UNIVERSITÀ DEGLI STUDI DI MESSINA

Introduzione al *Macaulay2*

Luca Amata

Anno Accademico 2020 - 2021

Indice

Introduzione

Il *Macaulay2* ([1]) è un sistema di computer algebra che permette, in particolare, di svolgere operazioni simboliche specializzate nel contesto dell'algebra commutativa. Oltre alle istruzioni di base sono presenti varie librerie per la gestione di problemi specifici.
Il sito ufficiale dove trovare tutte le informazioni, sia per i comandi da utilizzare sia per il software da installare, è raggiungibile alla pagina:

`https://faculty.math.illinois.edu/Macaulay2/`.

La guida ufficiale contiene dei tutorial, dal livello introduttivo a quello specialistico, per chi ha bisogno di imparare il linguaggio ma anche di spiegazioni su un utilizzo avanzato per la risoluzione di problemi complessi. Inoltre è presente un indice per poter consultare le singole istruzioni disponibili. Può essere consultata alla pagina:

`https://faculty.math.illinois.edu/Macaulay2/doc/Macaulay2-1.15/share/doc/Macaulay2/Macaulay2Doc/html/`.

Per trovare istruzioni dettagliate per l'installazione di *Macaulay2* su diverse piattaforme, ci si può riferire alla pagina:

`https://faculty.math.illinois.edu/Macaulay2/Downloads/`

È anche possibile utilizzare *Macaulay2* nella versione online: non è necessario installare alcunché, ma potrebbe essere richiesta un'iscrizione al portale (gratuita).

(1) `http://habanero.math.cornell.edu:3690/`

(2) `https://www.unimelb-macaulay2.cloud.edu.au/#home`

(3) `https://cocalc.com/` (Run CoCalc now, Notebook: Terminal, M2)

1 Linguaggio base

Il *Macaulay2* rispetta le direttive standard dei linguaggi di programmazione. È un linguaggio procedurale e *case sensitive*, dispone dei comuni operatori, permette

l'utilizzo di variabili e la costruzione di funzioni. Ad ogni accesso vengono dichiarate le librerie non standard richiamate automaticamente:

```
Macaulay2, version 1.17.2.1
with packages: ConwayPolynomials, Elimination, IntegralClosure, LLLBases,
               InverseSystems, MinimalPrimes, ReesAlgebra, Saturation,
               TangentCone, PrimaryDecomposition
i1 :
```

Il prompt dei comandi è numerato e per inizializzarlo nuovamente bisogna eseguire il reset dell'ambiente di sviluppo.

Per utilizzare **librerie** (packages) non caricate automaticamente all'avvio del client, si può utilizzare il seguente comando:

```
i1 :  loadPackage "<libreria>"
```

Nel caso in cui si tratti di librerie costruite localmente bisogna assicurarsi si effettuare l'upload del file tramite browser (nel caso on-line) o di inserire il file nella cartella dedicata alle librerie (nel caso di installazione locale).

Istruzioni di gestione dell'ambiente di sviluppo molto utili sono `help <istr>`, che restituisce una documentazione di aiuto relativa all'istruzione `<istr>`, e `clearAll` che elimina dalla memoria tutte le variabili dichiarate e i risultati ottenuti.

1.1 Numeri e operazioni

Gestisce gli insiemi numerici comuni in maniera automatica: `ZZ, QQ, RR, CC` (l'unità immaginaria viene identificata con `ii`). La priorità viene data agli insiemi più "piccoli": quella massima è data agli interi. Per riferirsi ad un elemento di un insieme specifico, ad esempio dei Reali, si può utilizzare l'operatore `_RR`: quindi `3_RR` sarà un'unità di $\mathbb{R}$. Le istruzioni `promote(x,R)` e `lift(x,R)` permettono di immergere/proiettare l'elemento `x` nell'anello `R`.
Le operazioni (`+, -, *, /, ^`) non sono mai sottintese e devono essere sempre esplicitate, anche il prodotto `*`. Sono presenti le funzioni e le costanti matematiche stan-

dard (`sqrt()`, `pi`, etc). Sono disponibili anche alcune funzioni di calcolo combinatorio (`partitions, combinations, binomial`, etc...) e per ottenere valori casuali (`random`). Le parentesi sono obbligatorie soltanto quando l'argomento è formato da più simboli.

L'operatore `^i`, applicato ad anelli, permette di ottenere il modulo libero di rango `i` degli insiemi numerici (`ZZ^2`, `CC^4`, etc...), mentre i quozienti di `ZZ` tramite numeri primi si ottengono tramite l'operatore `/` (`ZZ/2`, `ZZ/7`, etc...). L'operatore `**` permette di ottenere il prodotto cartesiano/tensoriale fra due strutture. I campi finiti, in generale, possono essere ottenuti tramite l'istruzione `GF(p,n)` che genera il campo finito di ordine p^n (p primo).

Le stringhe sono identificate dalle virgolette `"prova"`, l'operatore `#i` restituisce il carattere alla `i`-sima posizione ed esistono alcune funzioni per la loro gestione (`ascii`, `substring`, etc...). Tutto ciò che non fa parte del linguaggio o che non è stato assegnato in precedenza viene identificato come simbolo esterno.

