KRONECKER 0.166-9

Reference Manual

G. Lecerf

Université de Versailles St-Quentin-en-Yvelines,
45 avenue des Etats-Unis, Bâtiment Fermat,
78035 Versailles, France.

1 Introduction

1.1 Overview

Kronecker is a package for the Magma computer algebra system to solve polynomial systems of equations and inequations. It is a prototype resulting of a long term research by many people organized around the TERA project (`http://tera.medicis.polytechnique.fr`).

This version of Kronecker has been tested with Magma 2.9-2.

To begin smoothly with Kronecker read *Getting Started with Kronecker* (`http://kronecker.medicis.polytechnique.fr/doc/getstarted/getstarted.html`).

1.2 Credits

The present package has been originally designed by M. Giusti, G. Lecerf and B. Salvy, and written in Magma by G. Lecerf. The current main algorithm is the one presented in G. Lecerf's Phd thesis. The code contains contributions from E. Schost and L. Lehmann. A. Steel improved the basic arithmetic for bivariate polynomials in the Magma kernel. X. Suraud contributed to the MathML support.

1.3 Copying

The program Kronecker currently being distributed is to be used within the Magma computer algebra system distributed by the University of Sydney (Australia).

The Kronecker related packages are "free"; this means that everyone is free to use them and free to redistribute them on a free basis. The Kronecker related programs are not in the public domain, they belong to the CNRS (Centre National de la Recherche Scientifique).

*** THIS PACKAGE COMES WITH NO WARRANTY ***

2 Blackbox polynomials

This package provides a blackbox polynomial domain and is due to L. Lehmann. The verbose flag is `BbpVerbose`.

2.1 Creation of a blackbox polynomial algebra

BlackboxPolynomialAlgebra (R::`Rng`, n::`RngIntElt`) -> `RngMPol` intrinsic

It returns a ring of multivariate blackbox polynomials over R in n variables.

2.2 Creation of elements

Var (x::`RngMPolElt`) -> `Rec` intrinsic

It returns the blackbox polynomial element corresponding to x.

Cst (E::`RngMPol`, c) -> `Rec` intrinsic

- E is a domain returned by `BlackboxPolynomialAlgebra`.
- c is an element of the base ring of R.

It returns the blackbox polynomial corresponding to c.

2.3 Arithmetic operations

All elementary ring operations are supported.

'+' (x::`Rec`) -> `Rec` intrinsic

'+' (x::`Rec`, y) -> `Rec` intrinsic

'+' (y, x::`Rec`) -> `Rec` intrinsic

'-' (x::`Rec`) -> `Rec` intrinsic

'-' (x::`Rec`, y) -> `Rec` intrinsic

'-' (y, x::`Rec`) -> `Rec` intrinsic

'*' (x::`Rec`, y) -> `Rec` intrinsic

'*' (y, x::`Rec`) -> `Rec` intrinsic

'/' (x::`Rec`, c) -> `Rec` intrinsic

c must be invertible.

'^' (x::`Rec`, n::`RngIntElt`) -> `Rec` intrinsic

n must be a non-negative integer.

Add (E::RngMPol, l::[Rec]) -> Rec intrinsic

- E, returned by BlackboxPolynomialAlgebra.
- l, a sequence.

It returns the sum of the element of l.

Product (E::RngMPol, l::[Rec]) -> Rec intrinsic

- E, returned by BlackboxPolynomialAlgebra.
- l, a sequence.

It returns the product of the element of l.

2.4 Equality tests

Equality test corresponds to the test on the representations: two blackbox polynomials are equal iff their address is the same.

IsZero (e::Rec) -> BoolElt intrinsic

Tells whether the expression e is the constant 0 or not.

IsOne (e::Rec) -> BoolElt intrinsic

Tells whether the expression e is the constant 1 or not.

2.5 Evaluation

ClearRememberTableValues (~E::RngMPol) intrinsic

It clears the remember table of the current evaluation process.

