Irreducible fractions, patterns and straightness
DGtal, Arithmetic package (since 0.5)

Jacques-Olivier Lachaud

DGtal Meeting, june 2012

UMR 5127

Arithmetic package content

Content (New package in DGtal 0.5)

- elementary integer arithmetic algorithms (gcd, Bézout)
- several representations for irreducible fractions
  - Stern-Brocot tree
  - continued fractions
  - rational approximations
- patterns
- digital straight lines and subsegments

Location

- `{DGtal}/src/DGtal/arithmetic`
- `{DGtal}/tests/arithmetic`
- `{DGtal}/examples/arithmetic`

Elementary arithmetic over arbitrary integer types

Class `IntegerComputer<Int>`

- stores temporary variables (useful for `BigInteger`)
- elementary operations : max, min, abs, isPositive, ...
- provides classical arithmetic computations : gcd, extended Euclid, convergents, continued fraction

```
typedef DGtal::BigInteger Integer;
IntegerComputer<Integer> ic; // instance for computations
Integer g = ic.gcd( 192, 128 ); // 64
IntegerComputer<Integer>::Vector2I v
  = ic.extendedEuclid( 5, 12, 2 ); // solution to 5x+12y = 2
std::cout << "5x+12y=2 <=> x=" << v[0]
  << " y=" << v[1] << std::endl; // 5x+12y=2 <=> x=10 y=-4
std::vector<Integer> q; // quotients
ic.getCFrac( q, 5, 13 ); // continued fraction
std::cout << "5/13=[" << q[0] << ";" << q[1]
  << "," << q[2] << "," << q[3]
  << "," << q[4] << "]" << std::endl; // 5/13=[0;2,1,1,2]
```

+ more complex operations related to integer half spaces

Elementary arithmetic over arbitrary integer types (II)

```
ic.getCFrac( q,
             Integer("51234567894643563456345635435722900123"),
        Integer("3456785320876092394577594289 01234" ) );
std::cout << "[" << q[0];
for ( unsigned int i = 1; i < q.size(); ++i )
  std::cout << "," << q[i];
std::cout << "]" << std::endl;
// [148214,2,29,3,3,1,1,1,8,2,5,1,4,3,4,1,1,2,1,1,2,1,1,2,1,5,1,1,
//  4,2,2,1,1,1,4,3,2,1,1,2,3,1,2,3,1,14,1,3,13,7,1,1,1,1,1,9,1,1,
//  27,3,1,1,3,8,1,1,1,8,6,1,1,2,6,3,1,1,4,4]
```

Properties of positive irreducible fractions

Definition positive irreducible fraction

A fraction $\frac{p}{q}$ with $p, q \in \mathbb{Z}^+, \text{gcd}(p, q) = 1$.

- uniqueness, dense
- related to finite simple continued fractions (Euclid algorithm)
- generated by the Stern-Brocot tree

Simple continued fractions

Definition simple continued fraction

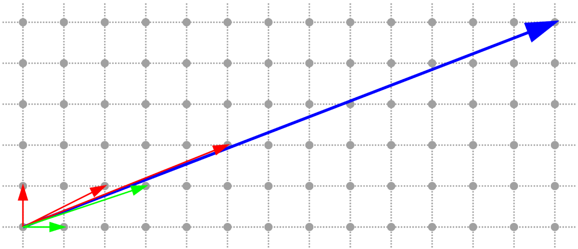
A number of the form $a_0 + \cfrac{1}{a_1 + \cfrac{1}{\dots + \cfrac{1}{a_n}}}$, where a_i are integers, commonly written as $[a_0; a_1, \dots, a_n]$. The a_i are the partial quotients.

- Any simple continued fraction is a positive irreducible fraction
- Any positive irreducible fraction has two simple continued fraction representations
- use Euclid algorithm (gcd, quotients), e.g. $\frac{5}{13}$.

p	=	u	$*$	q	+	r
5	=	0	$*$	13	+	5
13	=	2	$*$	5	+	3
5	=	1	$*$	3	+	2
3	=	1	$*$	2	+	1
2	=	2	$*$	1	+	0