Ad esempio:

```
i1 : 2*(3+5)
o1 = 16
i2 : 3/2
     3
o2 = -
     2
o2 : QQ
i3 : sqrt 2
o3 = 1.4142135623731
o3 : RR (of precision 53)
i3 : sqrt(-1)
o3 = ii
o3 : CC (of precision 53)
i4 : x
o4 = x
o4 : Symbol
```

Una particolare variabile d'ambiente `oo` viene aggiornata ad ogni operazione effettuata con successo in modo da contenere il valore precedentemente calcolato.

```
i1 : cos numeric pi
o1 = -1
o1 : RR (of precision 53)
i2 : sqrt oo
o2 = ii
o2 : CC (of precision 53)
```

Così come nel linguaggio *C*, l'operatore `=` serve ad eseguire assegnazioni, mentre l'operatore di confronto qualitativo è dato da `==`. È presente un ulteriore operatore di confronto `===` che valuta tecnicamente l'uguaglianza delle strutture così come sono state definite. L'operatore binario `?` restituisce la relazione esistente fra due quantità, se confrontabili.

Il *Macaulay2* permette di definire delle funzioni personalizzate tramite l'operatore `->`. La sintassi prevede di solito l'utilizzo di un'assegnazione per determinare il nome della funzione (prima dell' `=`). La parte che precede `->` indica l'insieme delle variabili indipendenti (o parametri), quella che segue l'operatore specifica le operazioni che la funzione deve svolgere per ottenere l'output:

```
<nome>=(<p1,..., pn>) -> (<istr1>; ...; <istrk>; return <var>)
```

```
i1 : f=(a,b)->(r=sqrt(a^2+b^2); return r)
o1 = f
o1 : FunctionClosure
i2 : f(1,1)
o2 = 1.4142135623731
o2 : RR (of precision 53)
i3 : f(3,4)
o3 = 5
o3 : RR (of precision 53)
```

Nel caso in cui si utilizzi l'ambiente online del *Macaulay2*, è possibile scrivere una funzione in un file di testo per poi incollarla nella linea di comando, oppure utilizzando l'apposito editor se previsto. Una volta eseguito il codice della funzione, esso rimarrà attivo in memoria per tutta la sessione e potrà quindi essere riutilizzato. Nel caso in cui si abbia la necessità di far restituire alla funzione più valori, è possibile creare una lista `L` (come vedremo fra poco) all'interno della funzione con tutti gli output desiderati e inserire come ultima istruzione della funzione: `return L`.

1.2 Strutture dati

La struttura dati fondamentale è quella della **lista**, un vettore ordinato di elementi indicizzati a partire da 0. Esistono anche diverse varianti, come le sequenze, i vettori o le liste aggiornabili; sono presenti soltanto delle piccole variazioni in base al contesto nelle quali si utilizzano. Per ottenere il numero di elementi di una lista `L`, basta premettere al suo nome l'operatore `#`: `#L`. Per ottenere invece l'elemento al posto i-esimo basta postporre al suo nome l'operatore `#i`: `L#i` (quest'ultimo funziona in modo circolare, modulo la lunghezza della lista). L'operatore `#i` in sola lettura può essere sostituito in alcuni casi da `_i`.
L'istruzione `m..n` crea la sequenza `(m,m+1,...,n-1,n)`.
Esistono degli operatori di conversione per passare da una struttura ad un'altra: `toList()`, `toSequence()`, etc...

```
i1 : l={a,1,ii}
o1 = {a, 1, ii}
o1 : List
i2 : s=(a,1,ii)
o2 = (a, 1, ii)
o2 : Sequence
i2 : v=[a,1,ii]
o2 = [a, 1, ii]
o2 : Array
```

```
i3 : #l
o3 = 3
i4 : s#2
o4 = ii
o4 : Constant
i5 : v#(-1)
o5 = ii
o5 : Constant
i6 : l_1
o6 = 1
```

Queste tipologie di liste non permettono l'aggiornamento di un singolo elemento, per poterlo fare bisogna utilizzare liste variabili: `MutableList`. Possono essere dichiarate tramite l'operatore `new` e visualizzate tramite `peek`. In questo caso il singolo elemento della lista `M` può essere utilizzato sia in lettura che in scrittura utilizzando il riferimento `M#i`.

```
i1 : m = new MutableList from {a,b,c}
o1 = MutableList{...3...}
o1 : MutableList
i2 : peek m
o2 = MutableList{a, b, c}
i3 : m#1=1
o3 = 1
i4 : m#2=ii
o4 = ii
o4 : Constant
i5 : peek m
o5 = MutableList{a, 1, ii}
i6 : toList(m)===l
o6 = true
```

È possibile eseguire varie manipolazioni sulle liste. D'ora in avanti faremo riferimento alla lista generica {}, ma le istruzioni varranno per le diverse tipologie viste.

Data una lista `L` è possibile richiamare le seguenti istruzioni, osservando che nessuna di esse modificherà automaticamente la lista `L`: volendo conservarne il risultato si dovrà provvedere manualmente all'assegnazione.

