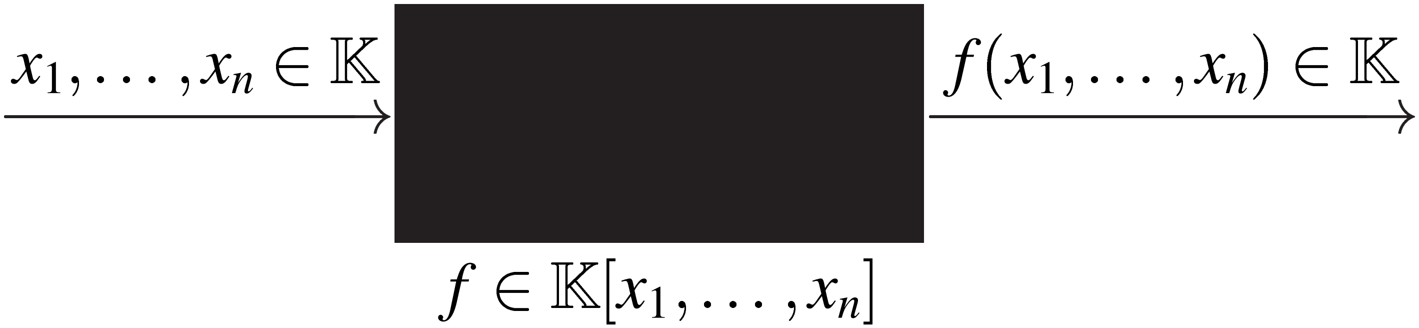
Generic Programming with Black Boxes

Erich Kaltofen
North Carolina State University
Raleigh, USA
kaltofen@math.ncsu.edu

Joint work with Angel Díaz
IBM T. J. Watson Research Center
Yorktown Heights, New York USA
aldiaz@us.ibm.com

Black Box Model for Multivariate Polynomials



$\mathbb{K}$ an arbitrary field, e.g., rationals, reals, complexes

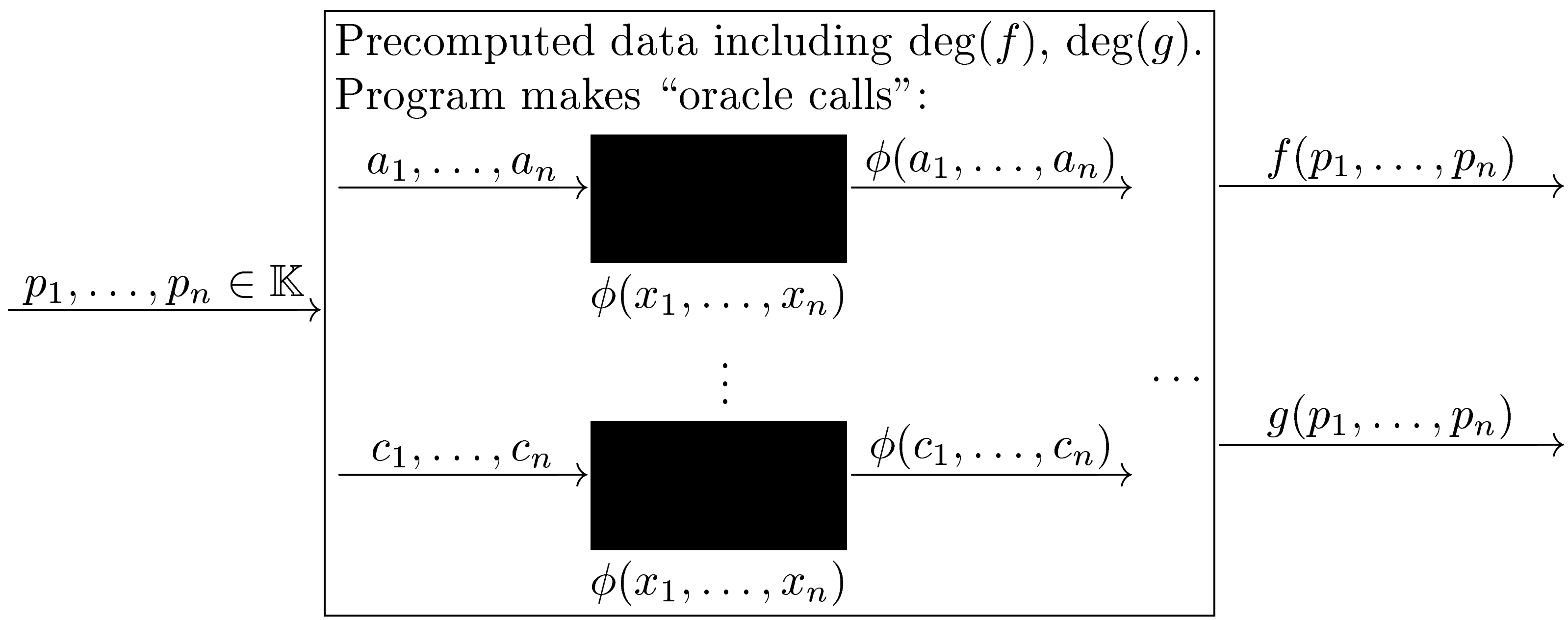
Perform polynmial algebra operations, e.g., factorization [K & Trager 90] with

