

# The Sumcheck Protocol

Azucena Garvia, Christoph Sprenger and Jonathan Bootle

May 26, 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.

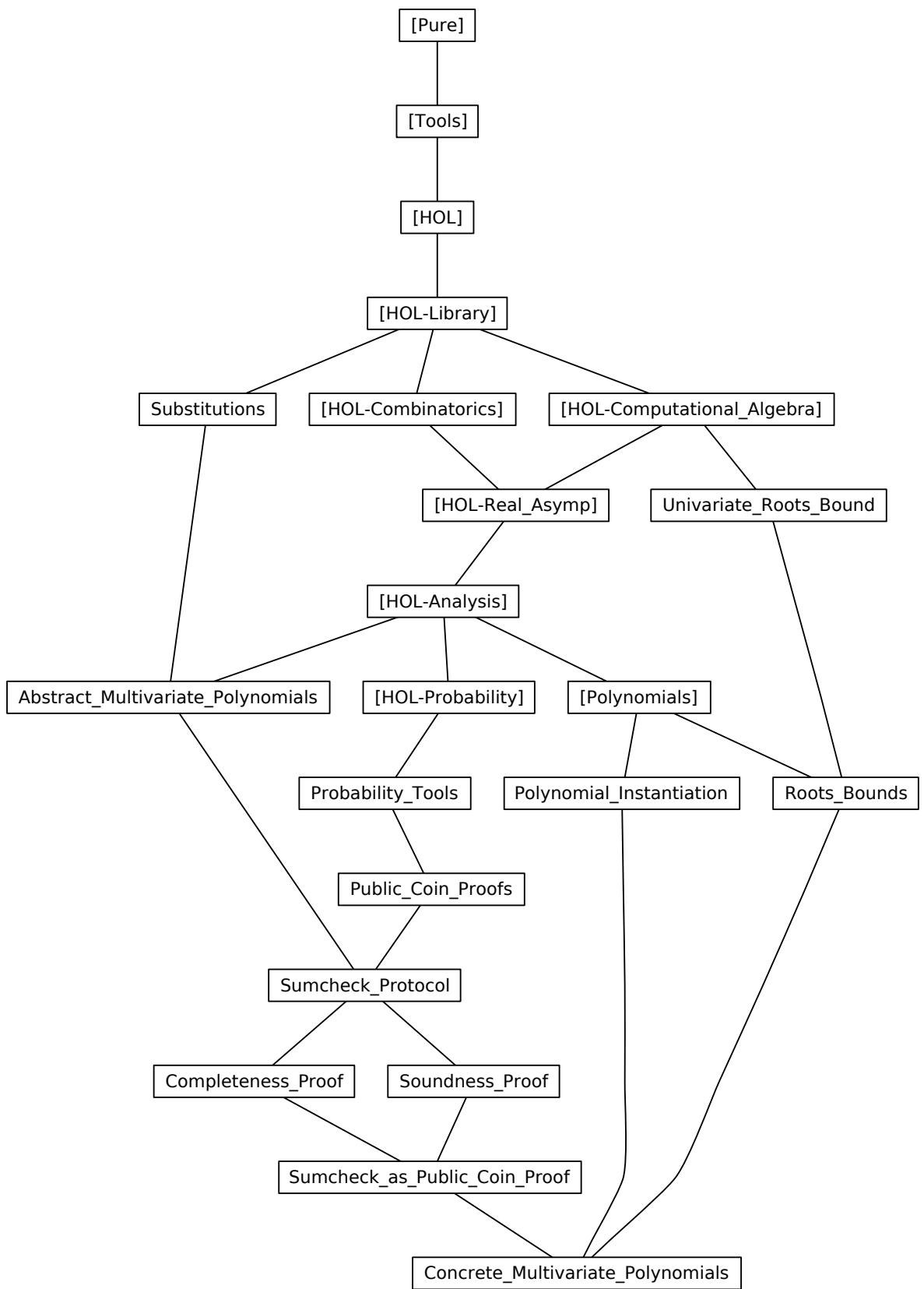
First, we give a general formalization of public-coin interactive proofs. We then define a *generalized sumcheck protocol* for which we axiomatize the underlying mathematical structure and we establish its soundness and completeness. Finally, we prove that these axioms hold for multivariate polynomials, 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

