The Sumcheck Protocol

Azucena Garvia, Christoph Sprenger and Jonathan Bootle

March 17, 2025

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

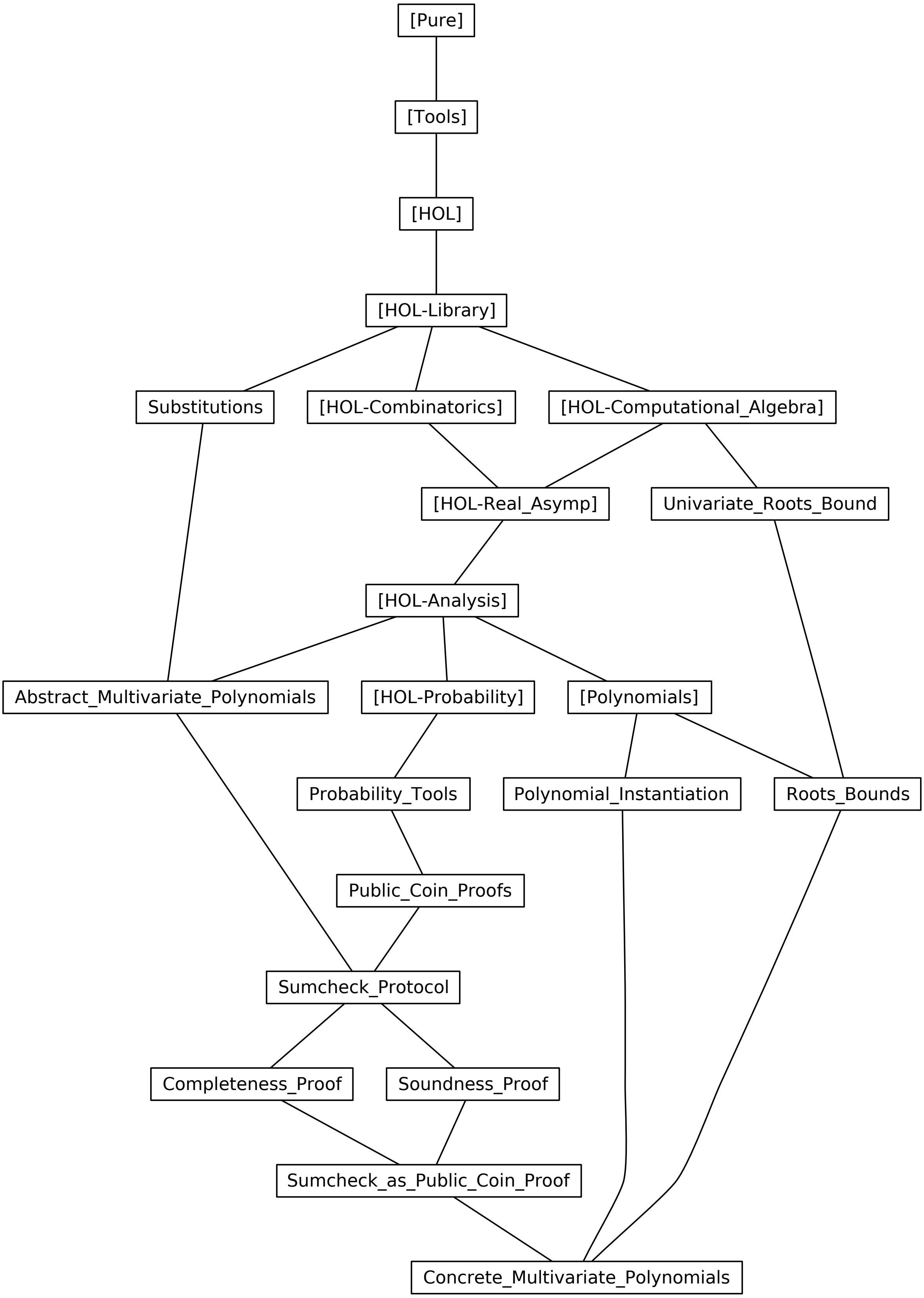

Figure 1: Theory dependencies

1 Auxiliary Lemmas Related to Probability Theory

theory *Probability-Tools*
 imports *HOL−Probability.Probability*
begin

1.1 Tuples

definition *tuples* :: ‹*'a set* ⇒ *nat* ⇒ *'a list set*› **where**
 ‹*tuples S n* = {*xs*. *set xs* ⊆ *S* ∧ *length xs* = *n*}›