$n^{O(1)}$ black box calls,
$n^{O(1)}$ arithmetic operations in $\mathbb{K}$ and
$n^{O(1)}$ randomly selected elements in $\mathbb{K}$

Cauchy matrix: factorization of numerator

$$\det\left(\begin{bmatrix} \frac{1}{x_1+y_1} & \frac{1}{x_1+y_2} & \cdots & \frac{1}{x_1+y_n} \\ \frac{1}{x_2+y_1} & \frac{1}{x_2+y_2} & \cdots & \frac{1}{x_2+y_n} \\ \vdots & \vdots & & \vdots \\ \frac{1}{x_n+y_1} & \frac{1}{x_n+y_2} & \cdots & \frac{1}{x_n+y_n} \end{bmatrix}\right) = \frac{\prod\limits_{1\le i<j\le n} (x_j - x_i)(y_j - y_i)}{\prod\limits_{1\le i,j\le n} (x_i + y_j)}.$$

K and Trager (1988) efficiently construct the following efficient program:



$$\phi(x_1, \ldots, x_n) = \frac{f(x_1,\ldots,x_n)}{g(x_1,\ldots,x_n)}, f, g \in \mathbb{K}[x_1, \ldots, x_n], \mathrm{GCD}(f, g) = 1.$$

Black Box Linear Algebra

The black box model of a matrix

$A \in \mathbb{K}^{n \times n}$ singular

$\mathbb{K}$ an arbitrary, e.g., finite field

Perform linear algebra operations, e.g., $A^{-1}b$ [Wiedemann 86] with

$$\begin{array}{rl} O(n) & \text{black box calls and} \\ n^2(\log n)^{O(1)} & \text{arithmetic operations in } \mathbb{K} \text{ and} \\ O(n) & \text{intermediate storage for field elements} \end{array}$$

RSA challenge systems over $\mathbb{Z}_2$

RSA-120: 245 881 rows and 252 222 columns with 10–217 non-zero entries/column and 11 037 745 non-zero entries total; the matrix occupies 48 Mbytes of memory space.

RSA-129: 569 466 rows and 524 339 columns with 13–23 214 non-zero entries/column (47 in the average) and 26 553 389 non-zero entries total; the matrix occupies 76 Mbytes of memory space.

RSA-130: 3 508 823 rows and 3 516 502 columns with and average of 39.4 entries/column and 138 690 745 non-zero entries total; the matrix occupies 700 Mbytes of memory space.