<b>1</b>	<b>Auxiliary Lemmas Related to Probability Theory</b>	<b>3</b>
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
<b>2</b>	<b>Generic Public-coin Interactive Proofs</b>	<b>5</b>
2.1	Generic definition . . . . .	5
2.2	Generic soundness and completeness . . . . .	5
<b>3</b>	<b>Substitutions</b>	<b>7</b>
<b>4</b>	<b>Abstract Multivariate Polynomials</b>	<b>9</b>
4.1	Arity: definition and some lemmas . . . . .	9
4.2	Lemmas about evaluation, degree, and variables of finite sums . . . . .	10
4.3	Lemmas combining eval, sum, and inst . . . . .	10
4.4	Merging sums over substitutions . . . . .	10

<b>5</b>	<b>Sumcheck Protocol</b>	<b>11</b>
5.1	The sumcheck problem . . . . .	11
5.2	The sumcheck protocol . . . . .	11
5.3	The sumcheck protocol as a public-coin proof instance . . . . .	12
<b>6</b>	<b>Completeness Proof for the Sumcheck Protocol</b>	<b>13</b>
<b>7</b>	<b>Soundness Proof for the Sumcheck Protocol</b>	<b>14</b>
<b>8</b>	<b>Sumcheck Protocol as Public-coin Proof</b>	<b>15</b>
8.1	Property-related definitions . . . . .	15
8.2	Public coin proof locale interpretation . . . . .	15
<b>9</b>	<b>Instantiation for Multivariate Polynomials</b>	<b>16</b>
9.1	Instantiation of monomials . . . . .	16
9.2	Instantiation of polynomials . . . . .	16
9.3	Full instantiation corresponds to evaluation . . . . .	17
<b>10</b>	<b>Roots Bound for Univariate Polynomials</b>	<b>18</b>
10.1	Basic lemmas . . . . .	18
10.2	Univariate roots bound . . . . .	18
<b>11</b>	<b>Roots Bound for Multivariate Polynomials of Arity at Most One</b>	<b>19</b>
11.1	Lemmas connecting univariate and multivariate polynomials . . . . .	19
11.1.1	Basic lemmas . . . . .	19
11.1.2	Total degree corresponds to degree for polynomials of arity at most one . . . . .	19
11.2	Roots bound for univariate polynomials of type <i>'a mpoly</i> . . . . .	20
<b>12</b>	<b>Multivariate Polynomials: Instance</b>	<b>21</b>
12.1	Auxiliary lemmas . . . . .	21
12.2	Proving the assumptions of the locale . . . . .	21
12.2.1	Variables . . . . .	21
12.2.2	Degree . . . . .	21
12.2.3	Evaluation . . . . .	22
12.2.4	Roots assumption . . . . .	22
12.3	Locale interpretation . . . . .	23



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* ::  $\langle 'a \text{ set} \Rightarrow \text{nat} \Rightarrow 'a \text{ list set} \rangle$  **where**  
   $\langle \text{tuples } S \ n = \{xs. \text{set } xs \subseteq S \wedge \text{length } xs = n\} \rangle$

**lemma** *tuplesI*:  $\langle \llbracket \text{set } xs \subseteq S; \text{length } xs = n \rrbracket \Longrightarrow xs \in \text{tuples } S \ n \rangle$   
   $\langle \text{proof} \rangle$

**lemma** *tuplesE* [*elim*]:  $\langle \llbracket xs \in \text{tuples } S \ n; \llbracket \text{set } xs \subseteq S; \text{length } xs = n \rrbracket \Longrightarrow P \rrbracket \Longrightarrow P \rangle$   
   $\langle \text{proof} \rangle$

**lemma** *tuples-Zero*:  $\langle \text{tuples } S \ 0 = \{\{\}\} \rangle$   
   $\langle \text{proof} \rangle$

**lemma** *tuples-Suc*:  $\langle \text{tuples } S \ (\text{Suc } n) = (\lambda(x, xs). x \# xs) \ ` (S \times \text{tuples } S \ n) \rangle$   
   $\langle \text{proof} \rangle$

**lemma** *tuples-non-empty* [*simp*]:  $\langle S \neq \{\} \Longrightarrow \text{tuples } S \ n \neq \{\} \rangle$   
   $\langle \text{proof} \rangle$

**lemma** *tuples-finite* [*simp*]:  $\langle \llbracket \text{finite } (S::'a \text{ set}); S \neq \{\} \rrbracket \Longrightarrow \text{finite } (\text{tuples } S \ n :: 'a \text{ list set}) \rangle$   
   $\langle \text{proof} \rangle$

