# Formalization of the Schwartz-Zippel Lemma

Sunpill Kim and Yong Kiam Tan

March 19, 2025

**Abstract**

This short entry formalizes a version of the Schwartz-Zippel lemma for probabilistic (multivariate) polynomial identity testing. The entry includes a textbook example using the lemma to test for perfect matchings in a bipartite graph. The lemma is attributed to several independent authors, including Schwartz [3], Zippel [4], and DeMillo and Lipton [1]; a historical perspective is given by Lipton [2].

## Contents



## 1 The Schwartz-Zippel Lemma

**theory** *Schwartz-Zippel* **imports**
*Factor-Algebraic-Polynomial.Poly-Connection*
*Polynomials.MPoly-Type*
*HOL−Probability.Product-PMF*
**begin**

This theory formalizes the Schwartz-Zippel lemma for multivariate polynomials (*mpoly*).

**lemma** *schwartz-zippel-uni*:
**fixes** $P$ :: ('*a*::*idom*) *Polynomial.poly*
**fixes** $S$ :: '*a set*
**assumes** *finite* $S$ $S \neq \{\}$
**assumes** *Polynomial.degree* $P \leq d$
**assumes** $P \neq 0$
**shows** *measure-pmf.prob* (*pmf-of-set* $S$) $\{r.$ *poly* $P$ $r = 0\} \leq$ *real* $d$ / *card* $S$
⟨*proof*⟩

**lemma** *degree-mpoly-to-poly* [*simp*]:

**assumes** *vars p* = {*x*}
**shows** *Polynomial.degree* (*mpoly-to-poly x p*) = *MPoly-Type.degree p x*
⟨*proof*⟩

**lemma** *total-degree-add*: *total-degree* (*x* + *y*) ≤ *max* (*total-degree x*) (*total-degree y*)
⟨*proof*⟩

**lemma** *total-degree-sum*:
**assumes** *finite S*
**shows***total-degree* (*sum f S*) ≤
*Max* ((*total-degree* ∘ *f*) ‘ *S*)
⟨*proof*⟩

**lemma** *coeff-mpoly-to-mpoly-poly-restrict*:
**shows** *coeff* (*mpoly-to-mpoly-poly x P*) *i* =
*sum* (λ*m*.
*MPoly-Type.monom* (*remove-key x m*)
(*MPoly-Type.coeff P m*) *when lookup m x* = *i*)
(*Poly-Mapping.keys* (*mapping-of P*))
⟨*proof*⟩

**lemma** *Max-le-Max*:
**assumes** *A* ≠ {}
**assumes** *finite A finite B*
**assumes** ⋀*a*. *a* ∈ *A* ⟹ ∃ *b*. *b* ∈ *B* ∧ *a* ≤ *b*
**shows** *Max A* ≤ *Max B*
⟨*proof*⟩

**lemma** *sum-lookup-remove-key*:
*sum* (*lookup* (*remove-key x y*)) (*keys* (*remove-key x y*)) + *lookup y x* =
*sum* (*lookup y*) (*keys y*)
⟨*proof*⟩

**lemma** *total-degree-nonzero*:
**assumes** *P* ≠ *0*
**shows** *total-degree P* =
*Max* ((λ*x*. *sum* (*lookup x*) (*keys x*)) ‘ *keys* (*mapping-of P*))
⟨*proof*⟩

**lemma** *poly-mapping-eq-iff*: (*m* = *m*′) ⟷ (∀ *i*. *lookup m i* = *lookup m*′ *i*)
⟨*proof*⟩

**lemma** *total-degree-coeff-mpoly-to-mpoly-poly*:
**assumes** *coeff* (*mpoly-to-mpoly-poly x P*) *i* ≠ *0*

**shows** *total-degree* (*coeff* (*mpoly-to-mpoly-poly x P*) *i*) + *i* ≤ *total-degree P*
⟨*proof*⟩

**lemma** *degree-le-total-degree*:
**shows** *MPoly-Type.degree P x* ≤ *total-degree P*
⟨*proof*⟩

**lemma** *insertion-update*:
**shows** *insertion* (*f*(*x* := *r*)) *P* = *poly* (*map-poly* (*insertion f*) (*mpoly-to-mpoly-poly x P*)) *r*
⟨*proof*⟩