Timings for finding 32 row dependencies in RSA-129 matrix
Lobo's WiLiSS (block Wiedemann Linear System Solver) system

N	Task	Blocking Factor 1×32	2×32	3×32
569 466	Step W1		$167^h 39'$	
	Step W2		$106^h 45'$	
	Step W3		$54^h 40'$	
	total time		$332^h 04'$	
	work		$35630^{\#}$	

Each processor is rated at 107.3 MIPS. Combined work is measured in MIPS-hours$^{\#}$. Elapse time was approx. 384^h (16 days). Total work is 4.07 MIPS-years.

Timings for finding 18 row dependencies in RSA-130 matrix
(Montgomery's implementation of block Lanczos)

67.5 CPU-hours on a Cray-C90.

Classes of randomized algorithms

Monte Carlo	$\equiv$	always fast, probably correct
Las Vegas	$\equiv$	always correct, probably fast
BPP	$\equiv$	probably correct, probably fast

Why Las Vegas algorithms may be bad for you

```
repeat
        pick random numbers
        compute candidate answer
until check if a solution succeeds
```

A programming bug leads to an infinite loop!

The FoxBox System [Díaz & Kaltofen 1998

– Base arithmetics

– Black box objects

– Common black box objects

– Black box algorithms

– Extended domain black box objects

– Homomorphic maps

– Parallel black boxes

Parallel black boxes

Once constructed, a black box algorithm utilizes a small amount of precomputed static information for evaluation.

– The parallel black box interface adds three member functions for administering remote evaluation

– An initialization phase transmits static information to each processor allowing for subsequent probes

— FoxBox provides an MPI compliant implementation of the parallel black boxes interface

Running example: determinant of symmetric Toeplitz matrix

$$\det\left(\begin{bmatrix} a_0 & a_1 & \dots & a_{n-2} & a_{n-1} \\ a_1 & a_0 & \dots & a_{n-3} & a_{n-2} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{n-2} & a_{n-3} & \dots & a_0 & a_1 \\ a_{n-1} & a_{n-2} & \dots & a_1 & a_0 \end{bmatrix}\right)$$

$$= F_1(a_0, \dots, a_{n-1}) \cdot F_2(a_0, \dots, a_{n-1}).$$

over the integers.

```
> readlib(showtime):
> showtime():
O1 := T := linalg[toeplitz]([a,b,c,d,e,f]):
time   0.03   words   7701
O2 := factor(linalg[det](T));
```

$$
\begin{aligned}
&-(2dca-2bce+2c^2a-a^3-da^2+2d^2c+d^2a+b^3+2abc-2c^2b\\
&\quad+d^3+2ab^2-2dcb-2cb^2-2ec^2+2eb^2+2fcb+2bae\\
&\quad+b^2f+c^2f+be^2-ba^2-fdb-fda-fa^2-fba+e^2a-2db^2\\
&\quad+dc^2-2deb-2dec-dba)(2dca-2bce-2c^2a+a^3\\
&\quad-da^2-2d^2c-d^2a+b^3+2abc-2c^2b+d^3-2ab^2+2dcb\\
&\quad+2cb^2+2ec^2-2eb^2-2fcb+2bae+b^2f+c^2f+be^2-ba^2\\
&\quad-fdb+fda-fa^2+fba-e^2a-2db^2+dc^2+2deb-2dec\\
&\quad+dba)
\end{aligned}
$$

```
time   27.30   words   857700
```

Example: a FoxBox common object and factor box

```
// initialize our SACLIB and NTL wrapper/adaptors
Word Stack; int i;
SaclibInitEnv( 1000000, Stack ); ShoupInitEnv( &MPCard );
```

```
// construct a symmetric Toeplitz determinant
// from which we create a factor box
typedef BlackBoxSymToeDet< SaclibQ, SaclibQX >  BBSymToeDetQ;
typedef BlackBoxFactors< SaclibQ, SaclibQX,
                         BBSymToeDetQ > BBFactorsQ;

BBSymToeDetQ SymToeDetQ( N, DegDet );
BBFactorsQ   FactorsQ( SymToeDetQ, Prob, Seed, &MPCard );
```

