow *bool*
and *sound-assm* :: '*vs* \Rightarrow '*ps* \Rightarrow '*i* \Rightarrow '*a list* \Rightarrow *bool*
assumes
completeness:
 $\llbracket I \in S; \text{compl-assm } vs \text{ ps } I (\text{map fst } rm) \rrbracket \implies \text{prove } vs \text{ honest-pr } ps \text{ I } r \text{ rm}$ **and**
soundness:
 $\llbracket I \notin S; \text{sound-assm } vs \text{ ps } I \text{ xs} \rrbracket \implies$
measure-pmf.prob
 (*pmf-of-set (tuples UNIV (length xs))*)
 {*rs. prove vs pr ps I r (zip xs rs)*} $\leq \text{sound-err } I$

begin

Show that this locale satisfies the weaker assumptions of *public-coin-proof-security*.

sublocale *pc-props*:

public-coin-proof-security ver0 ver1 S honest-pr λ . 0 sound-err compl-assm sound-assm
by (*unfold-locales*)
 (*fastforce simp add: prob-pmf-of-set-geq-1 tuples-Suc completeness,*
clarsimp simp add: soundness)

end

end

3 Substitutions

theory *Substitutions*

imports

Main

HOL-Library.FuncSet

begin

type-synonym ('v, 'a) *subst* = 'v \rightarrow 'a

definition *substs* :: 'v set \Rightarrow 'a set \Rightarrow ('v, 'a) *subst* set **where**

substs V H = { σ . dom σ = V \wedge ran $\sigma \subseteq$ H}

Small lemmas about the set of substitutions

lemma *substE* [*elim*]: $\llbracket \sigma \in \text{substs } V H; \llbracket \text{dom } \sigma = V; \text{ran } \sigma \subseteq H \rrbracket \Longrightarrow P \rrbracket \Longrightarrow P$

by (*simp add: substs-def*)

lemma *substs-empty-dom* [*simp*]: *substs* {} H = {*Map.empty*}

by (*auto simp add: substs-def*)

lemma *substs-finite*: $\llbracket \text{finite } V; \text{finite } H \rrbracket \Longrightarrow \text{finite } (\text{substs } V H)$

by (*simp add: finite-set-of-finite-maps substs-def*)

lemma *substs-nonempty*:

assumes $H \neq \{\}$

shows *substs* V H $\neq \{\}$

proof –

obtain *h* **where** *A1*: $h \in H$ **using** *assms* **by**(*auto*)

obtain ϱ **where** *A2*: $\varrho = (\lambda v. \text{if } v \in V \text{ then Some } h \text{ else None})$ **by**(*simp*)

have $\varrho \in \text{substs } V H$ **using** *A1 A2* **by**(*auto simp add: substs-def ran-def dom-def*)

then show *?thesis* **by**(*auto*)

qed

lemma *subst-dom*: $\langle \llbracket \varrho \in \text{substs } V H; x \notin V \rrbracket \Longrightarrow x \notin \text{dom } \varrho \rangle$

by(*auto simp add: substs-def*)

lemma *subst-add*:

assumes $x \in V$ **and** $\varrho \in \text{substs } (V - \{x\}) H$ **and** $a \in H$

shows $\varrho(x \mapsto a) \in \text{substs } V H$

using *assms*

by(*simp add: substs-def*)

(*auto simp add: dom-def ran-def*)

lemma *subst-im*:

assumes $x \in V$ **and** $\varrho \in \text{substs } V H$

shows *the* (ϱx) $\in H$

using *assms*

by(*auto simp add: substs-def dom-def ran-def*)

lemma *subst-restr*:

assumes $x \in V$ **and** $\varrho \in \text{substs } V H$

shows $\varrho \upharpoonright^{\text{dom } \varrho - \{x\}} \in \text{substs } (V - \{x\}) H$

using *assms*

by(*auto simp add: subst-def ran-def dom-def restrict-map-def*)

Bijection between sets of substitutions

lemma *restrict-map-dom*: $\sigma \mid' \text{dom } \sigma = \sigma$

by (*metis (no-types, lifting) domIff map-le-antisym map-le-def restrict-in restrict-out*)

lemma *bij-betw-set-substs*:

assumes $x \in V$

defines $f \equiv \lambda(a, \sigma::'v \rightarrow 'a). \sigma(x \mapsto a)$

and $g \equiv \lambda\vartheta::'v \rightarrow 'a. (\text{the } (\vartheta x), \vartheta \mid'(\text{dom } \vartheta - \{x\}))$

shows *bij-betw* f

$(H \times \text{substs } (V - \{x\}) H)$

$(\text{substs } V H)$

proof (*intro bij-betwI*)

show $f \in H \times \text{substs } (V - \{x\}) H \rightarrow \text{substs } V H$

using *assms*

by(*auto simp add: f-def subst-add*)

next

show $g \in \text{substs } V H \rightarrow H \times \text{substs } (V - \{x\}) H$

using *assms*

by(*auto simp add: g-def subst-im subst-restr*)

next

fix xa

assume $xa \in H \times \text{substs } (V - \{x\}) H$

then show $g (f xa) = xa$ **using** *assms*

by (*smt (verit, ccfv-threshold) Diff-insert-absorb SigmaE case-prod-conv domI*

fun-upd-None-restrict fun-upd-same fun-upd-upd mk-disjoint-insert option.sel restrict-map-dom subst-dom)

next

fix y

assume $y \in \text{substs } V H$

then show $f (g y) = y$ **using** *assms*

by(*auto simp add: g-def f-def*)

(*metis domIff fun-upd-restrict map-upd-triv option.exhaust-sel restrict-map-dom substE*)

qed

end

4 Abstract Multivariate Polynomials

theory *Abstract-Multivariate-Polynomials*

imports

Substitutions

HOL-Analysis.Finite-Cartesian-Product

begin

Multivariate polynomials, abstractly

locale *multi-variate-polynomial* =

fixes *vars* :: $\langle 'p :: \text{comm-monoid-add} \Rightarrow 'v \text{ set} \rangle$

and *deg* :: $\langle 'p \Rightarrow \text{nat} \rangle$

and *eval* :: $\langle 'p \Rightarrow ('v, 'a::\text{finite}) \text{ subst} \Rightarrow 'b :: \text{comm-monoid-add} \rangle$

and *inst* :: $\langle 'p \Rightarrow ('v, 'a) \text{ subst} \Rightarrow 'p \rangle$

assumes

— *vars*

vars-finite: $\langle \text{finite} (\text{vars } p) \rangle$ **and**

vars-zero: $\langle \text{vars } 0 = \{\} \rangle$ **and**

vars-add: $\langle \text{vars} (p + q) \subseteq \text{vars } p \cup \text{vars } q \rangle$ **and**

vars-inst: $\langle \text{vars} (\text{inst } p \sigma) \subseteq \text{vars } p - \text{dom } \sigma \rangle$ **and**

— *degree*

deg-zero: $\langle \text{deg } 0 = 0 \rangle$ **and**

deg-add: $\langle \text{deg} (p + q) \leq \max (\text{deg } p) (\text{deg } q) \rangle$ **and**

deg-inst: $\langle \text{deg} (\text{inst } p \varrho) \leq \text{deg } p \rangle$ **and**

— *eval*

eval-zero: $\langle \text{eval } 0 \sigma = 0 \rangle$ **and**

eval-add: $\langle \text{vars } p \cup \text{vars } q \subseteq \text{dom } \sigma \implies \text{eval} (p + q) \sigma = \text{eval } p \sigma + \text{eval } q \sigma \rangle$ **and**

eval-inst: $\langle \text{vars } p \subseteq \text{dom } \sigma \cup \text{dom } \varrho \implies \text{eval} (\text{inst } p \sigma) \varrho = \text{eval } p (\varrho ++ \sigma) \rangle$ **and**

— *small number of roots (variant for two polynomials)*

roots: $\langle \text{card} \{r. \text{deg } p \leq d \wedge \text{vars } p \subseteq \{x\} \wedge \text{deg } q \leq d \wedge \text{vars } q \subseteq \{x\} \wedge p \neq q \wedge \text{eval } p [x \mapsto r] = \text{eval } q [x \mapsto r]\} \leq d \rangle$

begin

lemmas *vars-addD* = *vars-add*[*THEN subsetD*]

4.1 Arity: definition and some lemmas

definition *arity* :: $\langle 'p \Rightarrow \text{nat} \rangle$ **where**

$\langle \text{arity } p = \text{card} (\text{vars } p) \rangle$

lemma *arity-zero*: $\langle \text{arity } 0 = 0 \rangle$

by (*simp add: arity-def vars-zero*)

lemma *arity-add*: $\langle \text{arity} (p + q) \leq \text{arity } p + \text{arity } q \rangle$

proof —

have $\langle \text{card} (\text{vars} (p + q)) \leq \text{card} (\text{vars } p \cup \text{vars } q) \rangle$

by (*intro card-mono*) (*auto simp add: vars-add vars-finite*)

also have $\langle \dots \leq \text{card} (\text{vars } p) + \text{card} (\text{vars } q) \rangle$ **by** (*simp add: card-Un-le*)

finally show *?thesis* **by** (*simp add: arity-def*)

qed

lemma *arity-inst*:
assumes $\langle \text{dom } \sigma \subseteq \text{vars } p \rangle$
shows $\langle \text{arity } (\text{inst } p \ \sigma) \leq \text{arity } p - \text{card } (\text{dom } \sigma) \rangle$
proof –
have $\langle \text{card } (\text{vars } (\text{inst } p \ \sigma)) \leq \text{card } (\text{vars } p - \text{dom } \sigma) \rangle$
by (*auto simp add: vars-finite vars-inst card-mono*)
also have $\langle \dots = \text{card } (\text{vars } p) - \text{card } (\text{dom } \sigma) \rangle$ **using** *assms*
by (*simp add: card-Diff-subset finite-subset vars-finite*)
finally show *?thesis* **by** (*simp add: arity-def*)
qed

4.2 Lemmas about evaluation, degree, and variables of finite sums

lemma *eval-sum*:
assumes $\langle \text{finite } I \rangle \langle \bigwedge i. i \in I \implies \text{vars } (pp \ i) \subseteq \text{dom } \sigma \rangle$
shows $\langle \text{eval } (\sum_{i \in I}. pp \ i) \ \sigma = (\sum_{i \in I}. \text{eval } (pp \ i) \ \sigma) \rangle$
proof –
have $\langle \text{eval } (\sum_{i \in I}. pp \ i) \ \sigma = (\sum_{i \in I}. \text{eval } (pp \ i) \ \sigma) \wedge \text{vars } (\sum_{i \in I}. pp \ i) \subseteq \text{dom } \sigma \rangle$ **using** *assms*
proof (*induction rule: finite.induct*)
case *emptyI*
then show *?case* **by** (*simp add: eval-zero vars-zero*)
next
case (*insertI A a*)
then show *?case*
by (*auto simp add: eval-add vars-add sum.insert-if dest!: vars-addD*)
qed
then show *?thesis ..*
qed

lemma *vars-sum*:
assumes $\langle \text{finite } I \rangle$
shows $\langle \text{vars } (\sum_{i \in I}. pp \ i) \subseteq (\bigcup_{i \in I}. \text{vars } (pp \ i)) \rangle$
using *assms*
proof (*induction rule: finite.induct*)
case *emptyI*
then show *?case* **by**(*auto simp add: vars-zero*)
next
case (*insertI A a*)
then show *?case* **using** *insertI* **by**(*auto simp add: sum.insert-if dest: vars-addD*)
qed

lemma *deg-sum*:
assumes $\langle \text{finite } I \rangle$ **and** $I \neq \{\}$
shows $\langle \text{deg } (\sum_{i \in I}. pp \ i) \leq \text{Max } \{ \text{deg } (pp \ i) \mid i. i \in I \} \rangle$
using *assms*
proof (*induction rule: finite.induct*)
case *emptyI*
then show *?case* **by**(*auto simp add: deg-zero*)
next
case (*insertI A a*)
show *?case*
proof(*cases A = \{\}*)

```

assume  $\langle A = \{\} \rangle$ 
then show ?thesis by(simp)
next
assume  $\langle A \neq \{\} \rangle$ 
then have *:  $\langle \text{Max} \{ \text{deg} (pp\ i) \mid i. i \in A \} \leq \text{Max} \{ \text{deg} (pp\ i) \mid i. i = a \vee i \in A \} \rangle$  using  $\langle \text{finite } A \rangle$ 
by (intro Max-mono) auto
show ?thesis using insertI  $\langle A \neq \{\} \rangle$ 
by (auto 4 4 simp add: sum.insert-if intro: Max-ge *[THEN [2] le-trans] deg-add[THEN le-trans])
qed
qed