lemma *tuplesI*: ‹⟦ *set xs* ⊆ *S*; *length xs* = *n* ⟧ ⟹ *xs* ∈ *tuples S n*›
 ⟨*proof*⟩

lemma *tuplesE* [*elim*]: ‹⟦ *xs* ∈ *tuples S n*; ⟦ *set xs* ⊆ *S*; *length xs* = *n* ⟧ ⟹ *P* ⟧ ⟹ *P*›
 ⟨*proof*⟩

lemma *tuples-Zero*: ‹*tuples S 0* = {[]}›
 ⟨*proof*⟩

lemma *tuples-Suc*: ‹*tuples S* (*Suc n*) = (λ(*x*, *xs*). *x* # *xs*) ‘ (*S* × *tuples S n*)›
 ⟨*proof*⟩

lemma *tuples-non-empty* [*simp*]: ‹*S* ≠ {} ⟹ *tuples S n* ≠ {}›
 ⟨*proof*⟩

lemma *tuples-finite* [*simp*]: ‹⟦ *finite* (*S*::*'a set*); *S* ≠ {} ⟧ ⟹ *finite* (*tuples S n* :: *'a list set*)›
 ⟨*proof*⟩

1.2 Congruence and monotonicity

lemma *prob-cong*: — adapted from Joshua
 assumes ‹⋀*x*. *x* ∈ *set-pmf M* ⟹ *x* ∈ *A* ⟷ *x* ∈ *B*›
 shows ‹*measure-pmf.prob M A* = *measure-pmf.prob M B*›
 ⟨*proof*⟩

lemma *prob-mono*:
 assumes ‹⋀*x*. *x* ∈ *set-pmf M* ⟹ *x* ∈ *A* ⟹ *x* ∈ *B*›
 shows ‹*measure-pmf.prob M A* ≤ *measure-pmf.prob M B*›
 ⟨*proof*⟩

1.3 Some simple derived lemmas

lemma *prob-empty*:
 assumes ‹*A* = {}›
 shows ‹*measure-pmf.prob M A* = *0*›
 ⟨*proof*⟩

lemma *prob-pmf-of-set-geq-1*:
 assumes *finite S* **and** *S* ≠ {}
 shows *measure-pmf.prob* (*pmf-of-set S*) *A* ≥ *1* ⟷ *S* ⊆ *A* ⟨*proof*⟩

1.4 Intersection and union lemmas

lemma *prob-disjoint-union*:
  assumes ‹$A \cap B = \{\}$›
  shows ‹$measure\text{-}pmf.prob\ M\ (A \cup B) = measure\text{-}pmf.prob\ M\ A + measure\text{-}pmf.prob\ M\ B$›
  ⟨*proof*⟩

lemma *prob-finite-Union*:
  assumes ‹*disjoint-family-on A I*› ‹*finite I*›
  shows ‹$measure\text{-}pmf.prob\ M\ (\bigcup i{\in}I.\ A\ i) = (\sum i{\in}I.\ measure\text{-}pmf.prob\ M\ (A\ i))$›
  ⟨*proof*⟩

lemma *prob-disjoint-cases*:
  assumes ‹$B \cup C = A$› ‹$B \cap C = \{\}$›
  shows ‹$measure\text{-}pmf.prob\ M\ A = measure\text{-}pmf.prob\ M\ B + measure\text{-}pmf.prob\ M\ C$›
⟨*proof*⟩

lemma *prob-finite-disjoint-cases*:
  assumes ‹$(\bigcup i{\in}I.\ B\ i) = A$› ‹*disjoint-family-on B I*› ‹*finite I*›
  shows ‹$measure\text{-}pmf.prob\ M\ A = (\sum i{\in}I.\ measure\text{-}pmf.prob\ M\ (B\ i))$›
⟨*proof*⟩

1.5 Independent probabilities for head and tail of a tuple

lemma *pmf-of-set-Times*: — by Andreas Lochbihler
  $pmf\text{-}of\text{-}set\ (A \times B) = pair\text{-}pmf\ (pmf\text{-}of\text{-}set\ A)\ (pmf\text{-}of\text{-}set\ B)$
  if *finite A finite B* $A \neq \{\}$ $B \neq \{\}$
  ⟨*proof*⟩