Evaluate (F::Rec, x::[]) -> . intrinsic

Evaluate (F::[Rec], x::[]) -> . intrinsic

It returns the sequence of the values of the elements of F when the i-th variable is specialized to $x[i]$, for i from 1 to $\#x$. If F is empty then it returns [Universe(x)|].
Error conditions:

$\#x$ must equal the rank of the polynomial domain.

InitializeEvaluation (F::[Rec]) intrinsic

- F, a nonempty sequence of expressions.

It initializes the evaluation process that will serve to evaluate the element of F only.

ChangeUniverseValues (~E::RngMPol, R::Rng, f::UserProgram) intrinsic

- E, a domain returned by BlackboxPolynomialAlgebra.
- R, a ring
- f, map into R.

It applies f onto the remember table of the current evaluation process.

Derivative (e::`Rec`, i::`RngIntElt`: *algorithm*:=`"backward"`) -> `Rec` intrinsic

The derivative of e wrt its i-th variable.

Gradient (e::`Rec`: *algorithm*:=`"backward"`) -> [] intrinsic

Gradient (F::[`Rec`]: *algorithm*:=`"backward"`) -> [] intrinsic

Sequence of the partial derivatives. The parameter *algorithm* can be `"backward"` (for Baur-Strassen) or `"forward"`.

ClearRememberTableGradients (~E::`RngMPol`) intrinsic

It clears the remember tables used for gradient storage.

2.6 Degree, coefficients

ConvertToPolynomial (e::`Rec`) -> . intrinsic

ConvertToPolynomial (e::[`Rec`]) -> . intrinsic

- e is an expression or a sequence of expressions.

It returns the multivariate polynomial elements represented by e.

IsHomogeneous (e::`Rec`) -> `BoolElt` intrinsic

It tells whether e is homogeneous (w.r.t. to grading on the variables of its polynomial ring). It is not probabilistic.

TotalDegree (e::`Rec`: *Strategy*:=`"UpperBound"`) -> `RngIntElt` intrinsic

TotalDegree (F::[`Rec`]: *Strategy*:=`"UpperBound"`) -> . intrinsic

It returns the total degrees of the mulivariate polynomial represented by F.
Parameter: Strategy can be either `"Deterministic"` or `"Probabilistic"`. *Strategy* can also be `"UpperBound"` in order to compute a deterministic upper bound only.

2.7 Linear algebra

The following linear algebra functionalities are based on Berkowitz' algorithm.

CharacteristicPolynomialBerkowitz (A) -> . intrinsic

- A, sequence of sequences of expressions, viewed as a square matrix.

It returns the sequence of coefficients of the characteristic polynomial of A, the last element being the coefficient of degree 0.

DeterminantBerkowitz (A) -> . intrinsic

- A, sequence of sequences of Expressions, viewed as a square matrix.

It returns the determinant of A.

2.8 Printing

PrintBbp (*s*::[]) intrinsic

PrintBbp (*e*::`Rec`) intrinsic

It prints the expression represented by *e*.

3 Lifting fiber

This package provides functions for manipulating *lifting libers*. Lifting fibers encode equidimensional algebraic varieties.
Before all, a `BlackboxPolynomialAlgebra` domain must have been built this way:

$E, x[1], ..., x[n]$:= `BlackboxPolynomialAlgebra`(F, n);

where F is either the field of the rational numbers or a prime field. Note that it may work with other fields as soon as the characteristic is big enough and a factorization function exists in $F[T]$. A fiber encoding an r-equidimensional algebraic variety V is the following record:

- `ResolutionField`, a field K over which the resolution is defined. There must exist a coercion from F to K.
- `LiftingSystem`, a lifting system F for V, it is a sequence of blackbox polynomials.
- `PrimitiveElement`, a linear form u in the $x[i]$'s over K, it is a multivariate polynomial in variables.
- `MinimalPolynomial`, a sequence q of monic univariate polynomials in $K[T]$, of size s.
- `Denominator`, a sequence p of univariate polynomials in $K[T]$ of size s.
- `Parametrization`, a sequence w of size s of sequences of elements of $K[T]$ of size $n-r$.
- `MagicPoint`, a sequence P of elements in K of size r.
- `ChangeOfVariables`, a record `<LinearPart,AffinePart>`, `LinearPart` is an invertible square matrix M of size and `AffinePart` is a vector b of size n. Booth have entries in K.
- `IsMultiple`, boolean flag telling if V is multiple as a solution of $F = 0$.
- `GenericTrace`, generic trace of the deflation process.
- `ParentBbp` points to `E`.

