Connecting Singular, GAP, Polymake, and Gfan

Mohamed Barakat, Janko Boehm, Wolfram Decker, Yue Ren, Hans Schönemann

Technische Universität Kaiserslautern

6 June 2013

Integrating new features into Singular

Investigate mathematical questions combining

- Algebraic Geometry
- Group Theory
- Number Theory
- Convex Geometry
- Tropical Geometry

Integrating new features into Singular

Investigate mathematical questions combining

- Algebraic Geometry
- Group Theory
- Number Theory
- Convex Geometry
- Tropical Geometry

- Kernel-level interfaces from SINGULAR to POLYMAKE and GFAN.

Integrating new features into Singular

Investigate mathematical questions combining

- Algebraic Geometry
- Group Theory
- Number Theory
- Convex Geometry
- Tropical Geometry

- Kernel-level interfaces from SINGULAR to POLYMAKE and GFAN.

Tools for users and library developers:

- User defined data types in SINGULAR.
- Parallel computations in SINGULAR.

Integrating new features into Singular

Investigate mathematical questions combining

- Algebraic Geometry
- Group Theory
- Number Theory
- Convex Geometry
- Tropical Geometry

- Kernel-level interfaces from SINGULAR to POLYMAKE and GFAN.

Tools for users and library developers:

- User defined data types in SINGULAR.
- Parallel computations in SINGULAR.

For system developers:

- Integrating custom C++ code into SINGULAR.

Integrating new features into Singular

Investigate mathematical questions combining

- Algebraic Geometry
- Group Theory
- Number Theory
- Convex Geometry
- Tropical Geometry

- Kernel-level interfaces from SINGULAR to POLYMAKE and GFAN.

Tools for users and library developers:

- User defined data types in SINGULAR.
- Parallel computations in SINGULAR.

For system developers:

- Integrating custom C++ code into SINGULAR.

Application:

- Computing the GIT-fan.

Using Polymake in Singular

Example

```
> LIB "polymake.so";
   Welcome to polymake
   Copyright (c) 1997-2012
   Ewgenij Gawrilow, Michael Joswig (TU Darmstadt)
   http://www.polymake.org
   // ** loaded polymake.so
```

Using Polymake in Singular

Example

```
> LIB "polymake.so";
   Welcome to polymake
   Copyright (c) 1997-2012
   Ewgenij Gawrilow, Michael Joswig (TU Darmstadt)
   http://www.polymake.org
   // ** loaded polymake.so
> ring r = 0,(x,y,z),dp; poly g = x3+y3+1;
```

Using Polymake in Singular

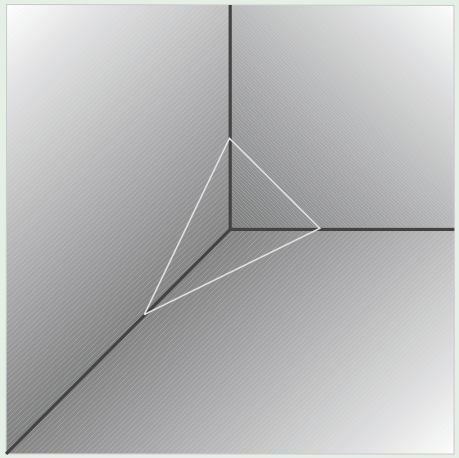
Example

```
> LIB "polymake.so";
   Welcome to polymake
   Copyright (c) 1997-2012
   Ewgenij Gawrilow, Michael Joswig (TU Darmstadt)
   http://www.polymake.org
   // ** loaded polymake.so
> ring r = 0,(x,y,z),dp; poly g = x3+y3+1;
> polytope p = newtonPolytope(g);
```

Using Polymake in Singular

Example

```
> LIB "polymake.so";
   Welcome to polymake
   Copyright (c) 1997-2012
   Ewgenij Gawrilow, Michael Joswig (TU Darmstadt)
   http://www.polymake.org
   // ** loaded polymake.so
> ring r = 0,(x,y,z),dp; poly g = x3+y3+1;
> polytope p = newtonPolytope(g);
> fan F = normalFan(p); F;
   RAYS:
   -1 -1 0 #0
   0 1 0   #1
   1 0 0   #2
   MAXIMAL_CONES:
   {0 1} #Dimension 3
   {0 2}
   {1 2}
```