lemma *prob-tuples-hd-tl-indep*:
  assumes ‹$S \neq \{\}$›
  shows
    ‹$measure\text{-}pmf.prob\ (pmf\text{-}of\text{-}set\ (tuples\ S\ (Suc\ n)))\ \{(r{::}'a{::}finite)\ \#\ rs \mid r\ rs.\ P\ r \wedge Q\ rs\}$
   $= measure\text{-}pmf.prob\ (pmf\text{-}of\text{-}set\ (S{::}'a\ set))\ \{r.\ P\ r\} *$
    $measure\text{-}pmf.prob\ (pmf\text{-}of\text{-}set\ (tuples\ S\ n))\ \{rs.\ Q\ rs\}$›
   (**is** *?lhs = ?rhs*)
⟨*proof*⟩

lemma *prob-tuples-fixed-hd*:
  ‹$measure\text{-}pmf.prob\ (pmf\text{-}of\text{-}set\ (tuples\ UNIV\ (Suc\ n)))\ \{rs{::}'a\ list.\ P\ rs\}$
  $= (\sum a \in UNIV.\ measure\text{-}pmf.prob\ (pmf\text{-}of\text{-}set\ (tuples\ UNIV\ n))\ \{rs.\ P\ (a\ \#\ rs)\})\ /\ real(CARD('a{::}finite))$›
  (**is** *?lhs = ?rhs*)
⟨*proof*⟩

end

2 Generic Public-coin Interactive Proofs

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

2.1 Generic definition

type-synonym $('i, 'r, 'a, 'resp, 'ps)\ prv = 'i \Rightarrow 'a \Rightarrow 'a\ list \Rightarrow 'r \Rightarrow 'ps \Rightarrow 'resp \times 'ps$

locale *public-coin-proof* =
  fixes $ver0 :: 'i \Rightarrow 'vs \Rightarrow bool$
    and $ver1 :: 'i \Rightarrow 'resp \Rightarrow 'r \Rightarrow 'a \Rightarrow 'a\ list \Rightarrow 'vs \Rightarrow bool \times 'i \times 'vs$
begin

fun $prove :: 'vs \Rightarrow ('i, 'r, 'a, 'resp, 'ps)\ prv \Rightarrow 'ps \Rightarrow 'i \Rightarrow 'r \Rightarrow ('a \times 'r)\ list \Rightarrow bool$ **where**
  $prove\ vs\ prv\ ps\ I\ r\ [] \longleftrightarrow ver0\ I\ vs$ |
  $prove\ vs\ prv\ ps\ I\ r\ ((x, r')\#rm) \longleftrightarrow$
    $(let\ (resp, ps') = prv\ I\ x\ (map\ fst\ rm)\ r\ ps\ in$
     $let\ (ok, I', vs') = ver1\ I\ resp\ r'\ x\ (map\ fst\ rm)\ vs\ in$
       $ok \wedge prove\ vs'\ prv\ ps'\ I'\ r'\ rm)$

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 $ver0 :: 'i \Rightarrow 'vs \Rightarrow bool$
  and $ver1 :: 'i \Rightarrow 'resp \Rightarrow 'r \Rightarrow 'a \Rightarrow 'a\ list \Rightarrow 'vs \Rightarrow bool \times 'i \times 'vs$ +
  fixes $S :: 'i\ set$ — problem specification
    and $honest\text{-}pr :: ('i, 'r, 'a, 'resp, 'ps)\ prv$
    and $compl\text{-}err :: 'i \Rightarrow real$
    and $sound\text{-}err :: 'i \Rightarrow real$
    and $compl\text{-}assm :: 'vs \Rightarrow 'ps \Rightarrow 'i \Rightarrow 'a\ list \Rightarrow bool$
    and $sound\text{-}assm :: 'vs \Rightarrow 'ps \Rightarrow 'i \Rightarrow 'a\ list \Rightarrow bool$
  assumes
    completeness:
      $[\![\ I \in S;\ compl\text{-}assm\ vs\ ps\ I\ xs\ ]\!] \Longrightarrow$

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

soundness:
⟦ *I* ∉ *S*; *sound-assm vs ps I xs* ⟧ ⟹
measure-pmf.prob
(pmf-of-set (tuples UNIV (length xs)))
{rs. prove vs pr ps I r (zip xs rs)} ≤ *sound-err I*

locale *public-coin-proof-strong-props* =
public-coin-proof ver0 ver1
for *ver0* :: *'i* ⇒ *'vs* ⇒ *bool*
and *ver1* :: *'i* ⇒ *'resp* ⇒ *'r::finite* ⇒ *'a* ⇒ *'a list* ⇒ *'vs* ⇒ *bool* × *'i* × *'vs* +
fixes *S* :: *'i set* — problem specification
and *honest-pr* :: (*'i*, *'r*, *'a*, *'resp*, *'ps*) *prv*
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 (map fst rm)* ⟧ ⟹ *prove vs honest-pr ps I r rm* **and**