Let y be the new variables defined by $x = M.y+b$, the following properties hold:

- V is a subvariety of the set of roots of $F = 0$.
- The variables y are in projective Noether position with respect to V.
- $y[1] = P[1], \ldots, y[r] = P[r]$ defines a finite fiber V' of V.
- The primitive element u separates the points of this fiber V'.
- The elements of q are monic and squarefree.
- The fiber V' is the union of the set of points described by the parametrizations:
  $q[l](T) = 0, p[l](T)y[r+1] = v[l](T), \ldots, p[l](T)y[n] = v[n](T)$, for l in $[1, \ldots, s]$.

We say the fiber is *isolated* if V is an isolated subvariety in the set of roots of the system $F = 0$. We say that the magic point is a *lifting point* if it satisfies the smoothess hypothesis of the fast deflation algorithm. The parametrization of is in the *Kronecker* presentation if $p[l] = q'[l]$ for all l and is in the *Shape-lemma* if $p[l] = 1$ for all l.

3.1 Creation of lifting fibers

LiftingFiber () -> `Cat` intrinsic

It returns the record format used to store lifting fibers.

WholeSpaceLF (*E*, *K*: *GenericLinearChangeOfVariables:=*`true`) -> . intrinsic

- *E*, a domain created by `BlackboxPolynomialAlgebra`.
- *K*, a field.

It returns a lifting fiber of the ambient space with resolution field *K*.
Parameter: If *GenericLinearChangeOfVariables* is set then a generic affine change of the coordinates is performed.

3.2 Getting properties

lf denotes a lifting fiber and *llf* a sequence of lifting fibers.

CodimensionLF (*lf*::`Rec`) -> `RngIntElt` intrinsic
It returns the codimension of the variety encoded by *lf*.

CodimensionLF (*llf*::[]) -> `RngIntElt` intrinsic
It returns the minimum of the codimensions of the elements of *llf*.
Error condition: llf must not be empty.

DegreeLF (*lf*::`Rec`) -> `RngIntElt` intrinsic
It returns the degree of the variety represented by *lf*.

DegreeLF (*llf*::[]) -> `RngIntElt` intrinsic
It returns the sum of the degrees of the element of *llf*.

DimensionLF (*lf*) -> `RngIntElt` intrinsic
See `RankLF`.

HasKroneckerParametrizationLF (*lf*::`Rec`) -> `BoolElt` intrinsic
It returns *true* if *lf* has a Kronecker parametrization.

HasKroneckerParametrizationLF (*llf*::[]) -> `BoolElt` intrinsic
It returns *true* if all the elements of *llf* have a Kronecker parametrization.

HasShapeLemmaParametrizationLF (*lf*::`Rec`) -> `BoolElt` intrinsic
It returns *true* if *lf* has a shape lemma parametrization.

HasShapeLemmaParametrizationLF (*llf*::[]) -> `BoolElt` intrinsic
It returns *true* if all the elements of *lf* have a shape lemma parametrization.

IsEmptyLF (*lf*::`Rec`) -> `BoolElt` intrinsic
It returns a boolean telling whether the variety represented by *lf* is empty or not.

IsEmptyLF (*llf*::[]) -> `BoolElt` intrinsic
It returns a boolean telling whether all the elements of *llf* represent the empty variety or not.

IsMultipleLF (*lf*::`Rec`) -> `BoolElt` intrinsic

It returns a boolean telling whether *lf* is a multiple component or not.

IsWholeSpaceLF (*lf*::`Rec`) -> `BoolElt` intrinsic

It returns whether *lf* represents the ambient space or not.

IsWholeSpaceLF (*llf*::[]) -> `BoolElt` intrinsic

It returns whether at least one of the elements of *llf* represents the ambient space or not.