$$\frac{5}{13} = 0 + \cfrac{1}{2 + \cfrac{1}{1 + \cfrac{1}{1 + \cfrac{1}{2}}}}$$

Convergents and approximation



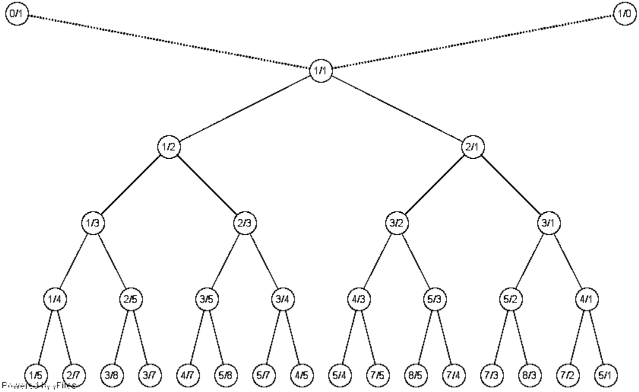
$z_4 = \frac{5}{13} = [0; 2, 1, 1, 2]$

odd convergents : $z_3 = \frac{2}{5} = [0; 2, 1, 1]$ $z_1 = \frac{1}{2} = [0; 2]$ $z_{-1} = \frac{1}{0} = []$

even convergents : $z_2 = \frac{1}{3} = [0; 2, 1]$ $z_0 = \frac{0}{1} = [0]$

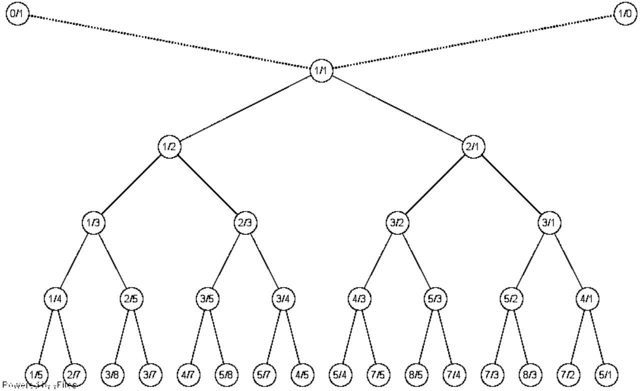
- convergents are the best approximations to fractions/real numbers
- thus related to digital straight lines

Stern-Brocot tree of irreducible fractions



- two starting fractions : $\frac{0}{1}$ and $\frac{1}{0}$
- mediant of two fractions : $\frac{p}{q} \oplus \frac{p'}{q'} = \frac{p+p'}{q+q'}$ (vector addition)

Link with continued fractions



- $u_0, u_1, \ldots, u_k$ = sequence of Right-then-Left moves from $\frac{1}{1}$, except last (one less).
- e.g. $\frac{5}{13} = [0; 2, 1, 1, 2]$, thus $R^0 L^2 R L R^{2-1}$.

Useful operations on fractions

- if we forget $+$, $-$, $*$, $/$ ..., interesting operations are related to the "tree" structure
- making a fraction from its quotients, getting quotients
- mediant, left or right descendant, adding a quotient
- father, previous partial, *m*-father,
- arbitrary convergent / reduced partial

Useful operations on fractions

- if we forget $+$, $-$, $*$, $/$ ..., interesting operations are related to the "tree" structure
- making a fraction from its quotients, getting quotients
- mediant, left or right descendant, adding a quotient
- father, previous partial, *m*-father,
- arbitrary convergent / reduced partial

Requirements

- Perform these operations in quasi-constant time !
- But storing quotients cost $O(\log(\max(p, q)))$

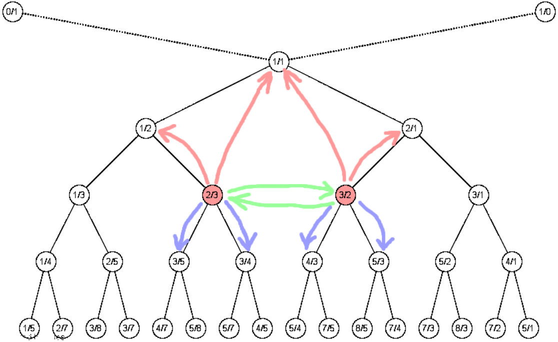
Useful operations on fractions

- if we forget $+$, $-$, $*$, / ..., interesting operations are related to the "tree" structure
- making a fraction from its quotients, getting quotients
- mediant, left or right descendant, adding a quotient
- father, previous partial, *m*-father,
- arbitrary convergent / reduced partial

Solution

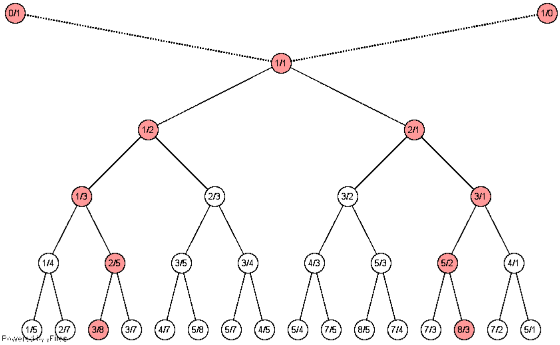
- irreducible fraction described by concept `CPositiveIrreducibleFraction`
- explicit representation of the Stern-Brocot tree
- each node stores k, u_k, p_k, q_k
- but on-the-fly instanciation of nodes.