## 1.2 Congruence and monotonicity

**lemma** *prob-cong*: — adapted from Joshua  
  **assumes**  $\langle \bigwedge x. x \in \text{set-pmf } M \Longrightarrow x \in A \longleftrightarrow x \in B \rangle$   
  **shows**  $\langle \text{measure-pmf.prob } M \ A = \text{measure-pmf.prob } M \ B \rangle$   
   $\langle \text{proof} \rangle$

**lemma** *prob-mono*:  
  **assumes**  $\langle \bigwedge x. x \in \text{set-pmf } M \Longrightarrow x \in A \Longrightarrow x \in B \rangle$   
  **shows**  $\langle \text{measure-pmf.prob } M \ A \leq \text{measure-pmf.prob } M \ B \rangle$   
   $\langle \text{proof} \rangle$

## 1.3 Some simple derived lemmas

**lemma** *prob-empty*:  
  **assumes**  $\langle A = \{\} \rangle$   
  **shows**  $\langle \text{measure-pmf.prob } M \ A = 0 \rangle$   
   $\langle \text{proof} \rangle$

**lemma** *prob-pmf-of-set-geq-1*:  
  **assumes** *finite* *S* **and**  $S \neq \{\}$   
  **shows**  $\text{measure-pmf.prob } (\text{pmf-of-set } S) \ A \geq 1 \longleftrightarrow S \subseteq A \langle \text{proof} \rangle$

## 1.4 Intersection and union lemmas

**lemma** *prob-disjoint-union*:

**assumes**  $\langle A \cap B = \{\} \rangle$

**shows**  $\langle \text{measure-pmf.prob } M (A \cup B) = \text{measure-pmf.prob } M A + \text{measure-pmf.prob } M B \rangle$

$\langle \text{proof} \rangle$

**lemma** *prob-finite-Union*:

**assumes**  $\langle \text{disjoint-family-on } A I \rangle \langle \text{finite } I \rangle$

**shows**  $\langle \text{measure-pmf.prob } M (\bigcup_{i \in I} A i) = (\sum_{i \in I} \text{measure-pmf.prob } M (A i)) \rangle$

$\langle \text{proof} \rangle$

**lemma** *prob-disjoint-cases*:

**assumes**  $\langle B \cup C = A \rangle \langle B \cap C = \{\} \rangle$

**shows**  $\langle \text{measure-pmf.prob } M A = \text{measure-pmf.prob } M B + \text{measure-pmf.prob } M C \rangle$

$\langle \text{proof} \rangle$

**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**  $\langle \text{measure-pmf.prob } M A = (\sum_{i \in I} \text{measure-pmf.prob } M (B i)) \rangle$

$\langle \text{proof} \rangle$

## 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*  $B$   $A \neq \{\}$   $B \neq \{\}$

$\langle \text{proof} \rangle$

**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)**

$\langle \text{proof} \rangle$

**lemma** *prob-tuples-fixed-hd*:

$\langle \text{measure-pmf.prob } (\text{pmf-of-set } (\text{tuples } UNIV (\text{Suc } n))) \{ rs::'a \text{ list. } P rs \}$

$= (\sum_{a \in UNIV} \text{measure-pmf.prob } (\text{pmf-of-set } (\text{tuples } UNIV n)) \{ rs. P (a \# rs) \}) / \text{real}(\text{CARD}('a::\text{finite})) \rangle$

**(is ?lhs = ?rhs)**

$\langle \text{proof} \rangle$

**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.prov*  
 (*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.prov*  
 (*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-provs* =  
*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.prov*  
 (*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-provs*:

*public-coin-proof-security ver0 ver1 S honest-pr*  $\lambda$ . *0 sound-err compl-assm sound-assm*  
 ⟨*proof*⟩

**end**

**end**

### 3 Substitutions

**theory** *Substitutions*

**imports**

*Main*

*HOL-Library.FuncSet*

**begin**

**type-synonym**  $( 'v, 'a ) \text{ subst} = 'v \rightarrow 'a$

**definition**  $\text{substs} :: 'v \text{ set} \Rightarrow 'a \text{ set} \Rightarrow ( 'v, 'a ) \text{ subst set}$  **where**