soundness:
⟦ *I* ∉ *S*; *sound-assm vs ps I xs* ⟧ ⟹
measure-pmf.prob
(pmf-of-set (tuples UNIV (length xs)))
{rs. prove vs pr ps I r (zip xs rs)} ≤ *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
⟨*proof*⟩

end

end

3 Substitutions

theory *Substitutions*
imports
Main
HOL−Library.FuncSet
begin

type-synonym $('v, 'a)\ subst = 'v \rightharpoonup 'a$

definition $substs :: 'v\ set \Rightarrow 'a\ set \Rightarrow ('v, 'a)\ subst\ set$ **where**
$substs\ V\ H = \{\sigma.\ dom\ \sigma = V \wedge ran\ \sigma \subseteq H\}$

Small lemmas about the set of substitutions

lemma *substE* [*elim*]: $[\![\ \sigma \in substs\ V\ H;\ [\![\ dom\ \sigma = V;\ ran\ \sigma \subseteq H\ ]\!] \Longrightarrow P\ ]\!] \Longrightarrow P$
⟨*proof*⟩

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

lemma *substs-finite*: $[\![\ finite\ V;\ finite\ H\ ]\!] \Longrightarrow finite\ (substs\ V\ H)$
⟨*proof*⟩

lemma *substs-nonempty*:
assumes $H \neq \{\}$
shows $substs\ V\ H \neq \{\}$
⟨*proof*⟩

lemma *subst-dom*: ‹$[\![\ \varrho \in substs\ V\ H;\ x \notin V\ ]\!] \Longrightarrow x \notin dom\ \varrho$›
⟨*proof*⟩

lemma *subst-add*:
assumes $x \in V$ **and** $\varrho \in substs\ (V - \{x\})\ H$ **and** $a \in H$
shows $\varrho(x \mapsto a) \in substs\ V\ H$
⟨*proof*⟩

lemma *subst-im*:
assumes $x \in V$ **and** $\varrho \in substs\ V\ H$
shows $the\ (\varrho\ x) \in H$
⟨*proof*⟩

lemma *subst-restr*:
assumes $x \in V$ **and** $\varrho \in substs\ V\ H$
shows ϱ |‘ $(dom\ \varrho - \{x\}) \in substs\ (V - \{x\})\ H$
⟨*proof*⟩

Bijection between sets of substitutions

lemma *restrict-map-dom*: σ |‘ $dom\ \sigma = \sigma$
⟨*proof*⟩

lemma *bij-betw-set-substs*:
assumes $x \in V$

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

⟨*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* :: ‹$'p$ :: *comm-monoid-add* ⇒ $'v$ *set*›
  and *deg* :: ‹$'p$ ⇒ *nat*›
  and *eval* :: ‹$'p$ ⇒ ($'v$, $'a$::*finite*) *subst* ⇒ $'b$ :: *comm-monoid-add*›
  and *inst* :: ‹$'p$ ⇒ ($'v$, $'a$) *subst* ⇒ $'p$›
assumes
  — vars
  vars-finite: ‹*finite* (*vars p*)› **and**
  vars-zero: ‹*vars 0* = {}› **and**
  vars-add: ‹*vars* (*p* + *q*) ⊆ *vars p* ∪ *vars q*› **and**
  vars-inst: ‹*vars* (*inst p σ*) ⊆ *vars p* − *dom σ*› **and**

  — degree
  deg-zero: ‹*deg 0* = *0*› **and**
  deg-add: ‹*deg* (*p* + *q*) ≤ *max* (*deg p*) (*deg q*)› **and**
  deg-inst: ‹*deg* (*inst p ϱ*) ≤ *deg p*› **and**

  — eval
  eval-zero: ‹*eval 0 σ* = *0*› **and**
  eval-add: ‹*vars p* ∪ *vars q* ⊆ *dom σ* ⟹ *eval* (*p* + *q*) *σ* = *eval p σ* + *eval q σ*› **and**
  eval-inst: ‹*vars p* ⊆ *dom σ* ∪ *dom ϱ* ⟹ *eval* (*inst p σ*) *ϱ* = *eval p* (*ϱ* ++ *σ*)› **and**

  — small number of roots (variant for two polynomials)
  roots: ‹*card* {*r*. *deg p* ≤ *d* ∧ *vars p* ⊆ {*x*} ∧ *deg q* ≤ *d* ∧ *vars q* ⊆ {*x*} ∧
          p ≠ *q* ∧ *eval p* [*x* ↦ *r*] = *eval q* [*x* ↦ *r*]} ≤ *d*›
begin