Example: a FoxBox homomorphic image

```
// map the factors black box to NTL's modular arithmetic
typedef BlackBoxSymToeDet< ShoupZP, ShoupZPX > BBSymToeDetZP;

typedef BlackBoxFactorsHMap< SaclibQ, SaclibQX,
                             ShoupZP, ShoupZPX,
                             BBSymToeDetQ, BBSymToeDetZP,
                             SaclibQShoupZP > BBFactorsQMapZP;


SaclibQShoupZP   h;
BBSymToeDetZP    SymToeDetZP( N, DegDet );
BBFactorsQMapZP  FactorsZP( SymToeDetZP, FactorsQ, h );

SaclibCleanUpEnv(); // we no longer need rational arithmetic
```

Example: FoxBox sparse interpolation

```
// interpolate the first factor
typedef BlackBoxSelector< ShoupZP, BBFactorsQMapZP > BBFactorZP;

BBFactorZP FirstFactorZP( FactorsZP, 0 );

SparseInterp( FirstFactorZP, Vars, Degs,
              DegDet, &MPCard,
              AnsDegs, AnsMons,
              ShoupZPXElm,
              ProbName, IsRestart, NumProc  );

ShoupCleanUpEnv();
```

Factorization challenge: construction

N	CPU Time	N	CPU Time
11	$0^h02'$	16	$0^h43'$
12	$0^h05'$	17	$1^h05'$
13	$0^h09'$	18	$1^h42'$
14	$0^h16'$	19	$2^h30'$
15	$0^h26'$	20	$3^h42'$

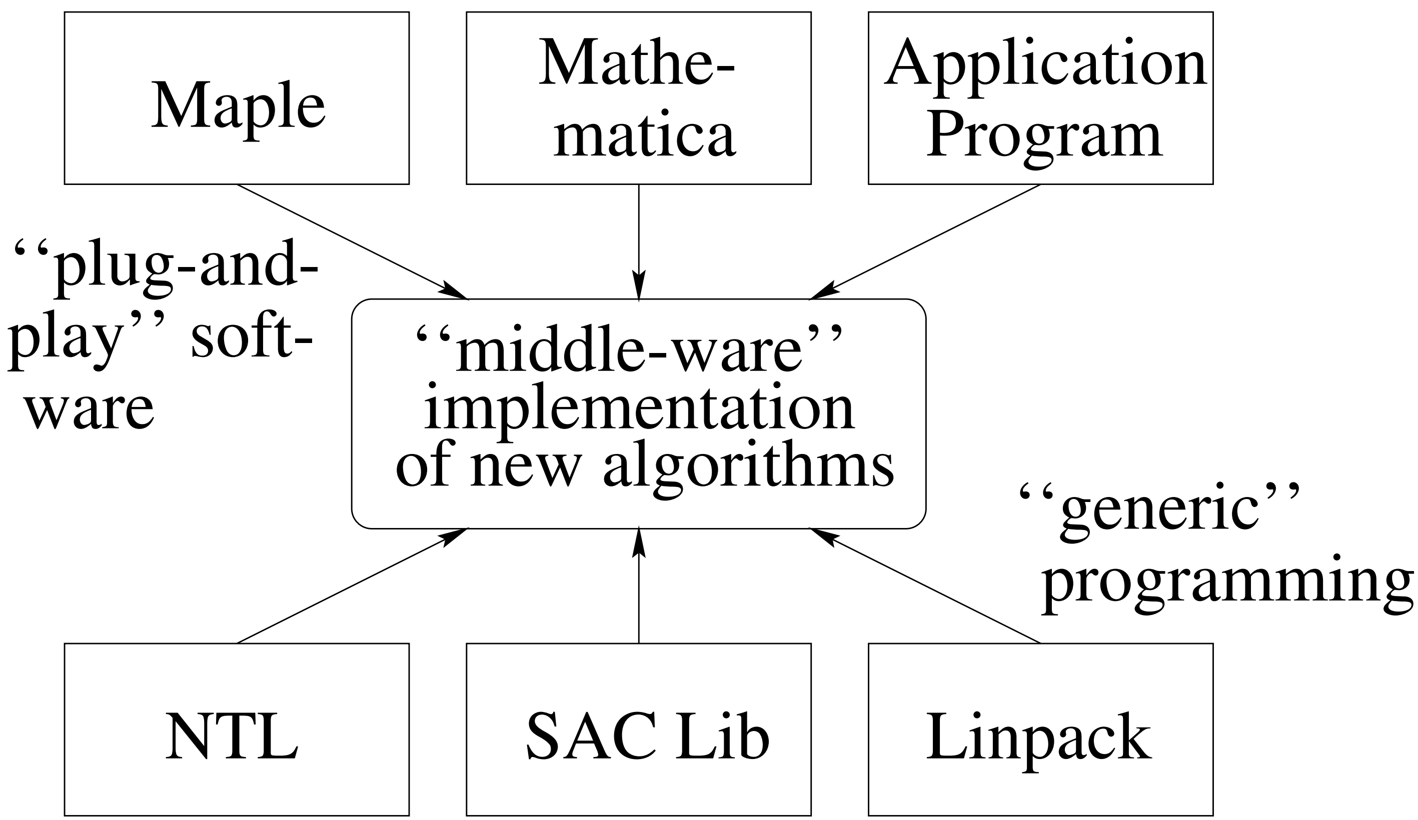
Total CPU times (hours^{h}minutes$'$) required to construct a factors black box (over $\mathbb{Q}$) that can evaluate both irreducible factors of the determinant of a symmetric Toeplitz matrix. The processor is a Sun Ultra 1/170 (128MB), Solaris 2.5.