(-) `append(L,x)`: aggiunge l'elemento `x` in coda

(-) `prepend(x,L)`: aggiunge l'elemento `x` in testa

(-) `insert(n,x,L)`: inserisce l'elemento `x` nella posizione `n`

(-) `switch(m,n,L)`: inverte l'elemento di posto `m` con quello di posto `n`

(-) `delete(x,L)`: elimina tutti gli elementi che hanno valore uguale a `x`

(-) `drop(L,m,n)`: elimina gli elementi della sottolista fra le posizioni `m` ed `n`

(-) `take(L,m,n)`: restituisce gli elementi della sottolista fra le posizioni `m` ed `n`

(-) `unique(L)`: restituisce gli elementi della lista uno per ciascun valore

(-) `reverse(L)`: inverte l'ordine degli elementi

(-) `sort(L), rsort(L)`: ordina in modo crescente/descrescente la lista

(-) `flatten L`: elimina le eventuali nidificazioni (di un livello)

(-) `join(L,M) o L|M`: data un'altra lista `M`, restituisce la concatenazione delle due liste

Ecco alcuni esempi di utilizzo:

```
i1 : L = toList(1..10)
o1 = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
o1 : List
i2 : L=prepend(10,L)
```

```
o2 = {10, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
o2 : List
i3 : L=delete(10,L)
o3 = {1, 2, 3, 4, 5, 6, 7, 8, 9}
o3 : List
i4 : M=take(L,{2,6})
o4 = {3, 4, 5, 6, 7}
o4 : List
i5 : M=M|{{8,{9}}}
o5 = {3, 4, 5, 6, 7, {8, {9}}}
o5 : List
i6 : flatten M
o6 = {3, 4, 5, 6, 7, 8, {9}}
o6 : List
i7 : flatten flatten M
o7 = {3, 4, 5, 6, 7, 8, 9}
o7 : List
```

Sulle liste è anche possibile effettuare delle operazioni logico-aritmetiche che tengano conto della natura degli elementi contenuti in esse:

(-) `max L` (o `maxPosition L`): restituisce il valore (o l'indice del valore) massimo di `L`

(-) `min L` (o `minPosition L`): restituisce il valore (o l'indice del valore) minimo di `L`

(-) `L+M` (o `L-M`): `L,M` numeriche della stessa lunghezza, restituisce una lista contenente le somme (o le differenze) termine a termine degli elementi di `L` e `M`

(-) `n*L`: `L` numerica, restituisce una lista contenente i prodotti di `n` per ogni elemento di `L` (Attenzione: `L*n` restituirà errore)

(-) `apply(L,f) o L/f o f\L`: restituisce una lista applicando la funzione `f` ad ogni elemento di `L`

(-) `select(L,cond)`: restituisce una lista selezionando esclusivamente gli elementi che rendono vera la funzione booleana `cond`. Esistono alcune funzioni predefinite utili a tale scopo (`even, odd`, etc) o se ne possono costruire (ad es. `(i->i>0)`)

(-) `position(L,cond)` (o `positions(L,cond)`): restituisce la prima posizione (o la lista di tutte le posizioni) dell'elemento della lista che rende (o rendono) vera la funzione booleana `cond`

(-) `number(L,cond)`: conta il numero di elementi della lista che rendono vera la funzione booleana `cond`

(-) `sum L`: somma fra loro gli elementi della lista, se possibile

(-) `product L`: moltiplica fra loro gli elementi della lista, se possibile

(-) `isSubset(L,M)`: funzione booleana che verifica se `L` è contenuta in `M`

Alcune implementazioni:

```
i1 : L = toList(1..10)
o1 = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
o1 : List
i2 : L=L/(x->x^2)
o2 = {1, 4, 9, 16, 25, 36, 49, 64, 81, 100}
o2 : List
i3 : number(L,odd)
o3 = 5
i4 : M=select(L,x->x<50 and x%2==1)
o4 = {1, 9, 25, 49}
o4 : List
i5 : positions(M,even)
```

```
o5 = {}
o5 : List
```

Un'altra struttura comunemente utilizzata, derivata dalle liste, è quella **matriciale**. Per definire una matrice basta utilizzare l'istruzione `matrix` passandogli una lista opportunamente strutturata, cioè che contenga delle sottotliste (righe) tutte delle stessa lunghezza (colonne). Sulle matrici è possibile utilizzare operazioni e funzioni per la loro gestione:

(-) `+, -, *, ^,...`: operazioni estese all'anello delle matrici

(-) `**`: effettua il prodotto tensoriale fra due matrici

(-) `A|B` (o `A||B`): concatena due matrici compatibili per colonne (o per righe)

(-) `A_{indici}` (o `A^{indici}`): estrae dalla matrice le colonne (o le righe) individuate dagli indici nella lista

(-) `rank A`: calcola il rango di una matrice

(-) `det A`: calcola il determinante di una matrice quadrata

(-) `transpose A`: restituisce la matrice trasposta

(-) `inverse A`: restituisce la matrice inversa, quando possibile

Alcuni semplici esempi:

```
i1 : A = matrix {{1,2,3},{4,5,6}}
o1 = | 1 2 3 |
     | 4 5 6 |
               2        3
o1 : Matrix ZZ  <--- ZZ
i2 : B=A|A
o2 = | 1 2 3 1 2 3 |
     | 4 5 6 4 5 6 |
```

```
                2        6
o2 : Matrix ZZ  <--- ZZ
i3 : C=B||B
o3 : | 1 2 3 1 2 3 |
     | 4 5 6 4 5 6 |
     | 1 2 3 1 2 3 |
     | 4 5 6 4 5 6 |
                4        6
o3 : Matrix ZZ  <--- ZZ
i4 : det C_{0..3}
o4 = 0
i5 : rank C_{0..3} == rank B
o5 = true
```