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

4.1 Arity: definition and some lemmas

definition *arity* :: ‹$'p$ ⇒ *nat*› **where**
  ‹*arity p* = *card* (*vars p*)›

lemma *arity-zero*: ‹*arity 0* = *0*›
  ⟨*proof*⟩

lemma *arity-add*: ‹*arity* (*p* + *q*) ≤ *arity p* + *arity q*›
⟨*proof*⟩

lemma *arity-inst*:
  assumes ‹*dom σ* ⊆ *vars p*›
  shows ‹*arity* (*inst p σ*) ≤ *arity p* − *card* (*dom σ*)›
⟨*proof*⟩

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

lemma *eval-sum*:
  assumes ‹*finite I*› ‹$\bigwedge i.\ i \in I \Longrightarrow vars\ (pp\ i) \subseteq dom\ \sigma$›
  shows ‹$eval\ (\sum i{\in}I.\ pp\ i)\ \sigma = (\sum i{\in}I.\ eval\ (pp\ i)\ \sigma)$›
⟨*proof*⟩

lemma *vars-sum*:
  assumes ‹*finite I*›
  shows ‹$vars\ (\sum i{\in}I.\ pp\ i) \subseteq (\bigcup i{\in}I.\ vars\ (pp\ i))$›
  ⟨*proof*⟩

lemma *deg-sum*:
  assumes ‹*finite I*› **and** $I \neq \{\}$
  shows ‹$deg\ (\sum i{\in}I.\ pp\ i) \leq Max\ \{deg\ (pp\ i) \mid i.\ i \in I\}$›
  ⟨*proof*⟩

4.3 Lemmas combining eval, sum, and inst

lemma *eval-sum-inst*:
  assumes ‹$vars\ p \subseteq V \cup dom\ \varrho$› ‹*finite V*›
  shows ‹$eval\ (\sum \sigma \in substs\ V\ H.\ inst\ p\ \sigma)\ \varrho = (\sum \sigma \in substs\ V\ H.\ eval\ p\ (\varrho\ {+}{+}\ \sigma))$›
⟨*proof*⟩

lemma *eval-sum-inst-commute*:
  assumes ‹$vars\ p \subseteq insert\ x\ V$› ‹$x \notin V$› ‹*finite V*›
  shows ‹$eval\ (\sum \sigma{\in}substs\ V\ H.\ inst\ p\ \sigma)\ [x \mapsto r]$
    $= (\sum \sigma{\in}substs\ V\ H.\ eval\ (inst\ p\ [x \mapsto r])\ \sigma)$›
⟨*proof*⟩

4.4 Merging sums over substitutions

lemma *sum-merge*:
  assumes ‹$x \notin V$›
  shows $(\sum h{\in}H.\ (\sum \sigma \in substs\ V\ H.\ eval\ p\ ([x \mapsto h]\ {+}{+}\ \sigma)))$
    $= (\sum \sigma{\in}substs\ (insert\ x\ V)\ H.\ eval\ p\ \sigma)$
⟨*proof*⟩

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* $\times$ $'p \times 'b$

definition (**in** *multi-variate-polynomial*)
  Sumcheck :: $('p, 'a, 'b)$ *sc-inst set* **where**
  Sumcheck $= \{(H, p, v) \mid H\ p\ v.\ v = (\sum \sigma \in \mathit{substs}\ (\mathit{vars}\ p)\ H.\ \mathit{eval}\ p\ \sigma)\}$

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 lools like

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

.

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

Sumcheck function

fun *sumcheck* :: $('p, 'a, 'b, 'v, 's)$ *prover* $\Rightarrow 's \Rightarrow ('p, 'a, 'b)$ *sc-inst* $\Rightarrow 'a \Rightarrow ('v \times 'a)$ *list* $\Rightarrow$ *bool*
where
  sumcheck pr s (H, p, v) *r-prev* $[] \longleftrightarrow v = \mathit{eval}\ p\ \mathit{Map.empty}$
| *sumcheck pr s* (H, p, v) *r-prev* $((x, r) \# rm) \longleftrightarrow$
    $(\mathit{let}\ (q, s') = \mathit{pr}\ (H, p, v)\ x\ (\mathit{map}\ \mathit{fst}\ rm)\ \mathit{r\text{-}prev}\ s\ \mathit{in}$
      $\mathit{vars}\ q \subseteq \{x\} \wedge \mathit{deg}\ q \leq \mathit{deg}\ p \wedge$
      $v = (\sum y \in H.\ \mathit{eval}\ q\ [x \mapsto y]) \wedge$
      $\mathit{sumcheck}\ \mathit{pr}\ s'\ (H, \mathit{inst}\ p\ [x \mapsto r], \mathit{eval}\ q\ [x \mapsto r])\ r\ rm)$