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

Small lemmas about the set of substitutions

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

*<proof>*

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

*<proof>*

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

*<proof>*

**lemma** *substs-nonempty*:

**assumes**  $H \neq \{ \}$

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

*<proof>*

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

*<proof>*

**lemma** *subst-add*:

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

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

*<proof>*

**lemma** *subst-im*:

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

**shows** *the*  $(\varrho x) \in H$

*<proof>*

**lemma** *subst-restr*:

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

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

*<proof>*

Bijection between sets of substitutions

**lemma** *restrict-map-dom*:  $\sigma \upharpoonright ( \text{dom } \sigma = \sigma$

*<proof>*

**lemma** *bij-betw-set-substs*:

**assumes**  $x \in V$



**defines**  $f \equiv \lambda(a, \sigma::'v \rightarrow 'a). \sigma(x \mapsto a)$   
**and**  $g \equiv \lambda\vartheta::'v \rightarrow 'a. (the (\vartheta x), \vartheta | '(dom \vartheta - \{x\}))$   
**shows** *bij-betw*  $f$   
 $(H \times substs (V - \{x\}) H)$   
 $(substs V H)$

*<proof>*

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

$\langle \text{proof} \rangle$

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

$\langle \text{proof} \rangle$

**lemma** *arity-inst*:

**assumes**  $\langle \text{dom } \sigma \subseteq \text{vars } p \rangle$

**shows**  $\langle \text{arity} (\text{inst } p \sigma) \leq \text{arity } p - \text{card} (\text{dom } \sigma) \rangle$

$\langle \text{proof} \rangle$

## 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**  $\langle \text{eval } (\sum_{i \in I}. pp\ i)\ \sigma = (\sum_{i \in I}. \text{eval } (pp\ i)\ \sigma) \rangle$   
 $\langle \text{proof} \rangle$

**lemma** *vars-sum*:

**assumes**  $\langle \text{finite } I \rangle$   
**shows**  $\langle \text{vars } (\sum_{i \in I}. pp\ i) \subseteq (\bigcup_{i \in I}. \text{vars } (pp\ i)) \rangle$   
 $\langle \text{proof} \rangle$

**lemma** *deg-sum*:

**assumes**  $\langle \text{finite } I \rangle$  **and**  $I \neq \{\}$   
**shows**  $\langle \text{deg } (\sum_{i \in I}. pp\ i) \leq \text{Max } \{\text{deg } (pp\ i) \mid i. i \in I\} \rangle$   
 $\langle \text{proof} \rangle$

## 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**  $\langle \text{eval } (\sum_{\sigma \in \text{substs } V\ H}. \text{inst } p\ \sigma)\ \varrho = (\sum_{\sigma \in \text{substs } V\ H}. \text{eval } p\ (\varrho ++ \sigma)) \rangle$   
 $\langle \text{proof} \rangle$

**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**  $\langle \text{eval } (\sum_{\sigma \in \text{substs } V\ H}. \text{inst } p\ \sigma)\ [x \mapsto r]$   
 $\quad = (\sum_{\sigma \in \text{substs } V\ H}. \text{eval } (\text{inst } p\ [x \mapsto r])\ \sigma) \rangle$   
 $\langle \text{proof} \rangle$

## 4.4 Merging sums over substitutions

**lemma** *sum-merge*:

**assumes**  $\langle x \notin V \rangle$   
**shows**  $\langle (\sum_{h \in H}. (\sum_{\sigma \in \text{substs } V\ H}. \text{eval } p\ ([x \mapsto h] ++ \sigma)))$   
 $\quad = (\sum_{\sigma \in \text{substs } (\text{insert } x\ V)\ H}. \text{eval } p\ \sigma) \rangle$   
 $\langle \text{proof} \rangle$