NumberOfVariablesLF (*lf*::`Rec`) -> `RngIntElt` intrinsic

It returns the dimension of the ambient space in which *lf* lives.

NumberOfFactorsLF (*lf*::`Rec`) -> `RngIntElt` intrinsic

It returns the number of factors *lf*, that is the cardinal of *lf*`‘MinimalPolynomial`.

ParentBbpLF (*lf*::`Rec`) -> `Cat` intrinsic

It returns the Multivariate Polynomial Algebra associated to the blackbox polynomial domain of the lifting system of *lf*.

RankLF (*lf*::`Rec`) -> `RngIntElt` intrinsic

It returns the dimension of the variety encoded by *lf*.

RankLF (*llf*::[]) -> `RngIntElt` intrinsic

It returns the maximum of the dimensions of the elements of *llf*.
Error condition: llf must not be empty.

3.3 Basic operations

lf denotes a lifting fiber and *llf* a sequence of lifting fibers.

ChangeResolutionFieldLF (~*lf*::`Rec`, K) intrinsic

- *lf*, a `LiftingFiber`.
- K, a field.

If there exists a coercion from *lf*`‘ResolutionField` to K, then it modifies *lf* to be a resolution over K.
Error condition: There must exist a coercion from *lf*`‘ResolutionField` to K.

ChangeResolutionFieldLF (~*llf*::[], K) intrinsic

It iterates `ChangeResolutionFieldLF` over *llf*.

ChangeResolutionFieldLF (~*lf*::`Rec`, K, f) intrinsic

- *lf*, a `LiftingFiber`.
- K, a field.
- f, a homomorphism from *lf*‘*ResolutionField* to K.

Applies f on *lf*`‘ResolutionField` to coerce it to K via f, so that in return *lf* is a resolution over K.

ChangeResolutionFieldLF (~*llf*::[], *K*, *f*) intrinsic

It iterates `ChangeResolutionFieldLF` over *llf*.

MakeKroneckerParametrizationLF (~*lf*::`Rec`) intrinsic

lf is changed to a Kronecker parametrization.

MakeKroneckerParametrizationLF (~*llf*::[]) intrinsic

It iterates `MakeKroneckerParametrizationLF` over each element of *llf*.

MakeMonicLF (~*lf*::`Rec`) intrinsic

It makes *lf*`‘MinimalPolynomial` monic.

MakeMonicLF (~*llf*::[]) intrinsic

It iterates `MakeMonicLF` over each element of *llf*.

MakeShapeLemmaParametrizationLF (~*lf*::`Rec`) intrinsic

It changes the parametrization of *lf* into a Shape Lemma form.
Error condition: If *lf*`‘Denominator` is not invertible modulo *lf*`‘MinimalPolynomial`. then *lf* contains a string in return.

MakeShapeLemmaParametrizationLF (~*llf*::[]) intrinsic

It iterates `MakeShapeLemmaParametrizationLF` over each element of *llf*.

3.4 Changes of fibers

ChangeAlgebraicVariablesLF (~*lf*::`Rec`, *N*, *c*) intrinsic

- *lf*, a `LiftingFiber` of dimension r in a n-dimensional space.
- *N*, a square matrix of size $n - r$ with rational numbers entries.
- *c*, a $n - r$-vector with rational numbers entries.

Let M and b be `lf‘ChangeOfVariables`, such that $x = My{+}b$. The procedure changes the coordinates of *lf* in the following way: $M := M.(Idr|N)$ and $b := M.(0|c){+}b$. This change of algebraic variables is applied in consequence to the parametrization and the primitive element so that *lf* remains consistent.
Error condition: The procedure raises an error if N is not invertible.

ChangeAlgebraicVariablesLF (~*llf*::[], *N*, *c*) intrinsic

It applies `ChangeAlgebraicVariablesLF` to each element of *llf*.

ChangeBackAlgebraicVariablesLF (~*lf*::`Rec`) intrinsic

- *lf*, a `LiftingFiber`.

In return *lf* has the identity for the part of the change of variables corresponding to the algebraic variables.

ChangeBackAlgebraicVariablesLF (~*llf*::[]) intrinsic

It applies `ChangeBackAlgebraicVariablesLF` to each element of *llf*.