Honest prover definition

fun *honest-prover* :: $('p, 'a, 'b, 'v, \mathit{unit})$ *prover* **where**
  honest-prover $(H, p, \text{-})\ x\ xs$ - - $= (\sum \sigma \in \mathit{substs}\ (\mathit{set}\ xs)\ H.\ \mathit{inst}\ p\ \sigma, ())$

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 $\mathit{vars}\ p \subseteq \mathit{insert}\ x\ V$ *finite* V $H \neq \{\}$ *finite* H
  shows $\mathit{vars}\ (\sum \sigma \in \mathit{substs}\ V\ H.\ \mathit{inst}\ p\ \sigma) \subseteq \{x\}$
⟨*proof*⟩

lemma *honest-prover-deg*:
  assumes $H \neq \{\}$ *finite V*
  shows $deg\ (\sum \sigma{\in}substs\ V\ H.\ inst\ p\ \sigma) \leq deg\ p$
⟨*proof*⟩

5.3 The sumcheck protocol as a public-coin proof instance

Define verifier functions

fun *sc-ver0* :: ('*p*, '*a*, '*b*) *sc-inst* ⇒ *unit* ⇒ *bool* **where**
  sc-ver0 (*H*, *p*, *v*) () ⟷ *v* = *eval p Map.empty*

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

sublocale *sc*: *public-coin-proof sc-ver0 sc-ver1* ⟨*proof*⟩

Equivalence of *sumcheck* with public-coin proofs instance

lemma *prove-sc-eq-sumcheck*:
  ‹*sc.prove* () *pr ps* (*H*, *p*, *v*) *r rm* = *sumcheck pr ps* (*H*, *p*, *v*) *r rm*›
⟨*proof*⟩

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
    ‹$v = (\sum \sigma \in substs\ (set\ (map\ fst\ rm))\ H.\ eval\ p\ \sigma)$›
    ‹$vars\ p \subseteq set\ (map\ fst\ rm)$›
    ‹$distinct\ (map\ fst\ rm)$›
    ‹$H \neq \{\}$›
  shows
    sumcheck honest-prover u $(H,\ p,\ v)$ *r-prev rm*
  ⟨*proof*⟩

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

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 deg q2 ≤ deg p and vars q2 ⊆ {x}
  shows measure-pmf.prob (pmf-of-set UNIV)
        {r. deg q1 ≤ deg p ∧ vars q1 ⊆ {x} ∧ q1 ≠ q2 ∧ eval q1 [x ↦ r] = eval q2 [x ↦ r]}
        ≤ real (deg p) / real CARD('a)
⟨proof⟩
```

Soundness proof

```
theorem soundness-inductive:
  assumes
    vars p ⊆ set vs and
    deg p ≤ d and
    distinct vs and
    H ≠ {}
  shows
    measure-pmf.prob
      (pmf-of-set (tuples UNIV (length vs)))
      {rs.
        sumcheck pr s (H, p, v) r (zip vs rs) ∧
        v ≠ (∑σ ∈ substs (set vs) H. eval p σ)}
    ≤ real (length vs) ∗ real d / real (CARD('a))
  ⟨proof⟩


corollary soundness:
  assumes
    (H, p, v) ∉ Sumcheck
    vars p = set vs and
    distinct vs and
    H ≠ {}
  shows
    measure-pmf.prob
      (pmf-of-set (tuples UNIV (arity p)))
      {rs. sumcheck pr s (H, p, v) r (zip vs rs)}
    ≤ real (arity p) ∗ real (deg p) / real (CARD('a))
  ⟨proof⟩


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*⟩

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* and $(\neq)$ *0* 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* $= 'a \rightharpoonup 'b$

lift-definition
  inst-monom-coeff :: ‹$(nat \Rightarrow_0 nat) \Rightarrow (nat, 'a)\ subst \Rightarrow 'a{::}comm\text{-}semiring\text{-}1$›
  is ‹$\lambda m\ \sigma.\ (\prod v \mid v \in dom\ \sigma \wedge the\ (\sigma\ v)\ \widehat{}\ m\ v \neq 1.\ the\ (\sigma\ v)\ \widehat{}\ m\ v)$› ⟨*proof*⟩

lift-definition
  inst-monom-resid :: ‹$(nat \Rightarrow_0 nat) \Rightarrow (nat, 'a)\ subst \Rightarrow (nat \Rightarrow_0 nat)$›
  is ‹$(\lambda m\ \sigma\ v.\ m\ v\ when\ v \notin dom\ \sigma)$›
  ⟨*proof*⟩