**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⟩

```

**lemma** *honest-prover-deg*:  
**assumes**  $H \neq \{\}$  *finite*  $V$   
**shows**  $\text{deg} (\sum_{\sigma \in \text{substs } V} H. \text{inst } p \sigma) \leq \text{deg } p$   
 $\langle \text{proof} \rangle$

### 5.3 The sumcheck protocol as a public-coin proof instance

Define verifier functions

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

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

**sublocale** *sc*: *public-coin-proof* *sc-ver0* *sc-ver1*  $\langle \text{proof} \rangle$

Equivalence of *sumcheck* with public-coin proofs instance

**lemma** *prove-sc-eq-sumcheck*:  
 $\langle \text{sc.prove } () \text{ pr ps } (H, p, v) r \text{ rm} = \text{sumcheck } \text{pr ps } (H, p, v) r \text{ rm} \rangle$   
 $\langle \text{proof} \rangle$

**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 rm
     $\langle \text{proof} \rangle$ 

corollary completeness:
  assumes
     $\langle (H, p, v) \in \text{Sumcheck} \rangle$ 
     $\langle \text{vars } p = \text{set } (\text{map } \text{fst } \text{rm}) \rangle$ 
     $\langle \text{distinct } (\text{map } \text{fst } \text{rm}) \rangle$ 
     $\langle H \neq \{\} \rangle$ 
  shows
    sumcheck honest-prover u (H, p, v) r rm
     $\langle \text{proof} \rangle$ 

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

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

$\langle \text{proof} \rangle$

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**

$\text{measure-pmf.prob}$

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

$\{rs.$

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

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

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

$\langle \text{proof} \rangle$

**corollary** *soundness*:

**assumes**

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

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

*distinct vs* **and**

$H \neq \{\}$

**shows**

$\text{measure-pmf.prob}$

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

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

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

$\langle \text{proof} \rangle$

**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*

$\langle$ *proof* $\rangle$

**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 ::  $\langle (nat \Rightarrow_0 nat) \Rightarrow (nat, 'a) \text{ subst} \Rightarrow 'a::comm-semiring-1 \rangle$ 
is  $\langle \lambda m \sigma. (\prod v \mid v \in dom \sigma \wedge the (\sigma v) \wedge m v \neq 1. the (\sigma v) \wedge m v) \rangle$  <proof>

```

#### lift-definition

```

inst-monom-resid ::  $\langle (nat \Rightarrow_0 nat) \Rightarrow (nat, 'a) \text{ subst} \Rightarrow (nat \Rightarrow_0 nat) \rangle$ 
is  $\langle (\lambda m \sigma v. m v \text{ when } v \notin dom \sigma) \rangle$ 
<proof>

```

```

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

```

**lemma** *lookup-inst-monom-resid:*

```

shows  $\langle lookup (inst-monom-resid m \sigma) v = (if v \in dom \sigma \text{ then } 0 \text{ else } lookup m v) \rangle$ 
<proof>

```

### 9.2 Instantiation of polynomials

#### definition

```

inst-fun ::  $\langle ((nat \Rightarrow_0 nat) \Rightarrow 'a) \Rightarrow (nat, 'a) \text{ subst} \Rightarrow (nat \Rightarrow_0 nat) \Rightarrow 'a::comm-semiring-1 \rangle$  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>

```

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

```

assumes  $\langle finite \{m. p m \neq 0\} \rangle$ 
shows  $\langle finite \{a. (\sum m' \mid inst-monom-resid m' \sigma = a \wedge p m' \neq 0 \wedge inst-monom-coeff m' \sigma \neq 0. p m' * inst-monom-coeff m' \sigma * (\prod aa. the (q aa) \wedge lookup (inst-monom-resid m' \sigma) aa)) \neq 0\} \rangle$ 
<proof>

```

#### lift-definition

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

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

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

### 9.3 Full instantiation corresponds to evaluation

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

**lemma** *inst-full-eq-insertion*:  
**fixes**  $p :: \langle 'a::\text{semidom} \rangle \text{ mpoly}$  **and**  $\sigma :: \langle nat \Rightarrow 'a \rangle$   
**shows**  $\langle \text{inst } p (Some \circ \sigma) = \text{Const} (\text{insertion } \sigma \ p) \rangle$   
 $\langle \text{proof} \rangle$

**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: ⟨finite {n. poly.coeff p n ≠ 0}⟩
  ⟨proof⟩
```

Univariate degree in terms of *Max*

```
lemma poly-degree-eq-Max-non-zero-coeffs:
  degree p = Max (insert 0 {n. poly.coeff p n ≠ 0})
  ⟨proof⟩
```

### 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::{comm-ring-1,ring-no-zero-divisors} poly
  and q :: 'a::{comm-ring-1,ring-no-zero-divisors} poly
  assumes p ≠ 0 and q ≠ 0
  shows card {x. poly p x * poly q x = 0} ≤ card {x. poly p x = 0} + card {x. poly q x = 0}
  ⟨proof⟩
```