**lemma** *measure-pmf-prob-dependent-product-bound*:
**assumes** *countable A* ⋀*i*. *countable* (*B i*)
**assumes** ⋀*a*. *a* ∈ *A* ⟹ *measure-pmf.prob N* (*B a*) ≤ *r*
**shows** *measure-pmf.prob* (*pair-pmf M N*) (*Sigma A B*) ≤ *measure-pmf.prob M A* ∗ *r*
⟨*proof*⟩

**lemma** *measure-pmf-prob-dependent-product-bound′*:
**assumes** *countable* (*A* ∩ *set-pmf M*) ⋀*i*. *countable* (*B i* ∩ *set-pmf N*)
**assumes** ⋀*a*. *a* ∈ *A* ∩ *set-pmf M* ⟹ *measure-pmf.prob N* (*B a* ∩ *set-pmf N*) ≤ *r*
**shows** *measure-pmf.prob* (*pair-pmf M N*) (*Sigma A B*) ≤ *measure-pmf.prob M A* ∗ *r*
⟨*proof*⟩

**lemma** *finite-set-pmf-Pi-pmf*:
**assumes** *finite A*
**assumes** ⋀*x*. *x* ∈ *A* ⟹ *finite* (*set-pmf* (*p x*))
**shows** *finite* (*set-pmf* (*Pi-pmf A def p*))
⟨*proof*⟩

**theorem** *schwartz-zippel*:
**fixes** *P* :: ('*a*::*idom*) *mpoly*
**fixes** *S* :: '*a set*
**assumes** *S*: *finite S S* ≠ {}
**assumes** *V*: *finite V*
**assumes** *P*: *total-degree P* ≤ *d P* ≠ *0 vars P* ⊆ *V*
**shows** *measure-pmf.prob* (*Pi-pmf V 0* (λ*i*. *pmf-of-set S*)) {*f*. *insertion f P* = *0*} ≤ *real d* / *card S*
⟨*proof*⟩

**end**

# 2 A Probabilistic Test for Perfect Matchings

**theory** *Rand-Perfect-Matching* **imports**
*Schwartz-Zippel*
*Jordan-Normal-Form.Determinant*
**begin**

We use a simple representation of bipartite graphs (with same no. vertices) $V$::*nat*, $E$::($nat \times nat$) *list* where $V$ is the number of vertices in each partition and $(x,y) \in E$ represents an edge between vertex $x$ in the left partition and vertex $y$ in the right partition.

**definition** *is-matching*::
($nat \times nat$) *set* $\Rightarrow$ ($nat \times nat$) *set* $\Rightarrow$ *bool*
**where** *is-matching E match* $\longleftrightarrow$
*match* $\subseteq E \wedge$
*inj-on fst match* $\wedge$
*inj-on snd match*

**definition** *has-perfect-matching*::
*nat* $\Rightarrow$ ($nat \times nat$) *set* $\Rightarrow$ *bool*
**where** *has-perfect-matching V E* $\longleftrightarrow$
($\exists$ *match. is-matching E match* $\wedge$ *card match* $= V$)

**definition** *adj-mat*::*nat* $\Rightarrow$ ($nat \times nat$) *set* $\Rightarrow$
*int mpoly mat*
**where** *adj-mat V E* $=$
*mat V V* ($\lambda(i,j)$.
*if* $(i,j) \in E$ *then Var* $(i*V+j)$ *else 0*)

**lemma** *adj-mat-square*[*simp*]:
**shows**
*dim-row* (*adj-mat V E*) $= V$
*dim-col* (*adj-mat V E*) $= V$
⟨*proof*⟩

**lemma** *perfect-match-set-map-fst*:
**assumes** $E \subseteq \{0..<V\} \times \{0..<V\}$
**assumes** *is-matching E match*
**assumes** *card match* $= V$
**shows** *fst* ‘ *match* $= \{0..<V\}$
⟨*proof*⟩

**lemma** *perfect-match-set-map-snd*:
**assumes** $E \subseteq \{0..<V\} \times \{0..<V\}$
**assumes** *is-matching E match*
**assumes** *card match* $= V$
**shows** *snd* ‘ *match* $= \{0..<V\}$
⟨*proof*⟩

**lemma** *is-matching-permutes*:
**assumes** $E \subseteq \{0..<V\} \times \{0..<V\}$
**assumes** *is-matching E match*
**assumes** *card match* $= V$
**obtains** $f$ **where**
$f$ *permutes* $\{0..<V\}$
$\bigwedge i.\ i < V \Longrightarrow (i, f\ i) \in E$
⟨*proof*⟩