Factorization challenge: sparse conversion

N	CPU Time	Degree	# Terms
10	$1^h 20'$	5	931
11	$1^h 34'$	5	847
12	$10^h 14'$	6	5577
13	$15^h 24'$	6	4982

CPU times (hours^{h}minutes$'$) to retrieve the distributed representation of a factor from the factors black box of a symmetric Toeplitz determinant black box. Construction is over $\mathbb{Q}$ evaluation is in $\mathrm{GF}(10^8+7)$ for $N = 10, 11$, and 12 (Pentium 133, Linux 2.0) and $\mathrm{GF}(2^{30}-35)$ for $N = 13$ (Sun Ultra 2 168MHz, Solaris 2.4).

Plug-And-Play Components



Problem solving environ's: end-user can easily custom-make symbolic software

Example: Maple black box factorization

```
> SymToeQ  := BlackBoxSymToe( BBNET_Q, 4, -1, 1.0 ):

> SymToeZP := BlackBoxSymToe( BBNET_ZP, 4, -1, 1.0 ):

> FactorsQ := BlackBoxFactors( BBNET_Q, SymToeQ, Mod, 1.0,
                              Seed ):

> FactorsZP := BlackBoxHomomorphicMap( BBNET_FACS, FactorsQ,
                                      SymToeZP ):

> FactorZP  := BlackBoxSelectValue( BBNET_ZP, FactorsZP, 0 ):

> FB1 := SparseConversion( BBNET_ZP, FactorZP,
               [ x1, x2, x3, x4 ], [ 4, 4, 4, 4 ], 4, Mod );
```

Software Design Issues

Plug-and-play

- Standard representation for transfer: MP, OpenMath, MathML
- Byte code for constructing objects vs. parse trees
- Visual programming environments for composition

Generic Programming

- Common object interface (wrapper classes),
  e.g., `K::random_generator(500)`
- Storage management vs. garbage collection
- Algorithmic shortcuts into the basic modules