```

4.3 Lemmas combining eval, sum, and inst

lemma *eval-sum-inst*:

```

assumes  $\langle \text{vars } p \subseteq V \cup \text{dom } \varrho \rangle$   $\langle \text{finite } V \rangle$ 
shows  $\langle \text{eval} (\sum \sigma \in \text{substs } V\ H. \text{inst } p\ \sigma)\ \varrho = (\sum \sigma \in \text{substs } V\ H. \text{eval } p\ (\varrho ++ \sigma)) \rangle$ 
proof –
have A1:  $\langle \bigwedge \sigma. \sigma \in \text{substs } V\ H \implies \text{vars} (\text{inst } p\ \sigma) \subseteq \text{dom } \varrho \rangle$  using assms(1) vars-inst by blast
have A2:  $\langle \bigwedge \sigma. \sigma \in \text{substs } V\ H \implies \text{vars } p \subseteq \text{dom } \sigma \cup \text{dom } \varrho \rangle$  using assms(1) by (auto)

have  $\langle \text{eval} (\sum \sigma \in \text{substs } V\ H. \text{inst } p\ \sigma)\ \varrho = (\sum \sigma \in \text{substs } V\ H. \text{eval} (\text{inst } p\ \sigma)\ \varrho) \rangle$  using A1
assms(2)
by (simp add: eval-sum substs-finite) — requires finite H
also have  $\langle \dots = (\sum \sigma \in \text{substs } V\ H. \text{eval } p\ (\varrho ++ \sigma)) \rangle$  using A2
by (simp add: eval-inst)
finally show ?thesis .
qed

```

lemma *eval-sum-inst-commute*:

```

assumes  $\langle \text{vars } p \subseteq \text{insert } x\ V \rangle$   $\langle x \notin V \rangle$   $\langle \text{finite } V \rangle$ 
shows  $\langle \text{eval} (\sum \sigma \in \text{substs } V\ H. \text{inst } p\ \sigma)\ [x \mapsto r] = (\sum \sigma \in \text{substs } V\ H. \text{eval} (\text{inst } p\ [x \mapsto r])\ \sigma) \rangle$ 
proof –
have  $\langle \text{eval} (\text{sum} (\text{inst } p)\ (\text{substs } V\ H))\ [x \mapsto r] = (\sum \sigma \in \text{substs } V\ H. \text{eval } p\ ([x \mapsto r] ++ \sigma)) \rangle$  using  $\langle \text{vars } p \subseteq \text{insert } x\ V \rangle$   $\langle \text{finite } V \rangle$ 
by (simp add: eval-sum-inst)
also have  $\langle \dots = (\sum \sigma \in \text{substs } V\ H. \text{eval } p\ (\sigma(x \mapsto r))) \rangle$  using  $\langle x \notin V \rangle$ 
by (intro Finite-Cartesian-Product.sum-cong-aux)
(auto simp add: map-add-comm subst-dom)
also have  $\langle \dots = (\sum \sigma \in \text{substs } V\ H. \text{eval} (\text{inst } p\ [x \mapsto r])\ \sigma) \rangle$  using  $\langle \text{vars } p \subseteq \text{insert } x\ V \rangle$ 
by (intro Finite-Cartesian-Product.sum-cong-aux)
(auto simp add: eval-inst)
finally show ?thesis .
qed

```

4.4 Merging sums over substitutions

lemma *sum-merge*:

```

assumes  $\langle x \notin V \rangle$ 
shows  $\langle \sum h \in H. (\sum \sigma \in \text{substs } V\ H. \text{eval } p\ ([x \mapsto h] ++ \sigma)) = (\sum \sigma \in \text{substs} (\text{insert } x\ V)\ H. \text{eval } p\ \sigma) \rangle$ 
proof –
have  $\langle \bigwedge h\ \sigma. (h, \sigma) \in H \times \text{substs } V\ H \implies \text{dom } [x \mapsto h] \cap \text{dom } \sigma = \{\} \rangle$  using  $\langle x \notin V \rangle$ 

```

```

by(auto simp add: subst-def)
then have *:  $\bigwedge h \sigma. (h, \sigma) \in H \times \text{subst } V H \implies [x \mapsto h] ++ \sigma = \sigma(x \mapsto h)$ 
by(auto simp add: map-add-comm)

have  $(\sum h \in H. (\sum \sigma \in \text{subst } V H. \text{eval } p ([x \mapsto h] ++ \sigma))) =$ 
 $(\sum (h, \sigma) \in H \times \text{subst } V H. \text{eval } p ([x \mapsto h] ++ \sigma))$ 
by(auto simp add: sum.cartesian-product)
also have ... =  $(\sum (h, \sigma) \in H \times \text{subst } V H. \text{eval } p (\sigma(x \mapsto h)))$  using *
by (intro Finite-Cartesian-Product.sum-cong-aux) (auto)
also have ... =  $(\sum \sigma \in \text{subst } (\text{insert } x V) H. \text{eval } p \sigma)$  using  $\langle x \notin V \rangle$ 
by(auto simp add: sum.reindex-bij-betw[OF bij-betw-set-substs,
 $\text{where } V1 = \text{insert } x V \text{ and } x1 = x \text{ and } H1 = H \text{ and } g = \lambda \sigma. \text{eval } p \sigma, \text{symmetric}]$ 
intro: Finite-Cartesian-Product.sum-cong-aux)
finally show ?thesis .
qed

end

end

```

5 Sumcheck Protocol

```

theory Sumcheck-Protocol
  imports
    Public-Coin-Proofs
    Abstract-Multivariate-Polynomials
begin

```

5.1 The sumcheck problem

Type of sumcheck instances

```

type-synonym ('p, 'a, 'b) sc-inst = 'a set × 'p × 'b

```

definition (in *multi-variate-polynomial*)

```

Sumcheck :: ('p, 'a, 'b) sc-inst set where
Sumcheck = {(H, p, v) | H p v. v = (∑ σ ∈ substs (vars p) H. eval p σ)}

```

5.2 The sumcheck protocol

Type of the prover

```

type-synonym ('p, 'a, 'b, 'v, 's) prover = (('p, 'a, 'b) sc-inst, 'a, 'v, 'p, 's) prv

```

Here is how the expanded type looks like

```

('p, 'a, 'b, 'v, 's) prover

```

.

```

context multi-variate-polynomial begin

```

Sumcheck function

```

fun sumcheck :: ('p, 'a, 'b, 'v, 's) prover ⇒ 's ⇒ ('p, 'a, 'b) sc-inst ⇒ 'a ⇒ ('v × 'a) list ⇒ bool
where
  sumcheck pr s (H, p, v) r-prev [] ←→ v = eval p Map.empty
| sumcheck pr s (H, p, v) r-prev ((x, r) # rm) ←→
  (let (q, s') = pr (H, p, v) x (map fst rm) r-prev s in
   vars q ⊆ {x} ∧ deg q ≤ deg p ∧
   v = (∑ y ∈ H. eval q [x ↦ y]) ∧
   sumcheck pr s' (H, inst p [x ↦ r], eval q [x ↦ r]) r rm)

```

Honest prover definition

```

fun honest-prover :: ('p, 'a, 'b, 'v, unit) prover where
  honest-prover (H, p, -) x xs - - = (∑ σ ∈ substs (set xs) H. inst p σ, ())

```

```

declare honest-prover.simps [simp del]

```

```

lemmas honest-prover-def = honest-prover.simps

```

Lemmas on variables and degree of the honest prover.

lemma *honest-prover-vars*:

```

assumes vars p ⊆ insert x V finite V H ≠ {} finite H
shows vars (∑ σ ∈ substs V H. inst p σ) ⊆ {x}

```

proof –


```

have *:  $\bigwedge \sigma. \sigma \in \text{substs } V H \implies \text{vars } (\text{inst } p \sigma) \subseteq \{x\}$  using assms
by (metis (no-types, lifting) Diff-eq-empty-iff Diff-insert subset-iff substE vars-inst)

have vars (sum (inst p) (substs V H))  $\subseteq (\bigcup \sigma \in \text{substs } V H. \text{vars } (\text{inst } p \sigma))$ 
using  $\langle \text{finite } V \rangle \langle \text{finite } H \rangle$ 
by (auto simp add: vars-sum substs-finite)
also have  $\dots \subseteq \{x\}$  using  $\langle H \neq \{\} \rangle *$ 
by (auto simp add: substs-nonempty vars-finite substs-finite)
finally show ?thesis .
qed

```

```

lemma honest-prover-deg:
assumes  $H \neq \{\}$  finite V
shows  $\text{deg } (\sum \sigma \in \text{substs } V H. \text{inst } p \sigma) \leq \text{deg } p$ 
proof -
have  $\text{deg } (\sum \sigma \in \text{substs } V H. \text{inst } p \sigma) \leq \text{Max } \{\text{deg } (\text{inst } p \sigma) \mid \sigma. \sigma \in \text{substs } V H\}$ 
by (auto simp add: substs-finite substs-nonempty deg-sum assms)
also have  $\dots \leq \text{deg } p$ 
by (auto simp add: substs-finite substs-nonempty deg-inst assms)
finally show ?thesis .
qed

```

5.3 The sumcheck protocol as a public-coin proof instance

Define verifier functions

```

fun sc-ver0 :: ('p, 'a, 'b) sc-inst  $\Rightarrow$  unit  $\Rightarrow$  bool where
  sc-ver0 (H, p, v) ()  $\longleftrightarrow v = \text{eval } p \text{ Map.empty}$ 

fun sc-ver1 ::
  ('p, 'a, 'b) sc-inst  $\Rightarrow$  'p  $\Rightarrow$  'a  $\Rightarrow$  'v  $\Rightarrow$  'v list  $\Rightarrow$  unit  $\Rightarrow$  bool  $\times$  ('p, 'a, 'b) sc-inst  $\times$  unit
where
  sc-ver1 (H, p, v) q r x - () = (
     $\text{vars } q \subseteq \{x\} \wedge \text{deg } q \leq \text{deg } p \wedge v = (\sum y \in H. \text{eval } q [x \mapsto y])$ ,
    (H, inst p [x  $\mapsto$  r], eval q [x  $\mapsto$  r]),
    ()
  )

```

```

sublocale sc: public-coin-proof sc-ver0 sc-ver1 .