Il *Macaulay2* considera le matrici come trasformazioni lineari fra spazi vettoriali, dunque è possibile utilizzare le seguenti istruzioni per ottenere informazioni aggiuntive:

(-) `source A`: restituisce il dominio della trasformazione lineare

(-) `target A`: restituisce il codominio di `A`

(-) `numgens source (target) A`: restituisce il numero di generatori, come modulo libero, del dominio (codominio) di `A`

(-) `ring A`: restituisce l'anello a cui appartengono le entrate della matrice

(-) `entries A`: restituisce le entrate della matrice in forma di lista

(-) `describe A`: restituisce tutte le informazioni dettagliate (e tecniche) della matrice e della struttura utilizzata

Qualche esempio:

```
i1 : A = random(ZZ^3,ZZ^2)
o1 = | 9 6 |
```

```
      | 4 3 |
      | 2 4 |
               3        2
o1 : Matrix ZZ  <--- ZZ
i2 : target A
       3
o2 = ZZ
o2 : ZZ-module, free
i3 : numgens target A
o3 = 3
i4 : entries A
o4 = {{9, 6}, {4, 3}, {2, 4}}
o4 : List
i5 : describe A
            3    2
o5 = map (ZZ , ZZ , {{9, 6}, {4, 3}, {2, 4}})
```

Come appena visto, strettamente collegate alle matrici sono le **mappe**, utilizzate per dichiarare applicazioni fra due strutture. La sintassi è:

```
<f>=map(<S>,<R>,<M>)
```

In tal modo viene definita una mappa da `R` in `S` che trasforma gli elementi come specificato nell'opportuna matrice `M` (anche sotto forma di lista). Applicazioni interessanti di tali mappe si vedranno in seguito quando si parlerà di anelli di polinomi.

Ecco un esempio di costruzione:

```
i1 : R=ZZ^3
       3
o1 = ZZ
o1 : ZZ-module, free
i2 : S=ZZ^4
       4
```

```
o2 = ZZ
o2 : ZZ-module, free
i3 : m=id_R||matrix{toList(numgens R:0)}
i4 : f=map(S,R,m)
o4 = | 1 0 0 |
     | 0 1 0 |
     | 0 0 1 |
     | 0 0 0 |
               4        3
o4 : Matrix ZZ  <--- ZZ
i5 : f(0_R)
o5 = | 0 |
     | 0 |
     | 0 |
     | 0 |
       4
o5 : ZZ
i6 : f(R_0)
o6 = | 1 |
     | 0 |
     | 0 |
     | 0 |
       4
o6 : ZZ
i7 : a=5*R_0+R_1-2*R_2
o7 = |  5 |
     |  1 |
     | -2 |
       3
o7 : ZZ
i8 : f(a)
```

```
o8 = |  5 |
     |  1 |
     | -2 |
     |  0 |
       4
o8 : ZZ
```

1.3 Strutture di controllo

Le strutture di controllo sono quelle tipiche dei linguaggi di programmazione comunemente diffusi. Tali strutture possono essere nidificate e contenute anche all'interno di funzioni.

La struttura **decisionale** permette di eseguire un'istruzione o in alternativa un'altra in base al verificarsi o meno di una condizione booleana. La sintassi è:

```
if <cond> then (<istr1>; ... <istrk>)
          else (<istr1>; ... <istrq>)
```

Ad esempio:

```
i1 : (-4..4)/(i -> if i < 0 then "neg"
                  else if i == 0 then "zer"
                       else "pos")
o1 = (neg, neg, neg, neg, zer, pos, pos, pos, pos)
o1 : Sequence
```

Le strutture di **ripetizione** permettono di ripetere un blocco di istruzioni. Esistono quella incondizionata (`for`) e quella condizionata (`while`) con controllo in testa. Oltre al classico utilizzo, quello di eseguire istruzioni tramite `do`, esiste nel *Macaulay2* una variante che consiste nel restituire una lista tramite `list` come se l'intera struttura fosse una funzione.
La sintassi, completa dei parametri sono opzionali, è la seguente:

```
(-) for <i> from <m> to <n> when <cond>
            list <funct>
            do (<istr1>; ... <istrk>)

(-) for <x> in <L> when <cond>
            list <funct>
            do (<istr1>; ... <istrk>)

(-) while <cond>
          list <funct>
          do (<istr1>; ... <istrk>)
```

Le due varianti del ciclo `for` funzionano, rispettivamente, per `i` che assume valori compresi fra `m` ed `n` (con step 1) e per `x` che assume tutti i valori, in ordine, presenti nella lista `L`. In ogni caso, il ciclo viene effettuato se la condizione `<cond>` è soddisfatta, nel momento in cui diventa falsa la ripetizione ha termine.
Il ciclo `while` controlla la condizione in testa e costruisce la lista come descritto da `<funct>` e/o esegue le istruzioni che seguono il `do`.