A univariate polynomial *p* has at most *deg p* roots.

```
lemma univariate-roots-bound:
  fixes p :: 'a::idom poly
  assumes ⟨p ≠ 0⟩
  shows ⟨card {x. poly p x = 0} ≤ degree p⟩
  ⟨proof⟩
```

```
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  $\langle vars\ p \subseteq \{v\} \rangle$ 
  shows  $mpoly\text{-to-poly}\ v\ p = 0 \longleftrightarrow p = 0$ 
   $\langle proof \rangle$ 
```

```
lemma keys-monom-subset-vars:
  fixes  $p::'a::zero$  mpoly
  assumes  $\langle m \in keys\ (mapping\text{-of}\ p) \rangle$ 
  shows  $keys\ m \subseteq vars\ p$ 
   $\langle proof \rangle$ 
```

```
lemma sum-lookup-keys-eq-lookup:
  fixes  $p::'a::zero$  mpoly
  assumes  $\langle m \in keys\ (mapping\text{-of}\ p) \rangle$  and  $\langle vars\ p \subseteq \{v\} \rangle$ 
  shows  $sum\ (lookup\ m)\ (keys\ m) = lookup\ m\ v$ 
   $\langle proof \rangle$ 
```

### 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  $\langle vars\ p \subseteq \{v\} \rangle$ 
  shows  $degree\ (mpoly\text{-to-poly}\ v\ p) = MPoly\text{-Type}.degree\ p\ v$ 
   $\langle proof \rangle$ 
```

```
lemma mpoly-degree-eq-total-degree:
  fixes  $p::'a::zero$  mpoly
  assumes  $\langle vars\ p \subseteq \{v\} \rangle$ 
  shows  $MPoly\text{-Type}.degree\ p\ v = total\text{-degree}\ p$ 
   $\langle proof \rangle$ 
```

```
corollary poly-degree-eq-total-degree:
  fixes  $p::'a::comm-monoid-add$  mpoly
  assumes  $\langle vars\ p \subseteq \{v\} \rangle$ 
  shows  $degree\ (mpoly\text{-to-poly}\ v\ p) = total\text{-degree}\ p$ 
   $\langle proof \rangle$ 
```

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

**lemma** *univariate-mpoly-roots-bound*:

**fixes**  $p::'a::\text{idom } \text{mpoly}$

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

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

$\langle \text{proof} \rangle$

**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  $\text{card } \{x. P \wedge Q\ x\} \leq d$ 
   $\langle \text{proof} \rangle$ 

```

```

lemma sum-point-neq-zero [simp]:  $(\sum x' \mid x' = x \wedge f\ x' \neq 0. f\ x') = f\ x$ 
   $\langle \text{proof} \rangle$ 

```

### 12.2 Proving the assumptions of the locale

#### 12.2.1 Variables

```

lemma vars-zero:  $\langle \text{vars } 0 = \{\} \rangle$ 
   $\langle \text{proof} \rangle$ 

```

```

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

```

```

lemma vars-minus:  $\langle \text{vars } p = \text{vars } (-p) \rangle$ 
   $\langle \text{proof} \rangle$ 

```

```

lemma vars-subtr:
  fixes  $p\ q :: \langle 'a::\text{comm-ring } \text{mpoly} \rangle$ 
  shows  $\langle \text{vars } (p - q) \subseteq \text{vars } p \cup \text{vars } q \rangle$ 
   $\langle \text{proof} \rangle$ 

```

#### 12.2.2 Degree

```

abbreviation deg ::  $\langle ('a::\text{zero})\ \text{mpoly} \Rightarrow \text{nat} \rangle$  where
   $\langle \text{deg } p \equiv \text{total-degree } p \rangle$ 

```

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

```

lemma deg-zero:  $\langle \text{deg } 0 = 0 \rangle$   $\langle \text{proof} \rangle$ 

```

```

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

```

```

lemma deg-inst:  $\langle \text{deg } (\text{inst } p\ \sigma) \leq \text{deg } p \rangle$ 

```

*<proof>*

**lemma** *deg-minus*:  $\langle \text{deg } p = \text{deg } (-p) \rangle$

*<proof>*

**lemma** *deg-subtr*:

**fixes**  $p\ q :: \langle 'a::\text{comm-ring } \text{mpoly} \rangle$

**shows**  $\langle \text{deg } (p - q) \leq \max (\text{deg } p) (\text{deg } q) \rangle$

*<proof>*

### 12.2.3 Evaluation

**abbreviation** *eval* ::  $\langle 'a\ \text{mpoly} \Rightarrow (\text{nat}, 'a)\ \text{subst} \Rightarrow ('a::\text{comm-semiring-1}) \rangle$  **where**