```

Equivalence of *sumcheck* with public-coin proofs instance

```

lemma prove-sc-eq-sumcheck:
   $\langle \text{sc.prove } () \text{ pr ps } (H, p, v) \text{ r rm} = \text{sumcheck pr ps } (H, p, v) \text{ r rm} \rangle$ 
proof (induction ()) pr ps (H, p, v) r rm arbitrary: p v rule: sc.prove.induct)
  case (1 vs prv ps r)
  then show ?case by (simp)
next
  case (2 vs prv ps r r' rs x xs)
  then show ?case by (simp split:prod.split)
qed

```

end

end

6 Completeness Proof for the Sumcheck Protocol

```

theory Completeness-Proof
  imports
    Sumcheck-Protocol
begin

context multi-variate-polynomial begin

Completeness proof

theorem completeness-inductive:
  assumes
     $\langle v = (\sum \sigma \in \text{substs } (\text{set } (\text{map } \text{fst } \text{rm})) \text{ } H. \text{eval } p \sigma) \rangle$ 
     $\langle \text{vars } p \subseteq \text{set } (\text{map } \text{fst } \text{rm}) \rangle$ 
     $\langle \text{distinct } (\text{map } \text{fst } \text{rm}) \rangle$ 
     $\langle H \neq \{\} \rangle$ 
  shows
    sumcheck honest-prover  $u$   $(H, p, v)$   $r$ -prev  $\text{rm}$ 
  using assms
proof(induction honest-prover  $u$   $(H, p, v)$   $r$ -prev  $\text{rm}$  arbitrary: H p v rule: sumcheck.induct)
  case  $(1 \text{ } s \text{ } H \text{ } p \text{ } v \text{ } r\text{-prev})$ 
  then show  $?case$  by(simp)
next
  case  $(2 \text{ } s \text{ } H \text{ } p \text{ } v \text{ } r\text{-prev } x \text{ } r \text{ } \text{rm})$ 

  note  $IH = 2(1)$  — induction hypothesis

  let  $?V = \text{set } (\text{map } \text{fst } \text{rm})$ 
  let  $?q = (\sum \sigma \in \text{substs } ?V \text{ } H. \text{inst } p \sigma)$ 

  have  $\langle \text{vars } p \subseteq \text{insert } x \text{ } ?V \rangle \langle x \notin ?V \rangle \langle \text{distinct } (\text{map } \text{fst } \text{rm}) \rangle$ 
  using  $2(3-4)$  by(auto)

  — evaluation check
  have  $\langle (\sum \sigma \in \text{substs } (\text{insert } x \text{ } ?V) \text{ } H. \text{eval } p \sigma) = (\sum h \in H. \text{eval } ?q [x \mapsto h]) \rangle$ 
  proof —
  have  $(\sum \sigma \in \text{substs } (\text{insert } x \text{ } ?V) \text{ } H. \text{eval } p \sigma) =$ 
     $(\sum h \in H. (\sum \sigma \in \text{substs } ?V \text{ } H. \text{eval } p ([x \mapsto h] ++ \sigma)))$ 
  using  $\langle x \notin ?V \rangle$ 
  by(auto simp add: sum-merge)
  also have  $\dots = (\sum h \in H. \text{eval } ?q [x \mapsto h])$ 
  using  $\langle \text{vars } p \subseteq \text{insert } x \text{ } ?V \rangle$ 
  by(auto simp add: eval-sum-inst)
  finally show  $?thesis$  .

qed
moreover
  — recursive check
  have  $\langle \text{sumcheck honest-prover } () \text{ } (H, \text{inst } p [x \mapsto r], \text{eval } ?q [x \mapsto r]) \text{ } r \text{ } \text{rm} \rangle$ 
  proof —
  have  $\langle \text{vars } (\text{inst } p [x \mapsto r]) \subseteq ?V \rangle$ 
  using  $\langle \text{vars } p \subseteq \text{insert } x \text{ } ?V \rangle$  vars-inst by fastforce
  moreover
  have  $\text{eval } ?q [x \mapsto r] = (\sum \sigma \in \text{substs } ?V \text{ } H. \text{eval } (\text{inst } p [x \mapsto r]) \sigma)$ 

```

```

using ⟨vars p ⊆ insert x ?V⟩ ⟨x ∉ set (map fst rm)⟩
by (auto simp add: eval-sum-inst-commute)
ultimately
show ?thesis using IH ⟨distinct (map fst rm)⟩ ⟨H ≠ {}⟩
by (simp add: honest-prover-def)
qed
ultimately show ?case using 2(2-3,5)
by (simp add: honest-prover-def honest-prover-vars honest-prover-deg)
qed

```

corollary *completeness:*

assumes

```

⟨(H, p, v) ∈ Sumcheck⟩
⟨vars p = set (map fst rm)⟩
⟨distinct (map fst rm)⟩
⟨H ≠ {}⟩

```

shows

```

sumcheck honest-prover u (H, p, v) r rm

```

using *assms*

by (auto simp add: Sumcheck-def intro: completeness-inductive)

end

end

7 Soundness Proof for the Sumcheck Protocol

theory *Soundness-Proof*

imports

Probability-Tools

Sumcheck-Protocol

begin

context *multi-variate-polynomial* **begin**

— Helper lemma: Proves that the probability of two different polynomials evaluating to the same value is small.

lemma *prob-roots*:

assumes $\text{deg } q2 \leq \text{deg } p$ **and** $\text{vars } q2 \subseteq \{x\}$

shows $\text{measure-pmf.prob } (\text{pmf-of-set } UNIV)$

$\{r. \text{deg } q1 \leq \text{deg } p \wedge \text{vars } q1 \subseteq \{x\} \wedge q1 \neq q2 \wedge \text{eval } q1 [x \mapsto r] = \text{eval } q2 [x \mapsto r]\}$
 $\leq \text{real } (\text{deg } p) / \text{real } \text{CARD}(a)$

proof –

have $\text{card } \{r. \text{deg } q1 \leq \text{deg } p \wedge \text{vars } q1 \subseteq \{x\} \wedge$
 $q1 \neq q2 \wedge \text{eval } q1 [x \mapsto r] = \text{eval } q2 [x \mapsto r]\} =$

$\text{card } \{r. \text{deg } q1 \leq \text{deg } p \wedge \text{vars } q1 \subseteq \{x\} \wedge$
 $\text{deg } q2 \leq \text{deg } p \wedge \text{vars } q2 \subseteq \{x\} \wedge$

$q1 \neq q2 \wedge \text{eval } q1 [x \mapsto r] = \text{eval } q2 [x \mapsto r]\}$ **using** *assms* **by**(*auto*)

also have $\dots \leq \text{deg } p$ **by**(*auto simp add: roots*)

finally show *?thesis* **by**(*auto simp add: measure-pmf-of-set divide-right-mono*)

qed

Soundness proof

theorem *soundness-inductive*:

assumes

$\text{vars } p \subseteq \text{set } vs$ **and**

$\text{deg } p \leq d$ **and**

distinct vs **and**

$H \neq \{\}$

shows

measure-pmf.prob

$(\text{pmf-of-set } (\text{tuples } UNIV (\text{length } vs)))$

$\{rs.$

$\text{sumcheck } pr s (H, p, v) r (\text{zip } vs rs) \wedge$

$v \neq (\sum \sigma \in \text{subst } (\text{set } vs) H. \text{eval } p \sigma)\}$

$\leq \text{real } (\text{length } vs) * \text{real } d / \text{real } (\text{CARD}(a))$

using *assms*

proof(*induction vs arbitrary: s p v r*)

case *Nil*

show *?case*

by(*simp*)

next

case (*Cons x vs*)

— abbreviations

let *?prob* = $\text{measure-pmf.prob } (\text{pmf-of-set } (\text{tuples } UNIV (\text{Suc } (\text{length } vs))))$

let *?reduced-prob* = $\text{measure-pmf.prob } (\text{pmf-of-set } (\text{tuples } UNIV (\text{length } vs)))$

let $?q = \sum \sigma \in \text{subst} (\text{set } vs) H. \text{inst } p \sigma$ — honest polynomial q

let $?pr-q = \text{fst} (pr (H, p, v) x vs r s)$ — polynomial q from prover

let $?pr-s' = \text{snd} (pr (H, p, v) x vs r s)$ — prover's next state

— some useful derived facts

have $\langle \text{vars } (p) \subseteq \text{insert } x (\text{set } vs) \rangle \langle x \notin \text{set } vs \rangle \langle \text{distinct } vs \rangle$

using $\langle \text{vars } (p) \leq \text{set } (x \# vs) \rangle \langle \text{distinct } (x \# vs) \rangle$ **by** *auto*

have *P0*:

$\langle ?prob \{r1 \# rs \mid r1 \text{ rs.}$

$\text{deg } (?pr-q) \leq \text{deg } (p) \wedge \text{vars } (?pr-q) \subseteq \{x\} \wedge$

$v = (\sum a \in H. \text{eval } (?pr-q) [x \mapsto a]) \wedge$

$\text{sumcheck } pr (?pr-s') (H, \text{inst } (p) [x \mapsto r1], \text{eval } (?pr-q) [x \mapsto r1]) r1 (\text{zip } vs \text{ rs}) \wedge$

$v \neq (\sum \sigma \in \text{subst } (\text{insert } x (\text{set } vs)) H. \text{eval } (p) \sigma) \wedge ?pr-q = ?q = 0 \rangle$

proof —

have $(\sum a \in H. \text{eval } (?q) [x \mapsto a]) =$

$(\sum a \in H. \sum \sigma \in \text{subst } (\text{set } vs) H. \text{eval } (p) ([x \mapsto a] ++ \sigma))$

using $\langle \text{vars } (p) \subseteq \text{insert } x (\text{set } vs) \rangle \langle x \notin \text{set } vs \rangle$

by *(auto simp add: eval-sum-inst)*

moreover

have $(\sum a \in H. \sum \sigma \in \text{subst } (\text{set } vs) H. \text{eval } (p) ([x \mapsto a] ++ \sigma)) =$

$(\sum \sigma \in \text{subst } (\text{insert } x (\text{set } vs)) H. \text{eval } (p) \sigma)$ **using** $\langle x \notin \text{set } vs \rangle$

by *(auto simp add: sum-merge)*

ultimately

have $\{r1 \# rs \mid r1 \text{ rs.}$

$v = (\sum a \in H. \text{eval } (?pr-q) [x \mapsto a]) \wedge$

$v \neq (\sum \sigma \in \text{subst } (\text{insert } x (\text{set } vs)) H. \text{eval } (p) \sigma) \wedge ?pr-q = ?q = \{\}$

by *(auto)*

then show *?thesis*

by *(intro prob-empty) (auto 4 4)*

qed

{ — left-hand-side case where we use the roots assumption

have $\langle ?prob \{r1 \# rs \mid r1 \text{ rs.}$

$\text{deg } (?pr-q) \leq \text{deg } (p) \wedge \text{vars } (?pr-q) \subseteq \{x\} \wedge$