Models of irreducible fractions (I)



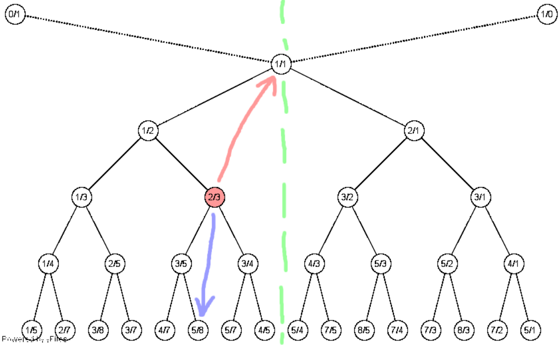
- Class SternBrocot, fraction is SternBrocot::Fraction
- Each node knows 5 other nodes (fathers, reciprocal, direct descendants on demand)
- Simple, fast for small fractions, memory costly, operations in $O(u_k)$

Models of irreducible fractions (I)



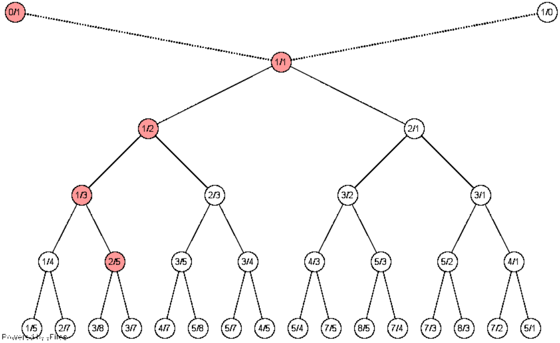
- Class `SternBrocot`, fraction is `SternBrocot::Fraction`
- Each node knows 5 other nodes (fathers, reciprocal, direct descendants on demand)
- Simple, fast for small fractions, memory costly, operations in $O(u_k)$

Models of irreducible fractions (II)



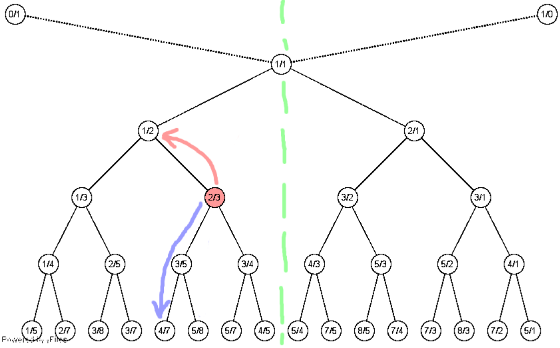
- Class `LightSternBrocot`, fraction is `LightSternBrocot::Fraction`
- Each node knows its reduced, mapping to next partials on demand
- fast for small fractions, less memory costly, but tricky cases

Models of irreducible fractions (II)



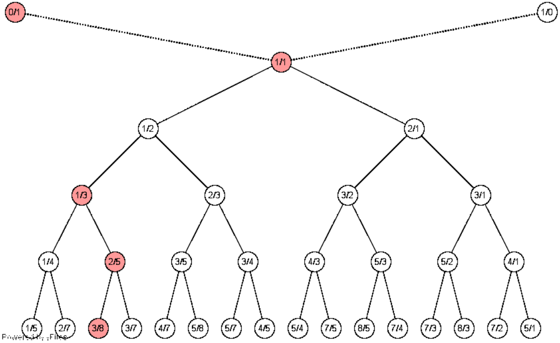
- Class `LightSternBrocot`, fraction is `LightSternBrocot::Fraction`
- Each node knows its reduced, mapping to next partials on demand
- fast for small fractions, less memory costly, but tricky cases

Models of irreducible fractions (III)



- Class `LighterSternBrocot`, fraction is `LighterSternBrocot::Fraction`
- Each node knows its origin, mapping to next partials on demand
- fast for big fractions, less memory costly, best trade-off

Models of irreducible fractions (III)



- Class `LighterSternBrocot`, fraction is `LighterSternBrocot::Fraction`
- Each node knows its origin, mapping to next partials on demand
- fast for big fractions, less memory costly, best trade-off

Using fractions

Choosing your type of fraction...

```
// quotients are int64_t, numerators are BigInteger.
typedef LighterSternBrocot<BigInteger,int64_t> SB;
typedef SB::Fraction Fraction;
```