ChangeBackFreeVariablesLF (~*lf*::`Rec`) intrinsic

- *lf*, a `LiftingFiber`.

In return *lf* has the identity in the part of its change of variables corresponding to the free variables.

ChangeBackFreeVariablesLF (~*llf*::[]) intrinsic

It iterates `ChangeBackFreeVariablesLF` on each element of *llf*.

ChangePrimitiveElementLF (~*lf*::`Rec`, u) intrinsic

- *lf*, a `LiftingFiber` of dimension r in a n-dimensional space.
- u, a linear form given as a multivariate polynomial.

It Changes `lf‘PrimitiveElement` to u, if it is actually a primitive element.
Error condition: The procedure returns `"Bad primitive element"` in *lf* if u is not a primitive element.

ChangePrimitiveElementLF (~*llf*::[], u) intrinsic

It applies `ChangePrimitiveElementLF` to each element of *llf*.

TranslateKthFreeVariableToZeroLF (~*lf*::`Rec`, k) intrinsic

- *lf*, a `LiftingFiber`.

Let $y[k]$ be the kth free variable of *lf*, the procedure modifies *lf*‘`ChangeOfVariables` replacing $y[k]$ by $y[k]$+*lf*‘`MagicPoint`$[k]$. Then `lf‘MagicPoint[k]` is set to 0.

TranslateKthFreeVariableToZeroLF (~*llf*::[], k) intrinsic

It iterates `TranslateKthFreeVariableToZeroLF` on each element of *llf*.

TranslateAllFreeVariablesToZeroLF (~*lf*::`Rec`) intrinsic

It applies `TranslateKthFreeVariableToZeroLF` to all the free variables of *lf*.

TranslateAllFreeVariablesToZeroLF (~*llf*::[]) intrinsic

It applies `TranslateAllFreeVariablesToZeroLF` to each element of *llf*.

TranslateLastFreeVariableToZeroLF (~*lf*::`Rec`) intrinsic

It applies `TranslateKthFreeVariableToZeroLF` to all the last free variables of *lf*.

TranslateLastFreeVariableToZeroLF (~*llf*::[]) intrinsic

It applies `TranslateLastFreeVariableToZeroLF` to each element of *llf*.

3.5 Evaluation

lf denotes a lifting fiber and *llf* a sequence of lifting fibers.

EvaluateLF (*lf*::`Rec`, *f*::`Rec`, *x*) `-> .` intrinsic

EvaluateLF (*lf*::`Rec`, *f*::[`Rec`], *x*::[]) `-> .` intrinsic

- *lf*, a `LiftingFiber`.
- *f*, a sequence of black box polynomials.
- *x*, a sequence of values.

It returns the sequence of the values of *f* evaluated on the point *x*.
Error conditions:

- `#`*x* must be equal to `NumberOfVariablesLF(`*lf*`)`.
- `BaseRing(ParentBbpLF(`*lf*`))` must be coercible to `Universe(`*x*`)`.

VerifyLF (*lf*::`Rec`: *Strategy*:=`"Probabilistic"`) `-> BoolElt` intrinsic

It returns `true` iff the parametrization of *lf* satisfies its lifting system modulo its `MinimalPolynomial`.
Parameter: Strategy, string:

- `""`, the verification is perfomed over *lf*`‘ResolutionField`.
- `"Probabilistic"`, the verification is probabilistic. Computations are done modulo a random prime number if the resolution field is the field of the rational numbers. If the resolution field is a rational function field, the variables are specialized at random.

VerifyLF (*llf*::[]: *Strategy*:=`"Probabilistic"`) `-> BoolElt` intrinsic

It tells whether all the elements of *llf* satisfies their lifting systems.

3.6 Splittings

SplitLF (*lf*::`Rec`, *f*::[`Rec`]) `-> Rec,Rec` intrinsic

SplitLF (*lf*::`Rec`, *f*::`Rec`) `-> Rec,Rec` intrinsic

- *lf*, a lifting fiber.
- *f*, an element of `BlackboxPolynomialAlgebra`.