$v = (\sum a \in H. \text{eval } (?pr-q) [x \mapsto a]) \wedge$

$\text{sumcheck } pr (?pr-s') (H, \text{inst } (p) [x \mapsto r1], \text{eval } (?pr-q) [x \mapsto r1]) r1 (\text{zip } vs \text{ rs}) \wedge$

$?pr-q \neq ?q \wedge \text{eval } (?pr-q) [x \mapsto r1] = \text{eval } (?q) [x \mapsto r1] \leq$

$?prob \{r1 \# rs \mid r1 \text{ rs.}$

$\text{deg } (?pr-q) \leq \text{deg } (p) \wedge \text{vars } (?pr-q) \subseteq \{x\} \wedge$

$?pr-q \neq ?q \wedge \text{eval } (?pr-q) [x \mapsto r1] = \text{eval } (?q) [x \mapsto r1] \rangle$

by *(intro prob-mono) (auto 4 4)*

also have $\langle \dots =$

$\text{measure-pmf.prob } (\text{pmf-of-set UNIV}) \{r1.$

$\text{deg } (?pr-q) \leq \text{deg } (p) \wedge \text{vars } (?pr-q) \subseteq \{x\} \wedge$

$?pr-q \neq ?q \wedge \text{eval } (?pr-q) [x \mapsto r1] = \text{eval } (?q) [x \mapsto r1] \rangle$

by *(auto simp add: prob-tuples-hd-tl-indep[where Q = $\lambda rs. \text{True}$, simplified])*

also have $\langle \dots \leq \text{real } (\text{deg } (p)) / \text{real } \text{CARD}'(a) \rangle$

using $\langle \text{vars } (p) \subseteq \text{insert } x (\text{set } vs) \rangle \langle H \neq \{\} \rangle$

by *(auto simp add: prob-roots honest-prover-deg honest-prover-vars)*

also have $\langle \dots \leq \text{real } d / \text{real } \text{CARD}(a) \rangle$ **using** $\langle \text{deg } (p) \leq d \rangle$
by $(\text{simp add: divide-right-mono})$

finally

have $\langle ?\text{prob } \{r1\#rs \mid r1 \text{ rs.}$
 $\text{deg } (?pr-q) \leq \text{deg } (p) \wedge \text{vars } (?pr-q) \subseteq \{x\} \wedge$
 $v = (\sum a \in H. \text{eval } (?pr-q) [x \mapsto a]) \wedge$
 $\text{sumcheck } pr \ (?pr-s') (H, \text{inst } (p) [x \mapsto r1], \text{eval } (?pr-q) [x \mapsto r1]) r1 (\text{zip } vs \text{ rs}) \wedge$
 $?pr-q \neq ?q \wedge \text{eval } (?pr-q) [x \mapsto r1] = \text{eval } (?q) [x \mapsto r1]\} \wedge$
 $\leq \text{real } d / \text{real } \text{CARD}(a) \rangle$.

}

note $RP\text{-left} = \text{this}$

{

have $*$: $\langle \bigwedge \alpha. \text{eval } (?q) [x \mapsto \alpha] = (\sum \sigma \in \text{substs } (\text{set } vs) H. \text{eval } (\text{inst } (p) [x \mapsto \alpha]) \sigma) \rangle$
using $\langle \text{vars } (p) \subseteq \text{insert } x (\text{set } vs) \rangle \langle x \notin \text{set } vs \rangle$
by $(\text{auto simp add: eval-sum-inst-commute})$

have $\langle \bigwedge \alpha. \text{vars } (\text{inst } (p) [x \mapsto \alpha]) \subseteq \text{set } vs \rangle$ **using** $\text{vars-inst } \langle \text{vars } (p) \subseteq \text{insert } x (\text{set } vs) \rangle$
by fastforce

have $\langle \bigwedge \alpha. \text{deg } (\text{inst } (p) [x \mapsto \alpha]) \leq d \rangle$ **using** $\text{deg-inst } \langle \text{deg } (p) \leq d \rangle$
using le-trans by blast

— right-hand-side case where we apply the induction hypothesis

have $\langle ?\text{prob } \{r1\#rs \mid r1 \text{ rs.}$
 $\text{deg } (?pr-q) \leq \text{deg } (p) \wedge \text{vars } (?pr-q) \subseteq \{x\} \wedge$
 $v = (\sum a \in H. \text{eval } (?pr-q) [x \mapsto a]) \wedge$
 $\text{sumcheck } pr \ (?pr-s') (H, \text{inst } (p) [x \mapsto r1], \text{eval } (?pr-q) [x \mapsto r1]) r1 (\text{zip } vs \text{ rs}) \wedge$
 $?pr-q \neq ?q \wedge \text{eval } (?pr-q) [x \mapsto r1] \neq \text{eval } (?q) [x \mapsto r1]\} \wedge$
 $\leq ?\text{prob } \{r1\#rs \mid r1 \text{ rs.}$
 $\text{sumcheck } pr \ (?pr-s') (H, \text{inst } (p) [x \mapsto r1], \text{eval } (?pr-q) [x \mapsto r1]) r1 (\text{zip } vs \text{ rs}) \wedge$
 $\text{eval } (?pr-q) [x \mapsto r1] \neq (\sum \sigma \in \text{substs } (\text{set } vs) H. \text{eval } (\text{inst } (p) [x \mapsto r1]) \sigma)\} \rangle$
by $(\text{intro prob-mono}) (\text{auto simp add: } *)$

— fix $r1$

also have $\langle \dots = (\sum \alpha \in \text{UNIV.}$
 $?reduced\text{-prob } \{rs.$
 $\text{sumcheck } pr \ (?pr-s') (H, \text{inst } (p) [x \mapsto \alpha], \text{eval } (?pr-q) [x \mapsto \alpha]) \alpha (\text{zip } vs \text{ rs}) \wedge$
 $\text{eval } (?pr-q) [x \mapsto \alpha] \neq (\sum \sigma \in \text{substs } (\text{set } vs) H. \text{eval } (\text{inst } (p) [x \mapsto \alpha]) \sigma)\} \rangle$
 $/ \text{real}(\text{CARD}(a)) \rangle$
by $(\text{auto simp add: prob-tuples-fixed-hd})$

— apply the induction hypothesis

also have $\langle \dots \leq (\sum \alpha \in (\text{UNIV}::'a \text{ set}). \text{real } (\text{length } vs) * \text{real } d / \text{real } \text{CARD}(a))$
 $/ \text{real}(\text{CARD}(a)) \rangle$

using $\langle \bigwedge \alpha. \text{vars } (\text{inst } (p) [x \mapsto \alpha]) \subseteq \text{set } vs \rangle$
 $\langle \bigwedge \alpha. \text{deg } (\text{inst } (p) [x \mapsto \alpha]) \leq d \rangle$
 $\langle \text{distinct } vs \rangle \langle H \neq \{\} \rangle$

by $(\text{intro divide-right-mono sum-mono Cons.IH}) (\text{auto})$

also have $\langle \dots = \text{real } (\text{length } vs) * \text{real } d / \text{real } \text{CARD}(a) \rangle$
by fastforce

finally

have $\langle ?prob \{r1\#rs \mid r1 \text{ rs.}$

$$deg (?pr-q) \leq deg (p) \wedge vars (?pr-q) \subseteq \{x\} \wedge$$

$$v = (\sum_{a \in H}. eval (?pr-q) [x \mapsto a]) \wedge$$

$$sumcheck pr (?pr-s') (H, inst (p) [x \mapsto r1], eval (?pr-q) [x \mapsto r1]) r1 (zip vs rs) \wedge$$

$$?pr-q \neq ?q \wedge eval (?pr-q) [x \mapsto r1] \neq eval (?q) [x \mapsto r1]\rangle$$

$$\leq real (length vs) * real d / real CARD('a)\rangle .$$

}

note *RP-right = this*

— main equational reasoning proof

have $\langle ?prob \{rs.$

$$sumcheck pr s (H, p, v) r (zip (x \# vs) rs) \wedge$$

$$v \neq (\sum \sigma \in substs (insert x (set vs)) H. eval p \sigma)\rangle$$

$$= ?prob \{r1\#rs \mid r1 \text{ rs. } sumcheck pr s (H, p, v) r (zip (x \# vs) (r1\#rs))$$

$$\wedge v \neq (\sum \sigma \in substs (insert x (set vs)) H. eval p \sigma)\rangle$$

by(*intro prob-cong*) (*auto simp add: tuples-Suc*)

— unfold sumcheck

also have $\langle \dots = ?prob \{r1\#rs \mid r1 \text{ rs.}$

$$(let (q, s') = pr (H, p, v) x vs r s \text{ in}$$

$$deg q \leq deg p \wedge vars q \subseteq \{x\} \wedge$$

$$v = (\sum_{a \in H}. eval q [x \mapsto a]) \wedge$$

$$sumcheck pr s' (H, inst p [x \mapsto r1], eval q [x \mapsto r1]) r1 (zip vs rs) \wedge$$

$$v \neq (\sum \sigma \in substs (insert x (set vs)) H. eval p \sigma)\rangle$$

by(*intro prob-cong*) (*auto del: subsetI*)

also have $\langle \dots = ?prob \{r1\#rs \mid r1 \text{ rs.}$

$$deg ?pr-q \leq deg p \wedge vars ?pr-q \subseteq \{x\} \wedge$$

$$v = (\sum_{a \in H}. eval ?pr-q [x \mapsto a]) \wedge$$

$$sumcheck pr ?pr-s' (H, inst p [x \mapsto r1], eval ?pr-q [x \mapsto r1]) r1 (zip vs rs) \wedge$$

$$v \neq (\sum \sigma \in substs (insert x (set vs)) H. eval p \sigma)\rangle$$

by (*intro prob-cong*) (*auto del: subsetI*)

— case split on whether prover's polynomial q equals honest one

also have $\langle \dots = ?prob \{r1\#rs \mid r1 \text{ rs.}$

$$deg (?pr-q) \leq deg (p) \wedge vars (?pr-q) \subseteq \{x\} \wedge$$

$$v = (\sum_{a \in H}. eval (?pr-q) [x \mapsto a]) \wedge$$

$$sumcheck pr (?pr-s') (H, inst (p) [x \mapsto r1], eval (?pr-q) [x \mapsto r1]) r1 (zip vs rs) \wedge$$

$$v \neq (\sum \sigma \in substs (insert x (set vs)) H. eval (p) \sigma) \wedge ?pr-q = ?q\}$$

$$+ ?prob \{r1\#rs \mid r1 \text{ rs.}$$

$$deg (?pr-q) \leq deg (p) \wedge vars (?pr-q) \subseteq \{x\} \wedge$$

$$v = (\sum_{a \in H}. eval (?pr-q) [x \mapsto a]) \wedge$$

$$sumcheck pr (?pr-s') (H, inst (p) [x \mapsto r1], eval (?pr-q) [x \mapsto r1]) r1 (zip vs rs) \wedge$$

$$v \neq (\sum \sigma \in substs (insert x (set vs)) H. eval (p) \sigma) \wedge ?pr-q \neq ?q\}$$

by(*intro prob-disjoint-cases*) *auto*

— first probability is 0

also have $\langle \dots = ?prob \{r1\#rs \mid r1 \text{ rs.}$

$$deg (?pr-q) \leq deg (p) \wedge vars (?pr-q) \subseteq \{x\} \wedge$$

$$v = (\sum_{a \in H}. eval (?pr-q) [x \mapsto a]) \wedge$$

$$sumcheck pr (?pr-s') (H, inst (p) [x \mapsto r1], eval (?pr-q) [x \mapsto r1]) r1 (zip vs rs) \wedge$$