Alcuni esempi :

```
i1 : s=0; p=1;
i2 : for i from 1 to 20 when i<=4 list i^2 do (s=s+i; p=p*i)
o2 = {1, 4, 9, 16}
o2 : List
i3 : s
o3 = 10
i4 : p
o4 = 24
i5 : pot=(a,b)->(p=1; for i in 1..b do p=p*a; p)
o5 = pot
o5 : FunctionClosure
i6 : pot(2,4)
```

```
o6 = 16
i7 : fatt=n->(p=1; while n>1 do (p=p*n; n=n-1); return p)
o7 = fatt
o7 : FunctionClosure
i8 : fatt(4)
o8 = 24
i9 : fatt 0
o9 = 1
```

L'utilizzo delle strutture di ripetizione può a volte essere sostituito dalla ricorsione:

```
i1 : fatt=n->(p=1; if n>1 then p=n*fatt(n-1); p)
o1 : fatt
o1 = FunctionClosure
i2 : fatt 0
o2 = 1
i3 : fatt 5
o3 = 120
```

Si osservi che l'istruzione `apply` o l'operatore `/`, su di una lista, permettono di eseguire operazioni iterative come in presenza di una struttura di ripetizione. Gli ideatori di *Macaulay2* incentivano l'uso della manipolazione delle liste, ove possibile.

1.4 Esercizi

1. Dati due numeri naturali positivi a e i, è sempre possibile rappresentare in modo unico a come somma di coefficienti binomiali della seguente forma:

$$a = \binom{a_i}{i} + \binom{a_{i-1}}{i-1} + \cdots + \binom{a_j}{j}$$

con $a_i > a_{i-1} > \cdots > a_j \geq j \geq 1$. Tale scrittura è chiamata **espansione di Macaulay** [2, Lemma 6.3.4], dal matematico che la utilizzò per formulare il teorema omonimo riguardante le funzioni di Hilbert di un ideale.

Scrivere una funzione che dati due numeri naturali positivi ne determini l'espansione di Macaulay, costruendo ad esempio una lista di coppie che rappresentino i binomiali.

2. Scrivere una funzione che data una lista di coppie, ad esempio quella di un'espansione di Macaulay, effettui la somma dei coefficienti binomiali associati a ciascuna coppia.

2 Anelli dei polinomi

Il *Macaulay2* definisce e utilizza in maniera naturale gli anelli di polinomi su anelli o campi. Per definirli basta far seguire ad un anello/campo il vettore (finito) delle indeterminate `[a,b,...,z]`. È anche possibile creare anelli di polinomi utilizzando come anello base un anello di polinomi.
È buona norma assegnare un nome agli anelli creati. In caso siano stati definiti più anelli verrà data precedenza all'ultimo definito. È possibile forzare l'utilizzo di un anello `R` tramite l'istruzione `use R`. Si consiglia di fare attenzione alla scelta delle indeterminate, evitando conflitti con variabili definite in precedenza.

Una particolarità dei linguaggi di calcolo simbolico è che anelli definiti allo stesso modo, ma con istanze diverse saranno considerati differenti ed i loro elementi saranno non confrontabili. Per poter effettuare confronti bisognerà effettuare delle conversioni tramite isomorfismi.

Nei seguenti esempi, per definire anelli dei polinomi, utilizzeremo il campo dei razionali `QQ`:

```
i1 : R=QQ[x,y,z]
o1 = R
o1 : PolynomialRing
i2 : S=QQ[x,y,z]
o2 = S
```

```
o2 : PolynomialRing
i3 : R===S
o3 = false
i4 : clearAll
i5 : R=QQ[x_1..x_6]
o5 = R
o5 : PolynomialRing
i6 : x_1*x_4
o6 = x * x
      1   4
o6 : R
i7 : p=oo+x_1*x_5*x_6
o7 = x * x + x * x * x
      1   4   1   5   6
o7 : R
```

Per ottenere informazioni direttamente dall'anello è possibile utilizzare l'istruzione `vars R` che restituisce una matrice riga con le indeterminate utilizzate. In alternativa, l'istruzione `gens R` restituisce la lista delle indeterminate e `numgens R` il loro numero. Per ottenere l'indice associato ad un'indeterminata è possibile utilizzare l'istruzione `index <var>`.
Lo *zero* e l'*unità* dell'anello `R` possono essere utilizzati, rispettivamente, tramite `0_R` e `1_R`.
L'istruzione `basis(d,R)` restituisce la base, come `QQ`-spazio vettoriale, della componente di grado `d` dell'anello di polinomi `R`.

Per verificare l'**appartenza** di un polinomio `p` all'anello `R` si può utilizzare l'operatore *modulo* rispetto alle indeterminate dell'anello `p % vars R` o anche `p % 1_R`. Se il risultato di tale operazione è `0` allora è verificata l'appartenenza $p \in R$.

Anche nel caso di un anello, il comando `describe` fornisce informazioni complete (che è possibile reperire dinamicamente attraverso l'istruzione `options`):

```
i1 : R=QQ[x,y,z]
o1 = R
o1 : PolynomialRing
i2 : describe R
o2 = QQ[x..z, Degrees => {3:1}, Heft => {1},
     MonomialOrder => {MonomialSize => 32}, DegreeRank => 1]
                      {GRevLex => {3:1}  }
                      {Position => Up    }
```

Come si vede nell'esempio precedente, l'anello dei polinomi contiene sempre informazioni riguardanti la graduazione delle indeterminate, che di default è quella standard (grado 1), e riguardanti l'**ordinamento monomiale**, che di default è il lessicografico graduato inverso (GRevLex). L'ordinamento monomiale associato all'anello influenzerà non soltanto la scrittura dei polinomi ma anche molte operazioni che riguardano l'elaborazione di polinomi tramite **basi di Gröbner**.