It returns two fibers *lfz*, *lfnz*. The first one represents the points of *lf* satisfying *f*`=0` and the second one the other points.

CleanLF (`~`*lf*::`Rec`, *ineqs*::`Rec`) intrinsic

Removes the points of *lf* satisfying *ineqs*`=0`.

CleanLF (`~`*lf*::`Rec`, *ineqs*::[]) intrinsic

Removes the points of *lf* satisfying *ineqs*`=0`.

CleanLF (~*llf*::[], *ineqs*) intrinsic

It iterates `CleanLF` over each element of *llf*.

FactorizationLF (~*lf*::`Rec`) intrinsic

lf is factorized.

Error condition: The parametrization must be shape lemma.

CombineLF (~*lf*::`Rec`: *Parametrization:=*`"Unknown"`) intrinsic

It combines the factors of *lf*. The parameter can be `"Unknown"` (default), `"ShapeLemma"` or `"Kronecker"` according to the properties known about *lf*.

MergeLF (*llf*::[]) `-> Rec` intrinsic

It merges the elements of *llf* into one.

Error conditions:

- The elements of *llf* must share the same `ResolutionField`, *LiftingSystem*, *PrimitiveElement*, *MagicPoint*, *ChangeOfVariables*, *GenericTrace* and *ParentBbp*.
- *llf* must not be empty.

3.7 Printing

PrintLF (*lf*::`Rec`) intrinsic

It prints *lf* on stdout.

PrintLF (*llf*::[]) intrinsic

Prints *llf* on stdout.

4 Lifting

This package provides an implementation of the global Newton lifting algorithm. Its verbose flag is `HenselVerbose`.

4.1 Splitting before lifting

LiftSplitLF (*lf*::`Rec`) -> `SeqEnum` intrinsic

- *lf*, is an isolated lifting fiber.

It returns a sequence of isolated lifting fibers corresponding to different subvarieties behaving differently wrt the the lifting process. This splitting must be achieved before performing any lifting.

4.2 Lift curves

Common requirements for all the lifting functions:

- *lf* must be shape lemma.
- The variety must be isolated wrt to the *lf*‘`LiftingSystem`.
- *lf* must be irreducible with respect to the lifting process.

LiftCurveLF (~*lf*::`Rec`, *destpoint*::`SeqEnum`, *precision*::`RngIntElt`) intrinsic

- *lf*, an isolated lifting fiber.
- *destpoint*, destination point.
- *precision*, integer.

This procedure lifts the curve from the lifting point to the destination point *destpoint* and up to *precision*.
Error condition: cf. common requirements above.

LiftCurveLF (~*llf*::[], *destpoint*::`SeqEnum`, *precision*::`RngIntElt`) intrinsic

It iterates `LiftCurveLF` over each element of *llf*.

LiftLastFreeVariableLF (~*lf*::`Rec`) intrinsic

- *lf*, an isolated lifting fiber.

This procedure lifts the last free variable of *lf*. *lf*‘`ResolutionField` becomes an univariate rational function field over the resolution field.
Error condition: cf. common requirements above.

LiftLastFreeVariableLF (~*llf*::[]) intrinsic

It iterates `LiftLastFreeVariableLF` over each element of *llf*.

4.3 Change the magic point

ChangeMagicPointLF (~*lf*::`Rec`, *magicpoint*) intrinsic

- *lf*, an isolated lifting fiber.
- *magicpoint*, the destination magic point.

Computes the lifting fiber for the magic point *magicpoint* and returns it in *lf*.
Error conditions:

- `#`*magicpoint* must be equal to the dimension of *lf*
- cf. common requirements above.

ChangeMagicPointLF (~*llf*::[], *magicpoint*) intrinsic
It iterates `ChangeMagicPointLF` over each element of *llf*.

4.4 Check lifting

HasLiftingPointLF (*lf*::`Rec`) `-> BoolElt` intrinsic

- *lf*, an isolated fiber.

It tells whether the magic point is a lifting point. If the variety is multiple wrt its lifting system then its generic trace must be known.
Error conditions:

- cf. common requirements above.