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

lemma *lookup-inst-monom-resid*:
  shows ‹$lookup\ (inst\text{-}monom\text{-}resid\ m\ \sigma)\ v = (if\ v \in dom\ \sigma\ then\ 0\ else\ lookup\ m\ v)$›
  ⟨*proof*⟩

9.2 Instantiation of polynomials

definition
  inst-fun :: ‹$((nat \Rightarrow_0 nat) \Rightarrow 'a) \Rightarrow (nat, 'a)\ subst \Rightarrow (nat \Rightarrow_0 nat) \Rightarrow 'a{::}comm\text{-}semiring\text{-}1$› **where**
  ‹$inst\text{-}fun\ p\ \sigma = (\lambda m.\ (\sum m' \mid inst\text{-}monom\text{-}resid\ m'\ \sigma = m \wedge p\ m' * inst\text{-}monom\text{-}coeff\ m'\ \sigma \neq 0.$
    $p\ m' * inst\text{-}monom\text{-}coeff\ m'\ \sigma))$›

lemma *finite-inst-fun-keys*:
  assumes ‹$finite\ \{m.\ p\ m \neq 0\}$›
  shows ‹$finite\ \{m.\ (\sum m' \mid inst\text{-}monom\text{-}resid\ m'\ \sigma = m \wedge p\ m' \neq 0 \wedge inst\text{-}monom\text{-}coeff\ m'\ \sigma \neq 0.$
    $p\ m' * inst\text{-}monom\text{-}coeff\ m'\ \sigma) \neq 0\}$›
⟨*proof*⟩

lemma *finite-inst-fun-keys-ext*:
  assumes ‹$finite\ \{m.\ p\ m \neq 0\}$›
  shows $finite\ \{a.\ (\sum m' \mid inst\text{-}monom\text{-}resid\ m'\ \sigma = a \wedge p\ m' \neq 0 \wedge inst\text{-}monom\text{-}coeff\ m'\ \sigma \neq 0.$
    $p\ m' * inst\text{-}monom\text{-}coeff\ m'\ \sigma * (\prod aa.\ the\ (\varrho\ aa)\ \widehat{}\ lookup\ (inst\text{-}monom\text{-}resid\ m'\ \sigma)\ aa)) \neq 0\}$
⟨*proof*⟩

lift-definition

inst-aux :: ‹((*nat* $\Rightarrow_0$ *nat*) $\Rightarrow_0$ *'a*) ⇒ (*nat*, *'a*) *subst* ⇒ (*nat* $\Rightarrow_0$ *nat*) $\Rightarrow_0$ *'a*::*semidom*›
is *inst-fun*
⟨*proof*⟩

lift-definition *inst* :: ‹*'a mpoly* ⇒ (*nat*, *'a*::*semidom*) *subst* ⇒ *'a mpoly*›
is *inst-aux* ⟨*proof*⟩

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

9.3 Full instantiation corresponds to evaluation

lemma *dom-Some*: ‹*dom* (*Some o f*) = *UNIV*›
⟨*proof*⟩

lemma *inst-full-eq-insertion*:
fixes *p* :: ‹(*'a*::*semidom*) *mpoly*› **and** σ :: ‹*nat* ⇒ *'a*›
shows ‹*inst p* (*Some o* σ) = *Const* (*insertion* σ *p*)›
⟨*proof*⟩

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 ‹*vars p* ⊆ {*v*}›
  shows *mpoly-to-poly v p* = *0* ⟷ *p* = *0*
  ⟨*proof*⟩

lemma *keys-monom-subset-vars*:
  fixes *p::'a::zero mpoly*
  assumes ‹*m* ∈ *keys* (*mapping-of p*)›
  shows *keys m* ⊆ *vars p*
  ⟨*proof*⟩

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*
  ⟨*proof*⟩

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*
  ⟨*proof*⟩

lemma *mpoly-degree-eq-total-degree*:
  fixes *p::'a::zero mpoly*
  assumes ‹*vars p* ⊆ {*v*}›
  shows *MPoly-Type.degree p v* = *total-degree p*
  ⟨*proof*⟩

corollary *poly-degree-eq-total-degree*:
  fixes *p::'a::comm-monoid-add mpoly*
  assumes ‹*vars p* ⊆ {*v*}›
  shows *degree* (*mpoly-to-poly v p*) = *total-degree p*
  ⟨*proof*⟩