Per associare un ordinamento monomiale ad un anello si può assegnare il parametro opzionale `MonomialOrder` tramite l'operatore `=>`:

```
R=QQ[x,y,z,MonomialOrder=>Lex]
```

Come valori per il `MonomialOrder` è possibile scegliere fra:

(-) `GRevLex`: ordinamento lessicografico graduato inverso

(-) `Lex`: ordinamento lessicografico

(-) `GLex`: ordinamento lessicografico graduato

(-) `RevLex`: ordinamento lessicografico inverso

(-) `Weights`: ordinamento personalizzato assegnando i pesi alle indeterminate

(-) `Eliminate`: ordinamento di eliminazione delle indeterminate

I **polinomi** possono essere elaborati tramite l'utilizzo delle operazioni standard già viste per gli anelli in generale. Inoltre è possibile manipolare i polinomi tramite alcune istruzioni riservate ad essi (alcune delle quali strettamente collegate all'ordinamento monomiale fissato):

(-) `terms p`: restituisce una lista contenente i termini del polinomio `p`

(-) `monomials p`: restituisce una matrice riga contenente i monomi di `p`

(-) `support p`: restituisce la lista delle indeterminate presenti in `p`

(-) `degree p`: restituisce il grado di `p`

(-) `degree(x,p)`: restituisce il grado dell'indeterminata `x` in `p`

(-) `exponents p`: restituisce la lista dei multigradi dei monomi di `p`

(-) `size p`: restituisce il numero dei monomi di `p`

(-) `homogenize(p,x)`: omogenizza `p` inserendo opportunamente l'indeterminata `x`

(-) `leadMonomial p` (o `leadTerm, leadCoefficient`): restituisce il monomio (o termine, coefficiente) iniziale di `p` in base all'ordinamento monomiale dell'anello di appartenenza

(-) `factor p`: restituisce la fattorizzazione del polinomio in irriducibili

(-) `roots p`: restituisce le radici del polinomio (in una variabile)

Ecco alcuni esempi:

```
i1 : R=QQ[x,y,z]
o1 = R
o1 : PolynomialRing
i2 : p=5*x*y+x*z^2-3*y*z
        2
o2 = x*z  + 5x*y - 3y*z
```

```
o2 : R
i3 : S=QQ[x,y,z,MonomialOrder=>Lex]
o3 = S
o3 : PolynomialRing
i4 : q=5*x*y+x*z^2-3*y*z
                2
o4 = 5x*y + x*z  - 3y*z
o4 : S
i5 : leadMonomial p
        2
o5 = x*z
o5 : R
i6 : leadMonomial q
o6 = x*y
o6 : S
i7 : f=map(S,R)
o7 = map(S,R,{x, y, z})
o7 : RingMap S <--- R
i8 : leadMonomial f(p)
o8 = x*y
o8 : S
```

Per ottenere una matrice casuale di polinomi si può utilizzare una variante del `random` utilizzato per le matrici, inserendo dei gradi negativi nel secondo parametro (che determinano il grado dei polinomi omogenei):

```
i1 : R=QQ[w..z]
o1 = R
o1 : PolynomialRing
i2 : random(R^3,R^{2:-1})
o2 = | 5/8w+x+9y+1/4z       2/7w+1/2x+1/8y+3/2z |
     | w+8/7x+2/5y+2/3z     3w+1/4x+3/5y+7/10z  |
```

```
      | 1/10w+7/2x+9/5y+1/2z 3/8w+9/5x+1/3y+z     |
             3       2
o2 : Matrix R  <--- R
```

Alcune importanti implicazioni degli ordinamenti monimiali verranno considerate dopo aver introdotto gli ideali e le basi di Gröebner.

2.1 Ideali

Per costruire un **ideale** di un anello di polinomi si può utilizzare l'istruzione `ideal L`, dove `L` è la lista (o equivalentemente una sequenza, matrice, etc...) dei generatori dell'ideale. Nel caso in cui la lista o la matrice contengano sottostrutture, esse verranno appiattite e tutti gli elementi saranno considerati generatori.
Per quozientare tramite un ideale basta utilizzare l'operatore `/`: `R/I`.

Ecco alcuni esempi:

```
i1 : R=QQ[w..z]
o1 = R
o1 : PolynomialRing
i2 : m=ideal vars R
o2 = ideal (w, x, y, z)
o2 : Ideal of R
i3 : I = ideal (w^2*y-x^2, y*z^4-z^3, x^5-w)
             2     2     4    3   5
o3 = ideal (w y - x , y*z  - z , x  - w)
o3 : Ideal of R
i4 : R/m
           R
o4 = ------------
      (w, x, y, z)
o4 : QuotientRing
i5 : S=R/I
```

```
o5 = S
o5 : QuotientRing
i6 : x^5-w
o6 = 0
o6 : S
i7 : x^5
o7 = w
o7 : S
```

Analogamente a quanto visto per gli anelli, per ottenere informazioni sugli ideali è possibile utilizzare l'istruzione `gens I` per ottenere la matrice dei generatori di `I` così come è stato costruito e `numgens I` per il loro numero. Per ottenere invece i generatori minimali si può utilizzare l'istruzione `mingens I`. L'istruzione `trim I` costruisce l'ideale coincidente con `I` tramite insieme minimale di generatori.
Anche in questo caso l'istruzione `basis(d,I)` restituisce la base, come `QQ`-spazio vettoriale, della componente di grado `d` dell'ideale `I`.