$v \neq (\sum \sigma \in \text{subst} (\text{insert } x (\text{set } vs)) H. \text{eval } (p) \sigma) \wedge ?pr-q \neq ?q \rangle$
by (*simp add: P0*)

— dropped condition

also have $\langle \dots \leq ?prob \{r1\#rs \mid r1 \text{ rs.}$
 $\text{deg } (?pr-q) \leq \text{deg } (p) \wedge \text{vars } (?pr-q) \subseteq \{x\} \wedge$
 $v = (\sum a \in H. \text{eval } (?pr-q) [x \mapsto a]) \wedge$
 $\text{sumcheck } pr (?pr-s^\wedge) (H, \text{inst } (p) [x \mapsto r1], \text{eval } (?pr-q) [x \mapsto r1]) r1 (\text{zip } vs \text{ rs}) \wedge$
 $?pr-q \neq ?q \rangle$
by(*intro prob-mono*) (*auto*)

also have $\langle \dots =$
 $?prob \{r1\#rs \mid r1 \text{ rs.}$
 $\text{deg } (?pr-q) \leq \text{deg } (p) \wedge \text{vars } (?pr-q) \subseteq \{x\} \wedge$
 $v = (\sum a \in H. \text{eval } (?pr-q) [x \mapsto a]) \wedge$
 $\text{sumcheck } pr (?pr-s^\wedge) (H, \text{inst } (p) [x \mapsto r1], \text{eval } (?pr-q) [x \mapsto r1]) r1 (\text{zip } vs \text{ rs}) \wedge$
 $?pr-q \neq ?q \wedge \text{eval } (?pr-q) [x \mapsto r1] = \text{eval } (?q) [x \mapsto r1] \rangle +$
 $?prob \{r1\#rs \mid r1 \text{ rs.}$
 $\text{deg } (?pr-q) \leq \text{deg } (p) \wedge \text{vars } (?pr-q) \subseteq \{x\} \wedge$
 $v = (\sum a \in H. \text{eval } (?pr-q) [x \mapsto a]) \wedge$
 $\text{sumcheck } pr (?pr-s^\wedge) (H, \text{inst } (p) [x \mapsto r1], \text{eval } (?pr-q) [x \mapsto r1]) r1 (\text{zip } vs \text{ rs}) \wedge$
 $?pr-q \neq ?q \wedge \text{eval } (?pr-q) [x \mapsto r1] \neq \text{eval } (?q) [x \mapsto r1] \rangle$
by(*intro prob-disjoint-cases*) (*auto*)

also have $\langle \dots \leq \text{real } d / \text{real } \text{CARD}('a) +$
 $(\text{real } (\text{length } vs) * \text{real } d) / \text{real } \text{CARD}('a) \rangle$
by (*intro add-mono RP-left RP-right*)

also have $\langle \dots = (1 + \text{real } (\text{length } vs)) * \text{real } d / \text{real } \text{CARD}('a) \rangle$
by (*metis add-divide-distrib mult-Suc of-nat-Suc of-nat-add of-nat-mult*)

finally show *?case by simp*

qed

corollary soundness:

assumes

$(H, p, v) \notin \text{Sumcheck}$

$\text{vars } p = \text{set } vs$ **and**

$\text{distinct } vs$ **and**

$H \neq \{\}$

shows

measure-pmf.prob

$(\text{pmf-of-set } (\text{tuples } \text{UNIV } (\text{arity } p)))$

$\{rs. \text{sumcheck } pr s (H, p, v) r (\text{zip } vs \text{ rs})\}$

$\leq \text{real } (\text{arity } p) * \text{real } (\text{deg } p) / \text{real } (\text{CARD}('a))$

using *assms*

proof —

have $*$: $\langle \text{arity } p = \text{length } vs \rangle$ **using** $\langle \text{vars } p = \text{set } vs \rangle$ $\langle \text{distinct } vs \rangle$

by (*simp add: arity-def distinct-card*)

have *measure-pmf.prob* $(\text{pmf-of-set } (\text{tuples } \text{UNIV } (\text{arity } p)))$

$\{rs. \text{sumcheck } pr s (H, p, v) r (\text{zip } vs \text{ rs})\} =$

```

      measure-pmf.prob (pmf-of-set (tuples UNIV (arity p)))
      {rs. sumcheck pr s (H, p, v) r (zip vs rs) ∧ (H, p, v) ∉ Sumcheck}
using ⟨(H, p, v) ∉ Sumcheck⟩
by (intro prob-cong) (auto)
also have ... ≤ real (arity p) * real (deg p) / real (CARD('a)) using assms(2-) *
by (auto simp add: Sumcheck-def intro!: soundness-inductive)
finally show ?thesis by simp
qed

end

end

```

8 Sumcheck Protocol as Public-coin Proof

theory *Sumcheck-as-Public-Coin-Proof*

imports

Completeness-Proof

Soundness-Proof

begin

context *multi-variate-polynomial* **begin**

8.1 Property-related definitions

fun *sc-sound-err* :: ('p, 'a, 'b) *sc-inst* \Rightarrow *real* **where**
sc-sound-err (H, p, v) = *real* (arity p) * *real* (deg p) / *real* (CARD('a))

fun *sc-compl-assm* **where**
sc-compl-assm vs ps (H, p, v) xs \longleftrightarrow
 set xs = vars p \wedge distinct xs \wedge H \neq {}

fun *sc-sound-assm* **where**
sc-sound-assm vs ps (H, p, v) xs \longleftrightarrow
 set xs = vars p \wedge distinct xs \wedge H \neq {}

8.2 Public coin proof locale interpretation

sublocale

scp: public-coin-proof-strong-props

sc-ver0 sc-ver1 Sumcheck honest-prover sc-sound-err sc-compl-assm sc-sound-assm

proof

fix I :: ('p, 'a, 'b) *sc-inst* **and**
 vs :: *unit* **and** ps :: *unit* **and**
 rm :: ('v \times 'a) *list* **and** r :: 'a
assume I \in *Sumcheck* **and** *sc-compl-assm* vs ps I (map fst rm)
then show *sc.prove* vs *honest-prover* ps I r rm
by (cases I) (*simp add: prove-sc-eq-sumcheck completeness*)

next

fix I :: ('p, 'a, 'b) *sc-inst* **and**
 vs :: *unit* **and** ps :: 'ps **and**
 r :: 'a **and** rs :: 'a *list* **and** xs :: 'v *list* **and** pr
assume I \notin *Sumcheck* **and** *sc-sound-assm* vs ps I xs
then show
measure-pmf.prob
 (pmf-of-set (tuples UNIV (length xs)))
 {rs. *sc.prove* vs pr ps I r (zip xs rs)}
 \leq *sc-sound-err* I
proof (cases I)
case (fields H p v)
have length xs = arity p **using** \langle *sc-sound-assm* vs ps I xs \rangle *fields*
by (*auto simp add: arity-def distinct-card dest: sym*)
then show ?thesis **using** \langle I \notin *Sumcheck* \rangle \langle *sc-sound-assm* vs ps I xs \rangle *fields*
by (*auto simp add: prove-sc-eq-sumcheck intro: soundness*)

qed

qed

end — context *multi-variate-polynomial*

end

9 Instantiation for Multivariate Polynomials

```
theory Polynomial-Instantiation
  imports
    Polynomials.More-MPoly-Type
begin
```

NOTE: if considered to be useful enough, the definitions and lemmas in this theory could be moved to the theory *Polynomials.More-MPoly-Type*.

Define instantiation of mpoly's. The conditions (\neq) $(1::'a)$ and (\neq) $(0::'a)$ in the sets being multiplied or added over are needed to prove the correspondence with evaluation: a full instance corresponds to an evaluation (see lemma below).

9.1 Instantiation of monomials

```
type-synonym ('a, 'b) subst = 'a  $\rightarrow$  'b
```

lift-definition

```
inst-monom-coeff ::  $\langle (nat \Rightarrow_0 nat) \Rightarrow (nat, 'a) subst \Rightarrow 'a::comm-semiring-1 \rangle$ 
is  $\langle \lambda m \sigma. (\prod v \mid v \in dom \sigma \wedge the (\sigma v) \wedge m v \neq 1. the (\sigma v) \wedge m v) \rangle$  .
```

lift-definition

```
inst-monom-resid ::  $\langle (nat \Rightarrow_0 nat) \Rightarrow (nat, 'a) subst \Rightarrow (nat \Rightarrow_0 nat) \rangle$ 
is  $\langle \lambda m \sigma v. m v \text{ when } v \notin dom \sigma \rangle$ 
by (metis (mono-tags, lifting) finite-subset mem-Collect-eq subsetI zero-when)
```

```
lemmas inst-monom-defs = inst-monom-coeff-def inst-monom-resid-def
```

```
lemma lookup-inst-monom-resid:
```

```
shows  $\langle lookup (inst-monom-resid m \sigma) v = (if v \in dom \sigma \text{ then } 0 \text{ else } lookup m v) \rangle$ 
by transfer simp
```

9.2 Instantiation of polynomials

definition

```
inst-fun ::  $\langle ((nat \Rightarrow_0 nat) \Rightarrow 'a) \Rightarrow (nat, 'a) subst \Rightarrow (nat \Rightarrow_0 nat) \Rightarrow 'a::comm-semiring-1 \rangle$  where
 $\langle inst-fun p \sigma = (\lambda m. (\sum m' \mid inst-monom-resid m' \sigma = m \wedge p m' * inst-monom-coeff m' \sigma \neq 0. p m' * inst-monom-coeff m' \sigma)) \rangle$ 
```

```
lemma finite-inst-fun-keys:
```

```
assumes  $\langle finite \{m. p m \neq 0\} \rangle$ 
shows  $\langle finite \{m. (\sum m' \mid inst-monom-resid m' \sigma = m \wedge p m' \neq 0 \wedge inst-monom-coeff m' \sigma \neq 0. p m' * inst-monom-coeff m' \sigma) \neq 0\} \rangle$ 
```

```
proof -
```

```
from  $\langle finite \{m. p m \neq 0\} \rangle$ 
```

```
have  $\langle finite ((\lambda m'. inst-monom-resid m' \sigma) \{x. p x \neq 0\}) \rangle$  by auto
```

```
moreover
```

```
have  $\langle \{m. (\sum m' \mid inst-monom-resid m' \sigma = m \wedge p m' \neq 0 \wedge inst-monom-coeff m' \sigma \neq 0. p m' * inst-monom-coeff m' \sigma) \neq 0\} \rangle$ 
```

```
subseteq  $\langle (\lambda m'. inst-monom-resid m' \sigma) \{m. p m \neq 0\} \rangle$ 
```

```
by (auto elim: sum.not-neutral-contains-not-neutral)
```

```
ultimately show ?thesis
```

by (auto elim: finite-subset)
qed

lemma *finite-inst-fun-keys-ext*:

assumes $\langle \text{finite } \{m. p\ m \neq 0\} \rangle$

shows $\text{finite } \{a. (\sum m' \mid \text{inst-monom-resid } m' \sigma = a \wedge p\ m' \neq 0 \wedge \text{inst-monom-coeff } m' \sigma \neq 0. \\ p\ m' * \text{inst-monom-coeff } m' \sigma * (\prod aa. \text{the } (\varrho\ aa) \wedge \text{lookup } (\text{inst-monom-resid } m' \sigma)\ aa)) \neq 0\}$

proof –

from $\langle \text{finite } \{m. p\ m \neq 0\} \rangle$

have $\langle \text{finite } ((\lambda m'. \text{inst-monom-resid } m' \sigma) \{x. p\ x \neq 0\}) \rangle$ by auto

moreover

have $\langle \{m. (\sum m' \mid \text{inst-monom-resid } m' \sigma = m \wedge p\ m' \neq 0 \wedge \text{inst-monom-coeff } m' \sigma \neq 0. \\ p\ m' * \text{inst-monom-coeff } m' \sigma * \\ (\prod aa. \text{the } (\varrho\ aa) \wedge \text{lookup } (\text{inst-monom-resid } m' \sigma)\ aa)) \neq 0\}$

