SECRETS OF...

... Singular and PolyBoRi
The Systems

- Singular: general commutative algebra
- PolyBoRi: commutative algebra in Boolean Rings
Singular - Terms

- A term in a polynomial is a struct containing:
  - coefficient
  - exponent vector
  - pointer to the next term

- A polynomial is identified with a pointer to its leading term
A monomial ordering can be represented by a matrix $A$

$$x^\alpha > x^\beta \iff A \cdot \alpha >_{lex} A \cdot \beta$$

these products are also stored in the term structure to speed up comparison of terms

The terms are ordered by monomial ordering

So the leading term is always the first term
Singular - polynomial structure

- highly manipulateable
- coef
- exp
- next pointer
- very compact in rings up to a medium number of variables
- very fast ordered iteration of polynomials
- arbitrary monomial ordering
- sparse
code example: cancel every multiple of „monom“

poly prev=c->S->m[i];
poly tail=c->S->m[i]->next;
while((tail!=NULL) && (pLmCmp(tail, monom)>=0))
{
    if (p_LmDivisibleBy(monom, tail, c->r))
    {
        prev->next=tail->next;
        tail->next=NULL;
        p_Delete(& tail, c->r);
        tail=prev;
    }
    prev=tail;
    tail=tail->next;
}
Consequences of this Style

- avoid a lot of copying/allocation
- very direct manipulation of polynomials possible
- „intuitive“ mainstream imperative style
- very fast polynomial arithmetic
- can introduce funny bugs and memory holes
- mutability of objects makes caching hard
Singular 3-1-0 - new rings

- Polynomial rings over
  - the integers
  - $\mathbb{Z}/m$
- Implemented for these rings:
  - arithmetic
  - Gröbner bases/normal forms
- Implemented by Oliver Wienand
Singular goes tropical...

*My curves come to a point!*
Tropical Geometry

* tropical.lib
* Anders Jensen
* Hannah Markwig
* Thomas Markwig
* tropical lifting (calling gfan)
* visualization
* j-invariants
* weierstrass form
* polymake.lib: Thomas Markwig
## Noncommutative News

<table>
<thead>
<tr>
<th>Library</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>discretize.lib</td>
<td>finite difference schemes</td>
</tr>
<tr>
<td>dmodapp.lib</td>
<td>applications d-modules</td>
</tr>
<tr>
<td>jacobson.lib</td>
<td>Smith/Jacobson Form</td>
</tr>
<tr>
<td>nchomolog.lib</td>
<td>noncommutative homological algebra</td>
</tr>
<tr>
<td>freegb.lib</td>
<td>two sided Gröbner bases in free algebras</td>
</tr>
</tbody>
</table>
Singular 3-0-1 further libraries

- redcgslib (Reduced Comprehensive Gröbner Systems)
- bfct.lib (Bernstein-Sato polynomial)
- decodegb.lib (Coding theory)
Secret pre-release version

http://www.mathematik.uni-kl.de/ftp/pub/Math/
Singular/devel/pre-3-1/
PolyBoRi

Model
- topology of digital circuits
- premade, optimized components
- variable orderings

Algorithm
- translation into Boolean polynomials
- one or several public/private key pairs for crypto
- suitable block orderings
- Buchberger specialized scripts
- F4
- Python
- C++

Data Structure
- Polynomial
- ZDD
- Matrices
Decision diagrams

* diagram decides if a term occurs in the polynomial
* term occurs if it exists as path leading to one
* Example: all Boolean terms of degree two

\[
a*b + a*c + a*d + a*e + b*c + b*d + b*e + c*d + c*e + d*e
\]
Monomial orderings

Given an monomial ordering $>$ and a polynomial in ZDD form, it is a priori unclear, how to

- calculate the leading term
- iterate over the terms (following the monomial ordering)

Implemented that for:

- lexicographical orderings (easy)
- Degree (reverse) lexicographical ordering
- Block orderings

For every ordering you have to find a special trick

- no general matrix orderings are supported
Lexicographical Ordering

- **Leading Term:**
  - always go right
- **ordered iteration:**
  - begin with lead
  - jump back and go left (repeatedly)

```
+ 1

+ x*y*z + x*y + x*z + x
+y*z + y
+z+1
+w
```
Degree Lexicographical Ordering

- Leading Term:
  - always go right, if exists max. degree term in this branch

- ordered iteration:
  - iterate lexicographical and jump over terms of "wrong" degree

\[ x*y*z + x*y + x*z + y*z + w + x + y + z + 1 \]
Further Orderings

- **Degree Reverse**
  Lexicographical
  \[(w < x < y < z)\]

- **Block Orderings**
  - **dlex blocks**
  - **degrevlex blocks**

- **Example:**
  - **derevlex:** \((w), (x, y, z)\)
Functional Style

- Every manipulation is forbidden/immutable objects

Pro

- Operations on polynomials can be cached on the level of diagram nodes

Contra

- Always creating new nodes can generate a lot of overhead (every node is guaranteed to be unique)
Example: canceling every multiple of monom

p = p.set()

p = p.diff(p.multiplesOf(monom))

p = Polynomial(p)
Recursive Implementation

template <class CacheType,
    class NaviType, class SetType>
SetType
dd_first_multiples_of(
    const CacheType& cache_mgr,
    NaviType navi, NaviType rhsNavi,
    SetType init){
    typedef typename SetType::dd_type dd_type;

    if(rhsNavi.isConstant())
        if(rhsNavi.terminalValue())
            return cache_mgr.generate(navi);
        else
            return cache_mgr.generate(rhsNavi);

    if(navi.isConstant() || (*navi > *rhsNavi))
        return cache_mgr.zero();

    if (*navi == *rhsNavi)
        return dd_first_multiples_of(
            cache_mgr, navi.thenBranch(),
            rhsNavi.thenBranch(), init).change(*navi);

    // Look up old result - if any
    NaviType result = cache_mgr.find(navi, rhsNavi);

    if (result.isValid())
        return cache_mgr.generate(result);

    // Compute new result
    init = dd_type(*navi,
        dd_first_multiples_of(
            cache_mgr, navi.thenBranch(),
            rhsNavi, init).diagram(),
        dd_first_multiples_of(cache_mgr,
            navi.elseBranch(),
            rhsNavi, init).diagram());

    // Insert new result in cache
    cache_mgr.insert(navi, rhsNavi, init.navigation());

    return init;
}
Solutions for overhead problem

- Replace many small operations by a few bigger ones
- Accept the overhead, understand the style decision diagram operations should be implemented and win in total by caching
- For a few operations, use alternative data structures (e.g. vectors of integers for Exponents of Boolean monomials)
The structure most different
from a ZDD is ...
... a dense matrix

- libm4ri
- Gregory Bard
- Martin Albrecht
- William Hart
- ...

* a good team:
* dense matrices for calculation with dense, random like systems
* ZDDs for structured/sparse polynomials