L'operatore `==` funziona analogamente a quanto già visto anche per l'uguaglianza fra ideali (come casi particolari `I==1_R` e `I==0_R`, tutto l'anello o l'ideale nullo) e la funzione booleana `isSubset(I,J)` verifica se l'ideale `I` sia **contenuto** nell'ideale `J`. Per verificare l'**appartenza** di un polinomio `p` all'ideale `I` si può ancora utilizzare l'operatore modulo rispetto a `I`: `p % I`. Se il risultato di tale operazione è `0` allora è verificata l'appartenenza $p \in I$.

Le operazioni fra ideali sono le seguenti:

(-) `+, *, ^`: operazioni estese agli ideali di somma, prodotto e potenza

(-) `I/J`: calcola l'ideale quoziente (l'ideale delle classi di equivalenza)

(-) `I:J` (o `quotient(I,J)`): calcola il colon o trasportatore fra i due ideali

(-) `intersect L`: calcola l'intersezione degli ideali nella lista `L`

(-) `radical I`: calcola il radicale di `I`

(-) `dim I`: calcola la dimensione di Krull di `I`

(-) `codim I`: calcola la codimensione (altezza) di `I`

(-) `isHomogeneous I`: verifica che `I` sia omogeneo (o graduato)

(-) `degrees I`: restituisce la lista dei gradi delle componenti graduate di `I`

(-) `minors(k,A)`: calcola l'ideale generato dai minori di ordine `k` della matrice `A`

Seguono alcuni esempi:

```
i1 : R=QQ[x_1..x_5]
o1 = R
o1 : PolynomialRing
i2 : I=ideal(x_1*x_4+x_5,x_2)
o2 = ideal (x x  + x , x )
             1 4    5   2
o2 : Ideal of R
i3 : J=ideal(x_2*x_3,x_1*x_4,x_5)
o3 = ideal (x x , x x , x )
             2 3   1 4   5
o3 : Ideal of R
i4 : intersect(I,J)
                                                          2
o4 = ideal (-x x , -x x , - x x  - x , -x x , - x x x  - x )
              2 5    2 5     1 4    5    2 3     1 4 5    5
o4 : Ideal of R
i5 : trim intersect(I,J)
o5 = ideal (x x , x x  + x , x x )
             2 5   1 4    5   2 3
o5 : Ideal of R
i6 : trim(I+J)
```

```
o6 = ideal (x , x , x x )
             5   2   1 4
o6 : Ideal of R
i7 : trim(I*J)
                              2           2     2 2    2
o7 = ideal (x x , x x x  + x , x x x , x x , x x  - x )
              2 5   1 4 5    5   1 2 4   2 3   1 4    5
o7 : Ideal of R
i8 : isSubset(I*J,intersect(I,J))
o8 = true
i9 : isSubset(intersect(I,J),I*J)
o9 = false
i10 : trim radical I
o10 = ideal (x , x x  + x )
              2   1 4    5
o10 : Ideal of R
i11 : trim radical (J*I)==(J*I)
o11 = false
```

2.2 Decomposizione primaria

Il *Macaulay2* permette tramite semplici istruzioni di calcolare una decomposizione primaria di un ideale. In questa sezione saranno presenti anche alcuni commenti sulle istruzioni relative a concetti collegati a quello della decomposizione.

Ecco alcune delle istruzioni collegate:

(-) `isPrime I`: funzione booleana che controlla se l'ideale `I` è primo

(-) `isPrimary I`: funzione booleana che controlla se l'ideale `I` è primario

(-) `primaryDecomposition I`: restituisce la lista degli ideali che formano una decomposizione primaria irridondante per l'ideale `I`

(-) `associatedPrimes I`: restituisce la lista degli ideali primi associati a `I`

(-) `minimalPrimes I`: restituisce la lista degli ideali primi minimali associati a `I`

(-) `primaryComponent(I,P)`: restituisce una componente primaria relativa all'ideale primo `P` associato a `I`

(-) `localize(I,P)`: localizza l'ideale `I` rispetto all'ideale primo `P`

Acuni esempi:

```
i1 : R=QQ[x_1..x_4]
o1 = R
o1 : PolynomialRing
i2 : I=ideal(x_1*x_4-x_2*x_3,x_3+x_4,x_2^2*x_3)
                                        2
o2 = ideal (- x x  + x x , x  + x , x x )
              2 3    1 4   3    4   2 3
o2 : Ideal of R
i3 : isPrime I
o3 = false
i4 : isPrimary I
o4 = false
i5 : primaryDecomposition I
                                 2
o5 = {ideal (x  + x , x  + x , x ), ideal (x , x )}
             3    4   1    2   2           4   3
o5 : List
i6 : associatedPrimes I
o6 = {ideal (x  + x , x , x ), ideal (x , x )}
             3    4   2   1           4   3
o6 : List
i7 : (primaryDecomposition I)/(x->radical x)
o7 = {ideal (x  + x , x , x ), monomialIdeal (x , x )}
```

```
                3    4   2   1                    3   4
o7 : List
```