$\langle \text{eval } p\ \sigma \equiv \text{insertion } (\text{the } o\ \sigma)\ p \rangle$

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

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

*<proof>*

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

*<proof>*

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

*<proof>*

**lemma** *eval-minus*:

**fixes**  $p :: \langle 'a::\text{comm-ring-1 } \text{mpoly} \rangle$

**shows**  $\langle \text{eval } (-p)\ \sigma = - \text{eval } p\ \sigma \rangle$

*<proof>*

**lemma** *eval-subtr*:

**fixes**  $p\ q :: \langle 'a::\text{comm-ring-1 } \text{mpoly} \rangle$

**assumes**  $\langle \text{vars } p \subseteq \text{dom } \sigma \rangle \langle \text{vars } q \subseteq \text{dom } \sigma \rangle$

**shows**  $\langle \text{eval } (p - q)\ \sigma = \text{eval } p\ \sigma - \text{eval } q\ \sigma \rangle$

*<proof>*

### 12.2.4 Roots assumption

**lemma** *univariate-eval-as-insertion*:

**fixes**  $p :: \langle 'a::\text{comm-ring-1 } \text{mpoly} \rangle$  **and**  $r$

**assumes**  $\text{vars } p \subseteq \{x\}$

**shows**  $\text{eval } p\ [x \mapsto r] = \text{insertion } (\lambda x. r)\ p$

*<proof>*

**lemma** *univariate-mpoly-roots-bound-eval*: — variant using *eval*

**fixes**  $p :: 'a::\text{idom } \text{mpoly}$

**assumes**  $\langle \text{vars } p \subseteq \{x\} \rangle \langle p \neq 0 \rangle$

**shows**  $\langle \text{card } \{r. \text{eval } p\ [x \mapsto r] = 0\} \leq \text{deg } p \rangle$

*<proof>*

**lemma** *mpoly-roots*:

**fixes**  $p\ q :: \langle 'a::\text{idom}\ \text{mpoly} \rangle$  **and**  $d\ x$   
**shows**  $\langle \text{card}\ \{r.\ \text{deg}\ p \leq d \wedge \text{vars}\ p \subseteq \{x\} \wedge \text{deg}\ q \leq d \wedge \text{vars}\ q \subseteq \{x\} \wedge$   
 $p \neq q \wedge \text{eval}\ p\ [x \mapsto r] = \text{eval}\ q\ [x \mapsto r]\} \leq d \rangle$   
 $\langle \text{proof} \rangle$

### 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  $\Rightarrow$  nat eval inst*  
 $\langle \text{proof} \rangle$

Here are the main results, specialized for type  $'a\ \text{mpoly}$ . The completeness theorem for this type is

$\llbracket (?H, ?p, ?v) \in \text{mpoly.Sumcheck}; \text{vars}\ ?p = \text{set}\ (\text{map}\ \text{fst}\ ?rm);$   
 $\text{distinct}\ (\text{map}\ \text{fst}\ ?rm); ?H \neq \{\}\rrbracket$   
 $\Longrightarrow \text{mpoly.sumcheck}\ \text{mpoly.honest-prover}\ ?u\ (?H, ?p, ?v)\ ?r\ ?rm$

and the soundness theorem reads

$\llbracket (?H, ?p, ?v) \notin \text{mpoly.Sumcheck}; \text{vars}\ ?p = \text{set}\ ?vs; \text{distinct}\ ?vs; ?H \neq \{\}\rrbracket$   
 $\Longrightarrow \text{measure-pmf.prob}\ (\text{pmf-of-set}\ (\text{tuples}\ \text{UNIV}\ (\text{mpoly.arity}\ ?p)))$   
 $\{\text{rs}.\ \text{mpoly.sumcheck}\ ?pr\ ?s\ (?H, ?p, ?v)\ ?r\ (\text{zip}\ ?vs\ \text{rs})\}$   
 $\leq \text{real}\ (\text{mpoly.arity}\ ?p) * \text{real}\ (\text{deg}\ ?p) / \text{real}\ \text{CARD}('a)$

.

**end**