**lemma** *Var-not-0*:
**shows** *Var* $x \neq (0::'a::idom\ mpoly)$
⟨*proof*⟩

**lemma** *sum-monom-0-iff*:
**assumes** *fin*: *finite S*
**and** *g*: $\bigwedge i\ j.\ i \in S \Longrightarrow j \in S \Longrightarrow g\ i = g\ j \Longrightarrow i = j$
**shows** *sum* $(\lambda\ i.$ *MPoly-Type.monom* $(g\ i)\ (f\ i))\ S = 0 \longleftrightarrow (\forall\ i \in S.\ f\ i = 0)$
(**is** *?l* $=$ *?r*)
⟨*proof*⟩

**lemma** *prod-Var*:
**assumes***finite S*
**shows** *prod* $(\lambda i.$ *Var*$(f\ i))\ S =$
*MPoly-Type.monom* (*sum* $(\lambda i.$ *monomial 1* $(f\ i))\ S)$ *1*
⟨*proof*⟩

**lemma** *det-adj-mat*:
**shows** *det* $($*adj-mat V E*$) =$
$(\sum p \mid p$ *permutes* $\{0..<V\}.$
*MPoly-Type.monom* (
*sum* $(\lambda i.$
*monomial 1* $(i * V + p\ i))\ \{0..<V\})$
(*if* $\forall i < V.\ (i,p\ i) \in E$ *then*
*of-int* (*sign p*)
*else 0*))
⟨*proof*⟩

**lemma** *vars-prod-Var*:
**assumes** *finite S*
**shows***vars* (*prod Var S*) $= S$
⟨*proof*⟩

**lemma** *prod-Var-eq*:
**assumes** *finite S finite T*
**assumes** *prod Var S* $=$ *prod Var T*
**shows** $S = T$
⟨*proof*⟩

**lemma** *pair-enc-eq*:
**assumes** $a * V + b = c * V + d$
**assumes** $b < V$ $d < V$
**shows** $b = (d::nat)$
⟨*proof*⟩

**lemma** *sum-monomial-eq*:
**assumes***f permutes* $\{0..<V\}$
**assumes***g permutes* $\{0..<V\}$
**assumes**
$(\sum i = 0..<V.$
*monomial* $(1::nat)$ $(i * V + (f\ i::nat))) =$
$(\sum i = 0..<V.$
*monomial* $(1::nat)$ $(i * V + g\ i))$
**shows** $f = g$
⟨*proof*⟩

**lemma** *perfect-matching-det*:
**assumes** $E \subseteq \{0..<V\} \times \{0..<V\}$
**assumes** *is-matching E match*
**assumes** *card match* $= V$
**shows***det* $(adj\text{-}mat\ V\ E) \neq 0$
⟨*proof*⟩

**lemma** *det-perfect-matching*:
**assumes** $E \subseteq \{0..<V\} \times \{0..<V\}$
**assumes** *det* $(adj\text{-}mat\ V\ E) \neq 0$
**obtains** *match* **where**
*is-matching E match*
*card match* $= V$
⟨*proof*⟩

**lemma** *has-perfect-matching-iff*:
**assumes** $E \subseteq \{0..<V\} \times \{0..<V\}$
**shows***has-perfect-matching V E* $\longleftrightarrow$ *det* $(adj\text{-}mat\ V\ E) \neq 0$
⟨*proof*⟩

**lemma** *sum-when-1*:
**assumes** *finite S* $x \in S$
**shows** $(\sum xa \in S.\ 1\ when\ xa = x) = 1$
⟨*proof*⟩

**lemma** *total-degree-monom*:
**assumes** *finite S*
**shows** *total-degree* (*MPoly-Type.monom* (*sum* $(\lambda i.\ monomial\ (Suc\ 0)\ i)\ S)\ c) =$
$(if\ c = 0\ then\ 0\ else\ card\ S)$
⟨*proof*⟩

**lemma** *total-degree-geI*:
**assumes** $m \in keys\ (mapping\text{-}of\ p)$ $(\sum v{\in}keys\ m.\ lookup\ m\ v) \geq n$
**shows** *total-degree* $p \geq n$
⟨*proof*⟩

**lemma** *total-degree-0-iff*: *total-degree* $p = 0 \longleftrightarrow vars\ p = \{\}$
⟨*proof*⟩

**lemma** *total-degree-0E*: *total-degree* $p = 0 \Longrightarrow (\bigwedge c.\ p = Const\ c \Longrightarrow P) \Longrightarrow P$
⟨*proof*⟩