Per verificare se un ideale è **massimale**, si può utilizzare un'istruzione contenuta nella libreria `QuillenSuslin` che restituisce l'ideale massimale che contiene l'ideale specificato:

```
i8 : loadPackage "QuillenSuslin"
o8 = QuillenSuslin
o8 : Package
i9 : getMaxIdeal I
o9 = ideal (x , x , x , x )
              4   3   2   1
o9 : Ideal of R
```

2.3 Ideali Monomiali

Nel *Macaulay2* gli **ideali monomiali** possono essere gestiti da una particolare classe *classe* che utilizza algoritmi ottimizzati. L'istruzione `monomialIdeal <L>` restituisce l'ideale monomiale associato ad una lista di monomi. Tramite l'istruzione `monomialSubideal I` si ottiene il più grande ideale monomiale contenuto nell'ideale `I`.

Nel caso in cui sia abbia a disposizione una lista (o una matrice) `L` di polinomi, fissato un ordinamento monomiale, l'istruzione `monomialIdeal L` restituisce l'ideale monomiale generato dai monomi iniziali di quelli della lista (o della matrice). Se invece `I` è un ideale non monomiale, allora `monomialIdeal I` restituisce l'ideale monomiale generato dai monomi iniziali dei generatori di una base di Gröbner di `I`, cioè l'**ideale iniziale** di `I`.

Ecco alcune istruzioni utili per gestire basi di Gröbner e ideali monomiali:

(-) `gb I`: calcola una base di Gröbner dell'ideale `I`, restituisce una struttura che memorizza anche dati realtivi ai calcoli effettuati e tramite alcune opzioni per-

mette di controllare gli algoritmi in modo da evitare errori o complessità troppo elevate

(-) `gens gb I`: restituisce la matrice dei generatori di una base di Gröbner dell'ideale

(-) `syz gb(I,Syzygies=>true)`: restituisce la matrice dei generatori del modulo delle sizigie dell'ideale

(-) `p % g`: se `p` è un polinomio e `g` una base di Gröbner, restituisce la forma normale del polinomio rispetto alla base

(-) `leadTerm I`: restituisce la matrice dei generatori dell'ideale iniziale di `I`, cioè i termini iniziali di una base di Gröbner di `I`

(-) `isMonomialIdeal I`: controlla se l'ideale `I` è monomiale

(-) `isSquareFree I`: controlla se l'ideale monomiale `I` è squarefree

Acuni esempi:

```
i1 : R=QQ[x,y]
o1 = R
o1 : PolynomialRing
i2 : I = ideal(x^3 - 2*x*y, x^2*y - 2*y^2 + x)
             3          2      2
o2 = ideal (x  - 2x*y, x y - 2y  + x)
o2 : Ideal of R
i3 : g=gb I
o3 = GroebnerBasis[status: done; S-pairs encountered up to degree 8]
o3 : GroebnerBasis
i4 : gens g
o4 = | 2y2-x xy x2 |
             1       3
o4 : Matrix R  <--- R
```

```
i5 : ideal leadTerm I
              2        2
o5 = ideal (2y , x*y, x )
o5 : Ideal of R
i6 : ideal leadTerm g
              2        2
o6 = ideal (2y , x*y, x )
o6 : Ideal of R
i7 : syz gb(I,Syzygies=>true)
o7 = {3} | -x2y+2y2-x |
     {3} | x3-2xy     |
              2       1
o7 : Matrix R  <--- R
```

Tramite le istruzioni viste finora è possibile verificare le soluzioni degli esercizi effettuati durante il corso. Si ricordi che per quanto riguarda i risultati finali, non vi è alcuna differenza fra l'utilizzo di funzioni per ideali generici e quelle per ideali monomiali: cambia il modo di eseguire i calcoli per via delle ottimizzazioni possibili nel secondo caso.
Nel seguito verranno analizzati alcuni algoritmi sugli ideali monomiali che potranno essere implementati per integrare/sostituire le funzioni finora esaminate.

2.4 Esercizi

1. Costruire una funzione che restituisca il multigrado del monomio iniziale di un polinomio (e quindi di un monomio nel caso particolare).

2. Costruire una funzione che restituisca il grado iniziale di un ideale.

3. Costruire una funzione booleana che controlli se un ideale è monomiale.

4. Dato un ideale monomiale I, costruire le funzioni per:

   (a) calcolare il radicale di I.

(b) verificare se un ideale è primo, irriducibile (o primario).

(c) ottenere la lista delle componenti primarie di una decomposizione (irridondante) di I.

(d) ottenere la lista $\mathrm{Ass}(I)$.

(e) ottenere la lista dei primi minimali associati ad I.

Riferimenti bibliografici

[1] D. R. Grayson and M. E. Stillman, "Macaulay2, a software system for research in algebraic geometry." Available at `http://www.math.uiuc.edu/Macaulay2/`.

[2] J. Herzog and T. Hibi, *Monomial ideals*, vol. 260 of *Graduate texts in mathematics*. Springer-Verlag London, 1 ed., 2011.