$\subseteq (\lambda m'. \text{inst-monom-resid } m' \sigma) \{m. p\ m \neq 0\} \rangle$

by (auto elim: sum.not-neutral-contains-not-neutral)

ultimately show ?thesis

by (auto elim: finite-subset)

qed

lift-definition

inst-aux :: $\langle ((\text{nat} \Rightarrow_0 \text{nat}) \Rightarrow_0 'a) \Rightarrow (\text{nat}, 'a)\ \text{subst} \Rightarrow (\text{nat} \Rightarrow_0 \text{nat}) \Rightarrow_0 'a::\text{semidom} \rangle$

is *inst-fun*

by (auto simp add: inst-fun-def intro: finite-inst-fun-keys)

lift-definition *inst* :: $\langle 'a\ \text{mpoly} \Rightarrow (\text{nat}, 'a::\text{semidom})\ \text{subst} \Rightarrow 'a\ \text{mpoly} \rangle$

is *inst-aux* .

lemmas *inst-defs* = *inst-def inst-aux-def inst-fun-def*

9.3 Full instantiation corresponds to evaluation

lemma *dom-Some*: $\langle \text{dom } (\text{Some } o\ f) = \text{UNIV} \rangle$

by (simp add: dom-def)

lemma *inst-full-eq-insertion*:

fixes $p :: \langle 'a::\text{semidom} \rangle\ \text{mpoly}$ and $\sigma :: \langle \text{nat} \Rightarrow 'a \rangle$

shows $\langle \text{inst } p\ (\text{Some } o\ \sigma) = \text{Const } (\text{insertion } \sigma\ p) \rangle$

proof *transfer*

fix $p :: \langle (\text{nat} \Rightarrow_0 \text{nat}) \Rightarrow_0 'a \rangle$ and $\sigma :: \langle \text{nat} \Rightarrow 'a \rangle$

show $\langle \text{inst-aux } p\ (\text{Some } o\ \sigma) = \text{Const}_0\ (\text{insertion-aux } \sigma\ p) \rangle$

unfolding *poly-mapping-eq-iff*

apply (simp add: Const₀-def inst-aux.rep-eq inst-fun-def inst-monom-defs

Poly-Mapping.single.rep-eq insertion-aux.rep-eq insertion-fun-def)

apply (rule ext)

subgoal for m

by (cases $m = 0$)

(*simp-all* add: *Sum-any.expand-set Prod-any.expand-set dom-Some*)

done

qed

end

10 Roots Bound for Univariate Polynomials

```

theory Univariate-Roots-Bound
  imports
    HOL-Computational-Algebra.Polynomial
begin

```

NOTE: if considered to be useful enough, the lemmas in this theory could be moved to the theory *HOL-Computational-Algebra.Polynomial*.

10.1 Basic lemmas

```

lemma finite-non-zero-coeffs:  $\langle \text{finite } \{n. \text{poly.coeff } p \ n \neq 0\} \rangle$ 
  using MOST-coeff-eq-0 eventually-cofinite
  by fastforce

```

Univariate degree in terms of *Max*

```

lemma poly-degree-eq-Max-non-zero-coeffs:
   $\text{degree } p = \text{Max } (\text{insert } 0 \ \{n. \text{poly.coeff } p \ n \neq 0\})$ 
  by (intro le-antisym degree-le) (auto simp add: finite-non-zero-coeffs le-degree)

```

10.2 Univariate roots bound

The number of roots of a product of polynomials is bounded by the sum of the number of roots of each.

```

lemma card-poly-mult-roots:
  fixes  $p :: 'a::\{\text{comm-ring-1, ring-no-zero-divisors}\}$  poly
  and  $q :: 'a::\{\text{comm-ring-1, ring-no-zero-divisors}\}$  poly
  assumes  $p \neq 0$  and  $q \neq 0$ 
  shows  $\text{card } \{x. \text{poly } p \ x * \text{poly } q \ x = 0\} \leq \text{card } \{x. \text{poly } p \ x = 0\} + \text{card } \{x. \text{poly } q \ x = 0\}$ 
proof —
  have  $\text{card } \{x . \text{poly } p \ x * \text{poly } q \ x = 0\} \leq \text{card } (\{x . \text{poly } p \ x = 0\} \cup \{x . \text{poly } q \ x = 0\})$ 
  by (auto simp add: poly-roots-finite assms intro!: card-mono)
  also have  $\dots \leq \text{card } \{x . \text{poly } p \ x = 0\} + \text{card } \{x . \text{poly } q \ x = 0\}$ 
  by(auto simp add: Finite-Set.card-Un-le)
  finally show ?thesis .
qed

```

A univariate polynomial p has at most $\text{deg } p$ roots.

```

lemma univariate-roots-bound:
  fixes  $p :: \langle 'a::\text{idom } \text{poly} \rangle$ 
  assumes  $\langle p \neq 0 \rangle$ 
  shows  $\langle \text{card } \{x. \text{poly } p \ x = 0\} \leq \text{degree } p \rangle$ 
  using assms
proof (induction degree p arbitrary: p rule: nat-less-induct)
  case 1
  then show ?case
  proof(cases  $\exists r. \text{poly } p \ r = 0$ )
    case True — A root exists

```

— Get root r of polynomial and write $p = [:- r, 1] \wedge \text{order } r \ p * q$ for some q .

```

then obtain  $r$  where  $\text{poly } p \ r = 0$  by(auto)
let  $?xr = [-\ r, 1:] \wedge \text{order } r \ p$ 
obtain  $q$  where  $p = ?xr * q$  using order-decomp  $\langle p \neq 0 \rangle$  by(auto)

— Useful facts about  $q$  and  $[-\ r, 1::'a:] \text{order } r \ p$ 
have  $q \neq 0$  using  $\langle p = ?xr * q \rangle \langle p \neq 0 \rangle$  by(auto)
have  $?xr \neq 0$  by(simp)
have  $\text{degree } ?xr > 0$  using  $\langle ?xr \neq 0 \rangle \langle p \neq 0 \rangle \langle \text{poly } p \ r = 0 \rangle$ 
  by (simp add: degree-power-eq order-root)
have  $\text{degree } q < \text{degree } p$ 
  using  $\langle ?xr \neq 0 \rangle \langle q \neq 0 \rangle \langle p = ?xr * q \rangle \langle \text{degree } ?xr > 0 \rangle$ 
    degree-mult-eq[where  $p = ?xr$  and  $q = q$ ]
  by (simp)
have  $x\text{-roots: card } \{r. \text{poly } ?xr \ r = 0\} = 1$  using  $\langle p \neq 0 \rangle \langle \text{poly } p \ r = 0 \rangle$ 
  by(simp add: order-root)
have  $q\text{-roots: card } \{r. \text{poly } q \ r = 0\} \leq \text{degree } q$  using  $\langle q \neq 0 \rangle \langle \text{degree } q < \text{degree } p \rangle 1$ 
  by (simp)

— Final bound
have  $\text{card } \{r. \text{poly } p \ r = 0\} \leq \text{degree } p$ 
  using  $\langle p = ?xr * q \rangle \langle q \neq 0 \rangle \langle ?xr \neq 0 \rangle \langle \text{degree } q < \text{degree } p \rangle$ 
    poly-mult[where  $p = ?xr$  and  $q = q$ ]
    card-poly-mult-roots[where  $p = ?xr$  and  $q = q$ ]
    x-roots q-roots
  by (simp)
then show ?thesis .
next
  case False — No root exists
  then show ?thesis by simp
qed
qed

end

```


11 Roots Bound for Multivariate Polynomials of Arity at Most One

```

theory Roots-Bounds
  imports
    Polynomials.MPoly-Type-Univariate
    Univariate-Roots-Bound
begin

```

NOTE: if considered to be useful enough, the lemmas in this theory could be moved to the theory *Polynomials.MPoly-Type-Univariate*.

11.1 Lemmas connecting univariate and multivariate polynomials

11.1.1 Basic lemmas

```

lemma mpoly-to-poly-zero-iff:
  fixes p::'a::comm-monoid-add mpoly
  assumes <vars p ⊆ {v}>
  shows mpoly-to-poly v p = 0 ⟷ p = 0
  by (metis assms mpoly-to-poly-inverse poly-to-mpoly0 poly-to-mpoly-inverse)

```

```

lemma keys-monom-subset-vars:
  fixes p::'a::zero mpoly
  assumes <m ∈ keys (mapping-of p)>
  shows keys m ⊆ vars p
  using assms
  by (auto simp add: vars-def)

```

```

lemma sum-lookup-keys-eq-lookup:
  fixes p::'a::zero mpoly
  assumes <m ∈ keys (mapping-of p)> and <vars p ⊆ {v}>
  shows sum (lookup m) (keys m) = lookup m v
  using assms
  by (auto simp add: subset-singleton-iff dest!: keys-monom-subset-vars)

```

11.1.2 Total degree corresponds to degree for polynomials of arity at most one

```

lemma poly-degree-eq-mpoly-degree:
  fixes p::'a::comm-monoid-add mpoly
  assumes <vars p ⊆ {v}>
  shows degree (mpoly-to-poly v p) = MPoly-Type.degree p v
  using assms
proof -
  have *: ⋀n. MPoly-Type.coeff p (Poly-Mapping.single v n) ≠ 0
    ⟷ (∃ m∈keys (mapping-of p). n = lookup m v)
  by (metis (no-types, opaque-lifting) Diff-eq-empty-iff Diff-insert add-0 keys-eq-empty
    keys-monom-subset-vars lookup-single-eq remove-key-keys remove-key-sum
    singleton-insert-inj-eq' coeff-keys[symmetric] assms)

  have degree (mpoly-to-poly v p)
    = Max (insert 0 {n. MPoly-Type.coeff p (Poly-Mapping.single v n) ≠ 0})
  by (simp add: poly-degree-eq-Max-non-zero-coeffs)

```

also have $\dots = MPoly\text{-Type.degree } p \ v$
 by (simp add: degree.rep-eq image-def *)
 finally show ?thesis .

qed

lemma mpoly-degree-eq-total-degree:

fixes $p::'a::zero \ mpoly$

assumes $\langle vars \ p \subseteq \{v\} \rangle$

shows $MPoly\text{-Type.degree } p \ v = total\text{-degree } p$

using *assms*

by (auto simp add: MPoly-Type.degree-def total-degree-def sum-lookup-keys-eq-lookup)

corollary poly-degree-eq-total-degree:

fixes $p::'a::comm\text{-monoid-add} \ mpoly$

assumes $\langle vars \ p \subseteq \{v\} \rangle$

shows $degree \ (mpoly\text{-to-poly } v \ p) = total\text{-degree } p$

using *assms*

by (simp add: poly-degree-eq-mpoly-degree mpoly-degree-eq-total-degree)