Integrating custom C++ code

Example

```
// To make the function plus available in the interpreter add the following to
// extra.cc or to your own shared library:
```

Integrating custom C++ code

Example

```
// To make the function plus available in the interpreter add the following to
// extra.cc or to your own shared library:
BOOLEAN plus(leftv res, leftv args)
{
        int n = (int)(long)args->Data();
        int m = (int)(long)args->next->Data();
        res->rtyp = INT_CMD;
        res->data = (void*) n+m;
        return TRUE;
}
```

Integrating custom C++ code

Example

```
// To make the function plus available in the interpreter add the following to
// extra.cc or to your own shared library:
BOOLEAN plus(leftv res, leftv args)
{
        int n = (int)(long)args->Data();
        int m = (int)(long)args->next->Data();
        res->rtyp = INT_CMD;
        res->data = (void*) n+m;
        return TRUE;
}
void add_setup()
{
        iiAddCproc("","add",FALSE,add);
}
```

In this way GFAN and POLYMAKE types and functions were integrated into SINGULAR.

User defined data types in Singular

Using the **newstruct** framework the user can define new data types in the SINGULAR interpreter.

User defined data types in Singular

Using the **newstruct** framework the user can define new data types in the SINGULAR interpreter. As an example we construct $\mathbb{Q}$:

Example

```
> newstruct("QQ","int num, int den");
```

User defined data types in Singular

Using the **newstruct** framework the user can define new data types in the SINGULAR interpreter. As an example we construct $\mathbb{Q}$:

Example

```
> newstruct("QQ","int num, int den");
> proc frac(int n, int d){
      QQ t;
      t.num = n; t.den = d;
      return(t);}
```

User defined data types in Singular

Using the **newstruct** framework the user can define new data types in the SINGULAR interpreter. As an example we construct $\mathbb{Q}$:

Example

```
> newstruct("QQ","int num, int den");
> proc frac(int n, int d){
      QQ t;
      t.num = n; t.den = d;
      return(t);}
> proc printQQ(QQ r){
      string(r.num)+"/"+string(r.den);}
```

User defined data types in Singular

Using the **newstruct** framework the user can define new data types in the SINGULAR interpreter. As an example we construct $\mathbb{Q}$:

Example

```
> newstruct("QQ","int num, int den");
> proc frac(int n, int d){
      QQ t;
      t.num = n; t.den = d;
      return(t);}
> proc printQQ(QQ r){
      string(r.num)+"/"+string(r.den);}
> system("install","QQ","print",printQQ,1);
```

User defined data types in Singular

Using the **newstruct** framework the user can define new data types in the SINGULAR interpreter. As an example we construct $\mathbb{Q}$:

Example

```
> newstruct("QQ","int num, int den");
> proc frac(int n, int d){
      QQ t;
      t.num = n; t.den = d;
      return(t);}
> proc printQQ(QQ r){
      string(r.num)+"/"+string(r.den);}
> system("install","QQ","print",printQQ,1);
> proc addQQ(QQ r, QQ s){
      return(frac(r.num*s.den + s.num*r.den, r.den*s.den));}
```

User defined data types in Singular

Using the **newstruct** framework the user can define new data types in the SINGULAR interpreter. As an example we construct $\mathbb{Q}$:

Example

```
> newstruct("QQ","int num, int den");
> proc frac(int n, int d){
      QQ t;
      t.num = n; t.den = d;
      return(t);}
> proc printQQ(QQ r){
      string(r.num)+"/"+string(r.den);}
> system("install","QQ","print",printQQ,1);
> proc addQQ(QQ r, QQ s){
      return(frac(r.num*s.den + s.num*r.den, r.den*s.den));}
> system("install","QQ","+",addQQ,2);
```