HasLiftingPointLF (*llf*::[]) `-> BoolElt` intrinsic
It iterates `HasLiftingPointLF` over each element of *llf*.

5 Equidimensional decomposition

This section provides functionalities to handle equidimensional decomposition. The verbose flag is `GeometricSolveVerbose`.

5.1 Inclusion of components

IsIncludedIrreducibleLF (*lf1*, *l*, *lf2*::`Rec`) `-> BoolElt` intrinsic

- *lf1* is a fiber.
- *l* is an integer.
- *lf2* is an isolated lifting fiber.

The function tells whether the *l*th irreducible factor of *lf1* is included in *lf2* or not. If it is detected that *lf2* is not an isolated lifting fiber then a string is returned.

IsIncludedIrreducibleLF (*lf*, *l*, *llf*::`SeqEnum`) `-> BoolElt` intrinsic

The function iterates `IsIncludedIrreducibleLF` over each element of *llf* and tells whether *lf*`[l]` is included in *llf*.

5.2 Set difference

DifferenceLF (~*lf1*, *lf2*::`Rec`) intrinsic

- *lf1* is a fiber.
- *lf2* is an isolated lifting fiber.

In return *lf1* contains its only components that are not included in *lf2*.
Error condition: If it is detected that *lf2* is not an isolated lifting fiber then *lf1* contains a string in return.

DifferenceLF (~*lf*, *llf*::`SeqEnum`) intrinsic

The procedure iterates `DifferenceLF` over each element of *llf*, so that *lf* only contains its components that are not included in *llf* in return.

5.3 Minimization

MinimizeLLF (~*llf*: *RemoveMultipleComponents:=*`false`) intrinsic

llf is an equidimenstional decomposition that may be redundant. In return *llf* does not contain redundant components. It is important that *llf*`[i]` contains the components of dimension `i+1`. In case of bugs *llf* may contain a string in return.

5.4 Intersection

IntersectLF (*llf*::`SeqEnum`, *f*::`Rec`, *h*::[`Rec`]: *RemoveMultipleComponents*:=`false`) -> `SeqEnum` intrinsic

- *llf*, a minimal sequence of sequences lifting fibers encoding an equidimensional decomposition of an algebraic variety.
- *f*, a blackbox polynomial.
- h, a sequence of blackbox polynomials.

It returns the minimal equidimensional decomposition for the intersection of the input variety given by *llf* with the hypersurface defined by $f = 0$ and outside $h = 0$.
Error condition: In case of problems (bug or unlucky choices) a string is returned.

6 Geometric Solve

6.1 Introduction

This package provides the main functionalities of Kronecker. The verbose flag is `GeometricSolveVerbose`.

6.2 Verbosity

KroneckerInformations () intrinsic

It displays informations about Kronecker package: version, author, date...

KroneckerVersion () -> `MonStgElt` intrinsic

KroneckerSetVerbose (*l*::`RngIntElt`) intrinsic

Make Kronecker verbose. Argument *l* is an integer between 0 and 5.

KroneckerSetVerbose () intrinsic

Set to default verbosity 2.

KroneckerSetMathMLVerbose () intrinsic

Make Kronecker verbose via MathML.

KroneckerUnsetMathMLVerbose () intrinsic

Stop Kronecker verbose via MathML.

6.3 Geometric solve functionalities

GeometricSolve (*equations*::`SeqEnum`, *inequations*::`SeqEnum`, *K*: *GenericLinearChangeOfVariables*:= `true`, *RemoveMultipleComponents*:=`false`) -> `Rec` intrinsic

- *equations*.
- *inequations*.
- *K*, field.

It returns a sequence of sequences *llf* of lifting fibers. *llf* describes the variety defined by the *equations* outside the *inequations*, if everything goes right. The computations are performed over *K*.
Error condition: Since the algorithm is probabilistic an error can be raised.

GeometricSolve (*equations*::`SeqEnum`, *inequations*::`SeqEnum`) -> `Rec` intrinsic

GeometricSolve (*equations*::`SeqEnum`) -> `Rec` intrinsic

Function Index

P

R

S

T

V

W

Table of Contents