11.2 Roots bound for univariate polynomials of type $'a \ mpoly$

lemma univariate-mpoly-roots-bound:

fixes $p::'a::idom \ mpoly$

assumes $\langle vars \ p \subseteq \{v\} \rangle \ \langle p \neq 0 \rangle$

shows $\langle card \ \{x. \ insertion \ (\lambda v. \ x) \ p = 0\} \leq total\text{-degree } p \rangle$

using *assms* univariate-roots-bound[of mpoly-to-poly v p, unfolded poly-eq-insertion[OF $\langle vars \ p \subseteq \{v\} \rangle$]]

by (auto simp add: poly-degree-eq-total-degree mpoly-to-poly-zero-iff)

end

12 Multivariate Polynomials: Instance

```

theory Concrete-Multivariate-Polynomials
  imports
    ../Generalized-Sumcheck-Protocol/Sumcheck-as-Public-Coin-Proof
    Polynomial-Instantiation
    Roots-Bounds
begin

declare total-degree-zero [simp del]

```

12.1 Auxiliary lemmas

```

lemma card-indep-bound:
  assumes  $P \implies \text{card } \{x. Q\ x\} \leq d$ 
  shows  $\text{card } \{x. P \wedge Q\ x\} \leq d$ 
  using assms
  by (cases P) auto

```

```

lemma sum-point-neq-zero [simp]:  $(\sum x' \mid x' = x \wedge f\ x' \neq 0. f\ x')$  =  $f\ x$ 
proof –
  have  $\langle (\sum x' \mid x' = x \wedge f\ x' \neq 0. f\ x') = (\sum x' \mid x' = x \wedge f\ x \neq 0. f\ x') \rangle$ 
    by (intro sum.cong) auto
  also have  $\langle \dots = f\ x \rangle$ 
    by (cases f\ x = 0) (simp-all)
  finally show ?thesis .
qed

```

12.2 Proving the assumptions of the locale

12.2.1 Variables

```

lemma vars-zero:  $\langle \text{vars } 0 = \{\} \rangle$ 
  by (simp add: vars-def zero-mpoly.rep-eq)

```

```

lemma vars-inst:  $\langle \text{vars } (\text{inst } p\ \sigma) \subseteq \text{vars } p - \text{dom } \sigma \rangle$ 
  by (auto simp add: vars-def inst-defs keys-def MPoly-inverse
    finite-inst-fun-keys lookup-inst-monom-resid
    elim!: sum.not-neutral-contains-not-neutral split: if-splits)

```

— Lemmas for to translate the roots bound to the format of the locale assumption.

```

lemma vars-minus:  $\langle \text{vars } p = \text{vars } (-p) \rangle$ 
  by(simp add: vars-def uminus-mpoly.rep-eq)

```

```

lemma vars-subtr:
  fixes  $p\ q :: \langle 'a::\text{comm-ring } \text{mpoly} \rangle$ 
  shows  $\langle \text{vars } (p - q) \subseteq \text{vars } p \cup \text{vars } q \rangle$ 
  by(simp add: vars-add[where ?p1.0 = p and ?p2.0 = -q, simplified] vars-minus[where p = q])

```

12.2.2 Degree

```

abbreviation deg ::  $\langle ('a::\text{zero})\ \text{mpoly} \Rightarrow \text{nat} \rangle$  where

```

⟨deg p ≡ total-degree p⟩

— We show the assumptions *multi-variate-polynomial.deg-zero*, *multi-variate-polynomial.deg-add* and *multi-variate-polynomial.deg-inst*.

lemma *deg-zero*: ⟨deg 0 = 0⟩ **by** (fact total-degree-zero)

lemma *deg-add*: ⟨deg (p + q) ≤ max (deg p) (deg q)⟩

proof –

have ⟨deg (p + q) = Max (insert 0 ((λx. sum (lookup x) (keys x)) ‘ keys (mapping-of p + mapping-of q)))⟩

by (simp add: total-degree.rep-eq plus-mpoly.rep-eq)

also have ⟨... ≤ Max (insert 0 ((λx. sum (lookup x) (keys x)) ‘ (keys (mapping-of p) ∪ keys (mapping-of q))))⟩

by (intro Max-mono Set.insert-mono image-mono Poly-Mapping.keys-add) (auto)

also have ⟨... = Max ((insert 0 ((λx. sum (lookup x) (keys x)) ‘ keys (mapping-of p))) ∪ (insert 0 ((λx. sum (lookup x) (keys x)) ‘ keys (mapping-of q))))⟩

by (simp add: image-Un)

also have ⟨... = max (Max (insert 0 ((λx. sum (lookup x) (keys x)) ‘ keys (mapping-of p))) (Max (insert 0 ((λx. sum (lookup x) (keys x)) ‘ keys (mapping-of q))))⟩

by (intro Max-Un) (auto)

also have ⟨... = max (deg p) (deg q)⟩

by (simp add: total-degree.rep-eq)

finally show ?thesis .

qed

lemma *deg-inst*: ⟨deg (inst p σ) ≤ deg p⟩

proof (transfer)

fix p :: ⟨(nat ⇒₀ nat) ⇒₀ 'a⟩ **and** σ :: (nat, 'a) subst

show ⟨Max (insert 0 ((λm. sum (lookup m) (keys m)) ‘ keys (inst-aux p σ)))

≤ Max (insert 0 ((λm. sum (lookup m) (keys m)) ‘ keys p))⟩

by (auto simp add: keys-def inst-defs finite-inst-fun-keys lookup-inst-monom-resid elim!: sum.not-neutral-contains-not-neutral)

(fastforce simp add: Max-ge-iff intro!: disjI2 intro: sum-mono2)

qed

— Lemmas for translating the roots bound to the format of the locale assumption.

lemma *deg-minus*: ⟨deg p = deg (−p)⟩

by(auto simp add: total-degree-def uminus-mpoly.rep-eq)

lemma *deg-subtr*:

fixes p q :: ⟨'a::comm-ring mpoly⟩

shows ⟨deg (p − q) ≤ max (deg p) (deg q)⟩

by(auto simp add: deg-add[where p = p and q = −q, simplified] deg-minus[where p = q])

12.2.3 Evaluation

abbreviation *eval* :: ⟨'a mpoly ⇒ (nat, 'a) subst ⇒ ('a::comm-semiring-1)⟩ **where**

⟨eval p σ ≡ insertion (the o σ) p⟩

— We show the assumptions *multi-variate-polynomial.eval-zero*, *multi-variate-polynomial.eval-add* and *multi-variate-polynomial.eval-inst*.

lemma *eval-zero*: $\langle \text{eval } 0 \ \sigma = 0 \rangle$
by (*fact insertion-zero*)

lemma *eval-add*: $\langle \text{vars } p \cup \text{vars } q \subseteq \text{dom } \sigma \implies \text{eval } (p + q) \ \sigma = \text{eval } p \ \sigma + \text{eval } q \ \sigma \rangle$
by (*intro insertion-add*)

— evaluation and instantiation

lemma *eval-inst*: $\langle \text{eval } (\text{inst } p \ \sigma) \ \varrho = \text{eval } p \ (\varrho ++ \sigma) \rangle$

proof (*transfer, transfer*)

fix $p :: \langle (\text{nat} \Rightarrow_0 \text{nat}) \Rightarrow 'a \rangle$ **and** $\sigma \ \varrho :: \langle (\text{nat}, 'a) \text{ subst} \rangle$

assume *fin*: $\langle \text{finite } \{m. p \ m \neq 0\} \rangle$

show $\langle \text{insertion-fun } (\text{the } \circ \ \varrho) \ (\text{inst-fun } p \ \sigma) = \text{insertion-fun } (\text{the } \circ \ \varrho ++ \sigma) \ p \rangle$

proof —

let $?mon = \langle \lambda \sigma \ m \ v. \text{the } (\sigma \ v) \wedge \text{lookup } m \ v \rangle$

have $\langle x \wedge \text{lookup } m \ v \neq 1 \implies v \in \text{keys } m \rangle$ **for** $x :: 'a$ **and** v **and** $m :: \text{nat} \Rightarrow_0 \text{nat}$

using *zero-less-iff-neq-zero* **by** (*fastforce simp add: in-keys-iff*)

then have *exp-fin*: $\langle \text{finite } \{v. P \ v \wedge f \ v \wedge \text{lookup } m \ v \neq 1\} \rangle$

for $f :: \text{nat} \Rightarrow 'a$ **and** $m :: \text{nat} \Rightarrow_0 \text{nat}$ **and** $P :: \text{nat} \Rightarrow \text{bool}$

by (*auto intro: finite-subset[where B=keys m]*)

note *fin-simps* [*simp*] = *fin this this[where P1= λ -. True, simplified]*

note *map-add-simps* [*simp*] = *map-add-dom-app-simps(1,3)*

have $\langle \text{insertion-fun } (\text{the } \circ \ \varrho) \ (\text{inst-fun } p \ \sigma) =$

$(\sum m. (\sum m' \mid \text{inst-monom-resid } m' \ \sigma = m \wedge p \ m' \neq 0 \wedge \text{inst-monom-coeff } m' \ \sigma \neq 0. \\ p \ m' * \text{inst-monom-coeff } m' \ \sigma) * (\prod v. ?mon \ \varrho \ m \ v)) \rangle$

by (*simp add: insertion-fun-def inst-fun-def*)

also have $\langle \dots =$

$(\sum m. (\sum m' \mid \text{inst-monom-resid } m' \ \sigma = m \wedge p \ m' \neq 0 \wedge \text{inst-monom-coeff } m' \ \sigma \neq 0. \\ p \ m' * \text{inst-monom-coeff } m' \ \sigma * (\prod v. ?mon \ \varrho \ m \ v))) \rangle$

by (*intro Sum-any.cong*) (*simp add: sum-distrib-right*)

also have $\langle \dots =$

$(\sum m. (\sum m' \mid \text{inst-monom-resid } m' \ \sigma = m \wedge p \ m' \neq 0 \wedge \text{inst-monom-coeff } m' \ \sigma \neq 0. \\ p \ m' * \text{inst-monom-coeff } m' \ \sigma * (\prod v \mid v \notin \text{dom } \sigma \wedge ?mon \ \varrho \ m' \ v \neq 1. ?mon \ \varrho \ m' \ v))) \rangle$

by (*intro Sum-any.cong sum.cong*)

(*auto simp add: lookup-inst-monom-resid Prod-any.expand-set intro: arg-cong*)

also have $\langle \dots =$

$(\sum m. (\sum m' \mid \text{inst-monom-resid } m' \ \sigma = m \wedge p \ m' \neq 0 \wedge \\ (\prod v \mid v \in \text{dom } \sigma \wedge ?mon \ \sigma \ m' \ v \neq 1. ?mon \ \sigma \ m' \ v) \neq 0. \\ p \ m' *$