User defined data types in Singular

Using the **newstruct** framework the user can define new data types in the SINGULAR interpreter. As an example we construct $\mathbb{Q}$:

Example

```
> newstruct("QQ","int num, int den");
> proc frac(int n, int d){
      QQ t;
      t.num = n; t.den = d;
      return(t);}
> proc printQQ(QQ r){
      string(r.num)+"/"+string(r.den);}
> system("install","QQ","print",printQQ,1);
> proc addQQ(QQ r, QQ s){
      return(frac(r.num*s.den + s.num*r.den, r.den*s.den));}
> system("install","QQ","+",addQQ,2);
> frac(2,3)+frac(3,5);
  19/15
```

Application: Computing the GIT-fan

$J \subset K[x_1, ..., x_n]$ homogeneous w.r.t. $Q = (q_{ij}) \in \mathbb{Q}^{r \times n}$ and $X = V(J)$.

Application: Computing the GIT-fan

$J \subset K[x_1, ..., x_n]$ homogeneous w.r.t. $Q = (q_{ij}) \in \mathbb{Q}^{r \times n}$ and $X = V(J)$.

Q defines a torus action $\quad \mathbb{T}^r \times X \to X$

GIT-fan describes the variation of good quotients $U /\!\!/ \mathbb{T}^r$ with $U \subseteq X$.

Application: Computing the GIT-fan

$J \subset K[x_1, ..., x_n]$ homogeneous w.r.t. $Q = (q_{ij}) \in \mathbb{Q}^{r \times n}$ and $X = V(J)$.

$$Q \text{ defines a torus action} \qquad \mathbb{T}^r \times X \to X$$

GIT-fan describes the variation of good quotients $U /\!\!/ \mathbb{T}^r$ with $U \subseteq X$.

Algorithm [Keicher, 2012]:

Face $\sigma \subseteq \gamma := \mathbb{R}^n_{\geq 0}$ is **J-face** if the $\mathbb{T}^n$-orbit of σ intersects X.

Application: Computing the GIT-fan

$J \subset K[x_1, \ldots, x_n]$ homogeneous w.r.t. $Q = (q_{ij}) \in \mathbb{Q}^{r \times n}$ and $X = V(J)$.

$$Q \text{ defines a torus action} \qquad \mathbb{T}^r \times X \to X$$

GIT-fan describes the variation of good quotients $U /\!\!/ \mathbb{T}^r$ with $U \subseteq X$.

Algorithm [Keicher, 2012]:

Face $\sigma \subseteq \gamma := \mathbb{R}^n_{\geq 0}$ is **J-face** if the $\mathbb{T}^n$-orbit of σ intersects X.

$$\Omega_J := \{Q(\sigma) \mid \sigma \text{ is } J\text{-face}\}$$

GIT-fan $\Lambda(J, Q)$ consists of all **GIT-chambers**

$$\lambda(w) = \bigcap_{\substack{\vartheta \in \Omega_J \\ w \in \vartheta}} \vartheta \qquad \text{for } w \in Q(\gamma).$$

Application: Computing the GIT-fan

We compute the GIT-fan of $\mathbb{G}(2,4)$:

Example

```
> LIB "gitfan.lib";
> ring R = 0,x(1..6),dp;
> ideal I = x(1)*x(6) - x(2)*x(5) + x(3)*x(4);
```

Application: Computing the GIT-fan

We compute the GIT-fan of $\mathbb{G}(2,4)$:

Example

```
> LIB "gitfan.lib";
> ring R = 0,x(1..6),dp;
> ideal I = x(1)*x(6) - x(2)*x(5) + x(3)*x(4);
> intmat Q[3][6] = 1,0,0,1,1,0,
                   0,1,0,1,0,1,
                   0,0,1,0,1,1,
```

Application: Computing the GIT-fan

We compute the GIT-fan of $\mathbb{G}(2,4)$:

Example

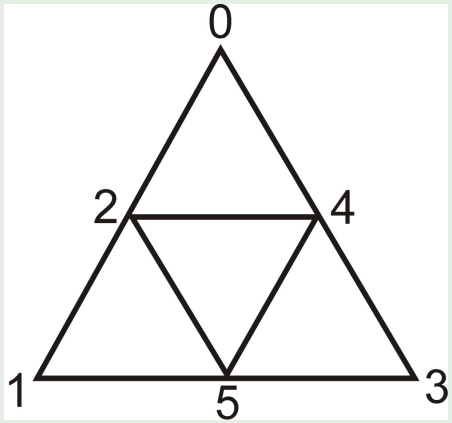
```
> LIB "gitfan.lib";
> ring R = 0,x(1..6),dp;
> ideal I = x(1)*x(6) - x(2)*x(5) + x(3)*x(4);
> intmat Q[3][6] = 1,0,0,1,1,0,
                   0,1,0,1,0,1,
                   0,0,1,0,1,1,
> fan F = gitFan(I, Q);
> rays(F);
   0 0 1
   0 1 0
   0 1 1
   1 0 0
   1 0 1
   1 1 0
```

Application: Computing the GIT-fan

We compute the GIT-fan of $\mathbb{G}(2,4)$:

Example

```
> LIB "gitfan.lib";
> ring R = 0,x(1..6),dp;
> ideal I = x(1)*x(6) - x(2)*x(5) + x(3)*x(4);
> intmat Q[3][6] = 1,0,0,1,1,0,
                   0,1,0,1,0,1,
                   0,0,1,0,1,1,
> fan F = gitFan(I, Q);
> rays(F);
   0 0 1
   0 1 0
   0 1 1
   1 0 0
   1 0 1
   1 1 0
```



Parallel computations in Singular

Example

```
> LIB("parallel.lib","random.lib");
> ring R = 0,x(1..4),dp;
> ideal I = randomid(maxideal(3),3,100);
```

Parallel computations in Singular

Example

```
> LIB("parallel.lib","random.lib");
> ring R = 0,x(1..4),dp;
> ideal I = randomid(maxideal(3),3,100);
> proc sizeStd(ideal I, string monord){
      def R = basering; list RL = ringlist(R);
      RL[3][1][1] = monord; def S = ring(RL); setring(S);
      return(size(std(imap(R,I))));}
```

Parallel computations in Singular

Example

```
> LIB("parallel.lib","random.lib");
> ring R = 0,x(1..4),dp;
> ideal I = randomid(maxideal(3),3,100);
> proc sizeStd(ideal I, string monord){
      def R = basering; list RL = ringlist(R);
      RL[3][1][1] = monord; def S = ring(RL); setring(S);
      return(size(std(imap(R,I))));}
> list commands = "sizeStd","sizeStd";
> list args = list(I,"lp"),list(I,"dp");
```

Parallel computations in Singular

Example

```
> LIB("parallel.lib","random.lib");
> ring R = 0,x(1..4),dp;
> ideal I = randomid(maxideal(3),3,100);
> proc sizeStd(ideal I, string monord){
      def R = basering; list RL = ringlist(R);
      RL[3][1][1] = monord; def S = ring(RL); setring(S);
      return(size(std(imap(R,I))));}
> list commands = "sizeStd","sizeStd";
> list args = list(I,"lp"),list(I,"dp");
> parallelWaitFirst(commands, args);
   [1] empty list
   [2] 11
```

Parallel computations in Singular

Example

```
> LIB("parallel.lib","random.lib");
> ring R = 0,x(1..4),dp;
> ideal I = randomid(maxideal(3),3,100);
> proc sizeStd(ideal I, string monord){
      def R = basering; list RL = ringlist(R);
      RL[3][1][1] = monord; def S = ring(RL); setring(S);
      return(size(std(imap(R,I))));}
> list commands = "sizeStd","sizeStd";
> list args = list(I,"lp"),list(I,"dp");
> parallelWaitFirst(commands, args);
   [1] empty list
   [2] 11
> parallelWaitAll(commands, args);
   [1] 55
   [2] 11
```