**lemma** *total-degree-ex*:
**assumes** $p \neq 0$
**shows** $\exists\, m.\ m \in keys\ (mapping\text{-}of\ p) \wedge (\sum v{\in}keys\ m.\ lookup\ m\ v) = total\text{-}degree\ p$
⟨*proof*⟩

**lemma** *coeff-times-const-left* [*simp*]: *MPoly-Type.coeff* $(Const\ c * p)\ m = c * MPoly\text{-}Type.coeff\ p\ m$
⟨*proof*⟩

**lemma** *total-degree-times-const-left*: *total-degree* $(Const\ c * p) \leq total\text{-}degree\ p$
⟨*proof*⟩

**lemma** *total-degree-of-Const* [*simp*]: *total-degree* $(Const\ x) = 0$
⟨*proof*⟩

**lemma** *total-degree-of-int* [*simp*]: *total-degree* $(of\text{-}int\ x) = 0$
⟨*proof*⟩

**lemma** *total-degree-det-adj-mat*: *total-degree* $(det\ (adj\text{-}mat\ V\ E)) \leq V$
⟨*proof*⟩

**lemma** *arith*:
**assumes** $i < V$
**assumes** $j < (V{::}nat)$
**shows** $i * V + j < V\hat{}2$
⟨*proof*⟩

**lemma** *vars-det-adj-mat*:
**shows***vars* $(det\ (adj\text{-}mat\ V\ E)) \subseteq \{0..{<}V\hat{}2\}$
⟨*proof*⟩

**definition** *int-adj-mat*::
$(nat \Rightarrow int) \Rightarrow$
$nat \Rightarrow$
$(nat \times nat)\ set \Rightarrow$
$int\ mat$
**where** *int-adj-mat f V E* =

*mat V V* (λ(*i*,*j*).
*if* (*i*,*j*) ∈ *E then f* (*i*∗ *V* + *j*) *else 0*)

**lemma** *map-mat-prod-def*:
**shows***map-mat f A* ≡
*Matrix.mat* (*dim-row A*) (*dim-col A*)
(λ(*i*,*j*). *f* (*A* $$ (*i*,*j*)))
⟨*proof*⟩

**lemma** *int-adj-mat*:
**shows***int-adj-mat f V E* =
*map-mat* (*insertion f*) (*adj-mat V E*)
⟨*proof*⟩

**lemma** *det-int-adj-mat*:
**shows***det*(*int-adj-mat f V E*) =
*insertion f* (*det* (*adj-mat V E*))
⟨*proof*⟩

**definition** *test-perfect-matching* :: *int* ⇒ *nat* ⇒ (*nat* × *nat*) *set* ⇒ *bool pmf*
**where** *test-perfect-matching n V E* =
*do* {
*f* ← *Pi-pmf* ({*0*..<*V*^*2*}) *0* (λ*i*. *pmf-of-set* {*0*::*int*..<*n*});
*return-pmf* (*det* (*int-adj-mat f V E*) ≠ *0*)
}

**theorem** *test-perfect-matching-false-positive*:
**assumes** *E* ⊆ {*0*..<*V*} × {*0*..<*V*}
**assumes** ¬*has-perfect-matching V E*
**shows** *pmf* (*test-perfect-matching n V E*) *True* = *0*
⟨*proof*⟩

**lemma** *test-perfect-matching-true-negative*:
**assumes** *E* ⊆ {*0*..<*V*} × {*0*..<*V*}
**assumes** ¬*has-perfect-matching V E*
**shows** *pmf* (*test-perfect-matching n V E*) *False* = *1*
⟨*proof*⟩

**theorem** *test-perfect-matching-false-negative*:
**assumes** (*n*::*nat*) > *0*
**assumes** *E* ⊆ {*0*..<*V*} × {*0*..<*V*}
**assumes** *has-perfect-matching V E*
**shows** *pmf* (*test-perfect-matching n V E*) *False* ≤ *V* / *n*
⟨*proof*⟩

**end**

## References


[1] R. A. DeMillo and R. J. Lipton. A probabilistic remark on algebraic program testing. *Inf. Process. Lett.*, 7(4):193–195, 1978.

[2] R. Lipton. The curious history of the Schwartz-Zippel lemma, 2009. Accessed on Apr 21, 2023.

[3] J. T. Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *J. ACM*, 27(4):701–717, 1980.

[4] R. Zippel. Probabilistic algorithms for sparse polynomials. In E. W. Ng, editor, *EUROSAM*, volume 72 of *LNCS*, pages 216–226. Springer.