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

lemma *univariate-mpoly-roots-bound*:
  fixes *p*::*'a*::*idom mpoly*
  assumes ‹*vars p* ⊆ {*v*}› ‹*p* ≠ *0*›
  shows ‹*card* {*x*. *insertion* (*λv*. *x*) *p* = *0*} ≤ *total-degree p*›
  ⟨*proof*⟩

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 $P \Longrightarrow card\ \{x.\ Q\ x\} \leq d$
shows $card\ \{x.\ P \wedge Q\ x\} \leq d$
⟨*proof*⟩

lemma *sum-point-neq-zero* [*simp*]: $(\sum x' \mid x' = x \wedge f\ x' \neq 0.\ f\ x') = f\ x$
⟨*proof*⟩

12.2 Proving the assumptions of the locale

12.2.1 Variables

lemma *vars-zero*: ‹$vars\ 0 = \{\}$›
⟨*proof*⟩

lemma *vars-inst*: ‹$vars\ (inst\ p\ \sigma) \subseteq vars\ p - dom\ \sigma$›
⟨*proof*⟩

lemma *vars-minus*: ‹$vars\ p = vars\ (-p)$›
⟨*proof*⟩

lemma *vars-subtr*:
fixes $p\ q$ :: ‹$'a$::*comm-ring mpoly*›
shows ‹$vars\ (p - q) \subseteq vars\ p \cup vars\ q$›
⟨*proof*⟩

12.2.2 Degree

abbreviation *deg* :: ‹$('a$::*zero*) *mpoly* ⇒ *nat*› **where**
‹$deg\ p \equiv total\text{-}degree\ p$›

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

lemma *deg-zero*: ‹$deg\ 0 = 0$› ⟨*proof*⟩

lemma *deg-add*: ‹$deg\ (p + q) \leq max\ (deg\ p)\ (deg\ q)$›
⟨*proof*⟩

lemma *deg-inst*: ‹$deg\ (inst\ p\ \sigma) \leq deg\ p$›

⟨*proof*⟩

lemma *deg-minus*: ‹*deg p* = *deg* (−*p*)›
⟨*proof*⟩

lemma *deg-subtr*:
fixes *p q* :: ‹′*a*::*comm-ring mpoly*›
shows ‹*deg* (*p* − *q*) ≤ *max* (*deg p*) (*deg* (*q*))›
⟨*proof*⟩

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*: ‹*eval 0 σ* = *0*›
⟨*proof*⟩

lemma *eval-add*: ‹*vars p* ∪ *vars q* ⊆ *dom σ* ⟹ *eval* (*p* + *q*) *σ* = *eval p σ* + *eval q σ*›
⟨*proof*⟩

lemma *eval-inst*: ‹*eval* (*inst p σ*) *ϱ* = *eval p* (*ϱ* ++ *σ*)›
⟨*proof*⟩

lemma *eval-minus*:
fixes *p* :: ‹′*a*::*comm-ring-1 mpoly*›
shows ‹*eval* (−*p*) *σ* = − *eval p σ*›
⟨*proof*⟩

lemma *eval-subtr*:
fixes *p q* :: ‹′*a*::*comm-ring-1 mpoly*›
assumes ‹*vars p* ⊆ *dom σ*› ‹*vars q* ⊆ *dom σ*›
shows ‹*eval* (*p* − *q*) *σ* = *eval p σ* − *eval q σ*›
⟨*proof*⟩

12.2.4 Roots assumption

lemma *univariate-eval-as-insertion*:
fixes *p*::‹′*a*::*comm-ring-1 mpoly*› **and** *r*
assumes *vars p* ⊆ {*x*}
shows *eval p* [*x* ↦ *r*] = *insertion* (*λx. r*) *p*
⟨*proof*⟩

lemma *univariate-mpoly-roots-bound-eval*: — variant using *eval*
fixes *p*::′*a*::*idom mpoly*
assumes ‹*vars p* ⊆ {*x*}› ‹*p* ≠ *0*›
shows ‹*card* {*r. eval p* [*x* ↦ *r*] = *0*} ≤ *deg p*›
⟨*proof*⟩

lemma *mpoly-roots*:

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

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*
⟨*proof*⟩

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

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

and the soundness theorem reads

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

.

end