$((\prod v \mid v \in \text{dom } \sigma \wedge ?mon \ \sigma \ m' \ v \neq 1. ?mon \ (\varrho ++ \sigma) \ m' \ v) * \\ (\prod v \mid v \notin \text{dom } \sigma \wedge ?mon \ \varrho \ m' \ v \neq 1. ?mon \ (\varrho ++ \sigma) \ m' \ v))) \rangle$

by (*simp add: inst-monom-coeff-def mult.assoc*)

also have $\langle \dots =$

$(\sum m. (\sum m' | \text{inst-monom-resid } m' \sigma = m \wedge p \ m' \neq 0 \wedge$
 $(\prod v | v \in \text{dom } \sigma \wedge ?\text{mon } \sigma \ m' \ v \neq 1. ?\text{mon } \sigma \ m' \ v) \neq 0 \wedge$
 $(\prod v | v \notin \text{dom } \sigma \wedge ?\text{mon } \varrho \ m' \ v \neq 1. ?\text{mon } \varrho \ m' \ v) \neq 0.$
 $p \ m' \ *$
 $((\prod v | v \in \text{dom } \sigma \wedge ?\text{mon } \sigma \ m' \ v \neq 1. ?\text{mon } (\varrho \ ++ \ \sigma) \ m' \ v) *$
 $(\prod v | v \notin \text{dom } \sigma \wedge ?\text{mon } \varrho \ m' \ v \neq 1. ?\text{mon } (\varrho \ ++ \ \sigma) \ m' \ v))))\rangle$
by (*intro Sum-any.cong sum.mono-neutral-right*) (*auto*)

also have $\langle \dots =$

$(\sum m. (\sum m' | \text{inst-monom-resid } m' \sigma = m \wedge p \ m' \neq 0 \wedge$
 $(\prod v | v \in \text{dom } \sigma \wedge ?\text{mon } \sigma \ m' \ v \neq 1. ?\text{mon } \sigma \ m' \ v) \neq 0 \wedge$
 $(\prod v | v \notin \text{dom } \sigma \wedge ?\text{mon } \varrho \ m' \ v \neq 1. ?\text{mon } \varrho \ m' \ v) \neq 0.$
 $p \ m' \ *$
 $(\prod v | v \in \text{dom } \sigma \wedge ?\text{mon } \sigma \ m' \ v \neq 1 \vee v \notin \text{dom } \sigma \wedge ?\text{mon } \varrho \ m' \ v \neq 1. ?\text{mon } (\varrho \ ++ \ \sigma)$
 $m' \ v))))\rangle$

by (*subst prod.union-disjoint[symmetric]*)

(*auto intro!: Sum-any.cong sum.cong prod.cong intro: arg-cong*)

also have $\langle \dots =$

$(\sum m. (\sum m' | \text{inst-monom-resid } m' \sigma = m \wedge p \ m' \neq 0 \wedge$
 $(\prod v | v \in \text{dom } \sigma \wedge ?\text{mon } \sigma \ m' \ v \neq 1. ?\text{mon } \sigma \ m' \ v) \neq 0 \wedge$
 $(\prod v | v \notin \text{dom } \sigma \wedge ?\text{mon } \varrho \ m' \ v \neq 1. ?\text{mon } \varrho \ m' \ v) \neq 0.$
 $p \ m' \ * (\prod v. ?\text{mon } (\varrho \ ++ \ \sigma) \ m' \ v))))\rangle$

apply (*intro Sum-any.cong sum.cong arg-cong[where f=(*) x for x]*, *simp*)

apply (*simp add: Prod-any.expand-set*)

apply (*intro prod.cong, simp-all*)

by (*metis (no-types, opaque-lifting) map-add-dom-app-simps(1,3)*)

also have $\langle \dots =$

$(\sum m. (\sum m' | \text{inst-monom-resid } m' \sigma = m \wedge p \ m' \neq 0 \wedge (\prod v. ?\text{mon } (\varrho \ ++ \ \sigma) \ m' \ v) \neq 0.$
 $p \ m' \ * (\prod v. ?\text{mon } (\varrho \ ++ \ \sigma) \ m' \ v))))\rangle$

apply (*intro Sum-any.cong sum.mono-neutral-right, simp-all*)

apply (*safe, simp-all*)

— *fixme: cannot get auto/fastforce to do instantiations below*

subgoal for $m \ v \ z$

by (*auto dest: Prod-any-not-zero[rotated, where a=v]*)

subgoal for $m' \ v$

by (*auto simp add: domIff dest: Prod-any-not-zero[rotated, where a=v]*)

done

also have $\langle \dots =$

$(\sum m. (\text{sum}$
 $(\lambda m'. p \ m' \ * (\prod v. ?\text{mon } (\varrho \ ++ \ \sigma) \ m' \ v))$
 $\{m' \in \{m'. p \ m' \neq 0 \wedge (\prod v. ?\text{mon } (\varrho \ ++ \ \sigma) \ m' \ v) \neq 0\}.$
 $\text{inst-monom-resid } m' \ \sigma = m\}))\rangle$

by (*intro Sum-any.cong sum.cong*) (*auto*)

also have $\langle \dots =$

$(\sum m \in (\lambda m'. \text{inst-monom-resid } m' \ \sigma) \ \{m'. p \ m' \neq 0 \wedge (\prod v. \text{the } ((\varrho \ ++ \ \sigma) \ v) \ \hat{\wedge} \ \text{lookup } m'$
 $v) \neq 0\}.$

(*sum*

$(\lambda m'. p \ m' \ * (\prod v. ?\text{mon } (\varrho \ ++ \ \sigma) \ m' \ v))$

$\{m' \in \{m'. p \ m' \neq 0 \wedge (\prod v. ?\text{mon } (\varrho \ ++ \ \sigma) \ m' \ v) \neq 0\}.$

$inst\text{-}monom\text{-}resid\ m'\ \sigma = m\})\rangle$
by (*intro Sum-any.expand-superset*) (*auto elim: sum.not-neutral-contains-not-neutral*)
also have $\langle \dots = (\sum m. p\ m * (\prod v. ?mon\ (\varrho\ ++\ \sigma)\ m\ v)) \rangle$
by (*subst Sum-any.expand-set, subst sum.group*) (*auto*)
also have $\langle \dots = insertion\text{-}fun\ (the\ \circ\ \varrho\ ++\ \sigma)\ p \rangle$
by (*simp add: insertion-fun-def*)
finally show *?thesis* .
qed
qed

— Lemmas for translating the roots bound to the format of the locale assumption.

lemma *eval-minus*:
fixes $p :: \langle 'a::comm\text{-}ring\text{-}1\ mpoly \rangle$
shows $\langle eval\ (-p)\ \sigma = -\ eval\ p\ \sigma \rangle$
using *sum-negf*[**where** $f = \lambda a . (lookup\ (mapping\text{-}of\ p)\ a * (\prod aa. the\ (\sigma\ aa)\ \wedge\ lookup\ a\ aa))$]
by(*auto simp add: uminus-mpoly.rep-eq insertion-def insertion-aux-def insertion-fun-def*)
(smt (verit) Collect-cong Sum-any.expand-set add.inverse-neutral neg-equal-iff-equal)

lemma *eval-subtr*:
fixes $p\ q :: \langle 'a::comm\text{-}ring\text{-}1\ mpoly \rangle$
assumes $\langle vars\ p \subseteq dom\ \sigma \rangle\ \langle vars\ q \subseteq dom\ \sigma \rangle$
shows $\langle eval\ (p - q)\ \sigma = eval\ p\ \sigma - eval\ q\ \sigma \rangle$
using *assms*
by(*auto simp add: vars-minus*[**where** $p = q$] *eval-minus*[**where** $p = q$]
eval-add[**where** $p = p$ **and** $q = -q$, *simplified*])

12.2.4 Roots assumption

lemma *univariate-eval-as-insertion*:
fixes $p::\langle 'a::comm\text{-}ring\text{-}1\ mpoly \rangle$ **and** r
assumes $vars\ p \subseteq \{x\}$
shows $eval\ p\ [x \mapsto r] = insertion\ (\lambda x. r)\ p$
using *assms*
by (*intro insertion-irrelevant-vars*) *auto*

lemma *univariate-mpoly-roots-bound-eval*: — variant using *eval*
fixes $p::'a::idom\ mpoly$
assumes $\langle vars\ p \subseteq \{x\} \rangle\ \langle p \neq 0 \rangle$
shows $\langle card\ \{r. eval\ p\ [x \mapsto r] = 0\} \leq deg\ p \rangle$
using *assms*
by (*simp add: univariate-eval-as-insertion univariate-mpoly-roots-bound*)

lemma *mpoly-roots*:
fixes $p\ q :: \langle 'a::idom\ mpoly \rangle$ **and** $d\ x$
shows $\langle card\ \{r. deg\ p \leq d \wedge vars\ p \subseteq \{x\} \wedge deg\ q \leq d \wedge vars\ q \subseteq \{x\} \wedge$
 $p \neq q \wedge eval\ p\ [x \mapsto r] = eval\ q\ [x \mapsto r]\} \leq d \rangle$
proof (*intro card-indep-bound*)
assume $\langle deg\ p \leq d \rangle\ \langle vars\ p \subseteq \{x\} \rangle\ \langle deg\ q \leq d \rangle\ \langle vars\ q \subseteq \{x\} \rangle\ \langle p \neq q \rangle$
show $\langle card\ \{r. eval\ p\ [x \mapsto r] = eval\ q\ [x \mapsto r]\} \leq d \rangle$
proof —

```

have ⟨card {r. eval p [x ↦ r] = eval q [x ↦ r]} = card {r. eval (p - q) [x ↦ r] = 0}⟩
  using ⟨vars p ⊆ {x}⟩ ⟨vars q ⊆ {x}⟩ by (simp add: eval-subtr)
also have ⟨... ≤ deg (p - q)⟩
  using ⟨vars p ⊆ {x}⟩ ⟨vars q ⊆ {x}⟩ ⟨p ≠ q⟩
  by (intro univariate-mpoly-roots-bound-eval subset-trans[OF vars-subtr]) (auto)
also have ⟨... ≤ d⟩ using ⟨deg p ≤ d⟩ ⟨deg q ≤ d⟩
  by (intro le-trans[OF deg-subtr]) (simp)
finally show ?thesis .
qed
qed

```

12.3 Locale interpretation

Finally, collect all relevant lemmas and instantiate the abstract polynomials locale.

```

lemmas multi-variate-polynomial-lemmas =
  vars-finite vars-zero vars-add vars-inst
  deg-zero deg-add deg-inst
  eval-zero eval-add eval-inst
  mpoly-roots

```

interpretation mpoly:

```

multi-variate-polynomial vars deg :: 'a::{finite, idom} mpoly ⇒ nat eval inst
by (unfold-locales) (auto simp add: multi-variate-polynomial-lemmas)

```

Here are the main results, specialized for type 'a mpoly. The completeness theorem for this type is

```

[[ (?H, ?p, ?v) ∈ mpoly.Sumcheck; vars ?p = set (map fst ?rm);
  distinct (map fst ?rm); ?H ≠ {}]]
⇒ mpoly.sumcheck mpoly.honest-prover ?u (?H, ?p, ?v) ?r ?rm

```

and the soundness theorem reads

```

[[ (?H, ?p, ?v) ∉ mpoly.Sumcheck; vars ?p = set ?vs; distinct ?vs; ?H ≠ {}]]
⇒ measure-pmf.prob (pmf-of-set (tuples UNIV (mpoly.arity ?p)))
  {rs. mpoly.sumcheck ?pr ?s (?H, ?p, ?v) ?r (zip ?vs rs)}
  ≤ real (mpoly.arity ?p) * real (deg ?p) / real CARD(?a)

```

.

end