Using fractions

Elementary methods : `z` is a fraction

Name	Expression	Semantics
Constructor	`Fraction( p, q )`	creates the fraction p'/q', where $p' = p/g$, $q' = q/g$, $g = \gcd(p, q)$
numerator	`z.p()`	returns the numerator
denominator	`z.q()`	returns the denominator
quotient	`z.u()`	returns the quotient u_k
depth	`z.k()`	returns the depth k
null test	`z.null()`	returns 'true' if the fraction is null 0/0
even parity	`z.even()`	returns 'true' iff k is even
odd parity	`z.odd()`	returns 'true' iff k is odd

Using fractions

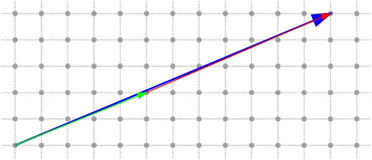
Creating fractions and getting convergents...

```
Fraction z( 643, 432 ); // classical instanciation
SB::display( std::cout, z ); // z=z_3=[1,2,21,10]
std::cout << std::endl;
std::cout << "Nb nodes = " << SB::instance().nbFractions
  << std::endl; // 6 nodes
Fraction z2 = z.previousPartial(); // z_{n-1}
SB::display( std::cout, z2 ); // z_2=[1,2,21]
std::cout << std::endl;
Fraction z1 = z.reduced( 2 ); // z_{n-2}
SB::display( std::cout, z1 );  // z_1=[1,2]
std::cout << std::endl;
z.pushBack( make_pair( 12, 4 ) ); // deeper fraction
SB::display( std::cout, z ); // z=z_4=[1,2,21,10,12]
// [Fraction f=7780/5227 u=12 k=4 [1,2,21,10,12] ]
std::cout << std::endl;
// Fraction is a Back Insert Sequence
back_insert_iterator<Fraction> outIt = back_inserter( z );
*outIt++ = make_pair( 1, 5 ); // u_5 = 1
*outIt++ = make_pair( 3, 6 ); // u_6 = 3
SB::display( std::cout, z );
// [Fraction f=33049/22204 u=3 k=6 [1,2,21,10,12,1,3] ]
std::cout << std::endl;
```

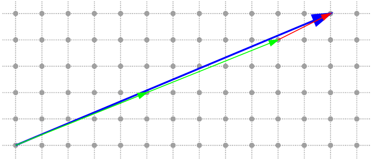
Using fractions

Other useful methods...

Name	Expression	Semantics
splitting formula	`z.getSplit(z1, z2)`	$z_1 \oplus z_2 = z$
Berstel splitting	`z.getSplitBerstel(x1, n1, x2, n2)`	$(z_1)^{n_1} \oplus (z_2)^{n_2} = z$



split $5/12 = 2/5 \oplus 3/7$



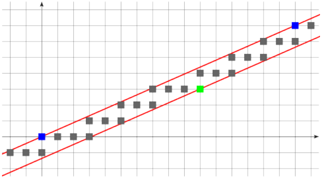
Berstel $5/12 = 2/5 \oplus 2/5 \oplus 1/2$

- obvious link with Bézout points, leaning points of straight lines.

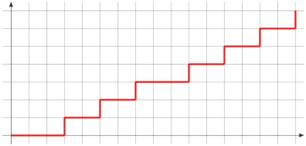
Digital straight segments as Patterns

Définition (Pattern)

Freeman chain code between two consecutive upper leaning points of a digital straight line



DSL(7, 16, 0)



00010010010001001001001

= Christoffel words [[Christoffel, 1875]]

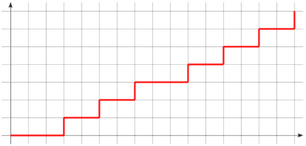
Digital straight segments as Patterns

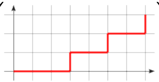
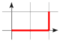
Recursive formula [Berstel, 96] (also splitting formula [Bruckstein ...])

$$\frac{7}{16} = [0, 2, 3, 2]$$

$$E([0, 2, 3, 2]) = E([0, 2, 3])^2 \quad E([0, 2])$$

$$0001001001000100100 1001 = (0001001001)^2 \quad 001$$



$=$ ()^2 

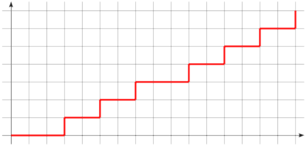
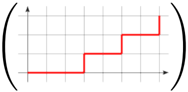
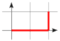
Digital straight segments as Patterns

Recursive formula [Berstel, 96] (also splitting formula [Bruckstein ...])

$$\frac{7}{16} = [0, 2, 3, 2]$$

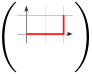
$$E([0,2,3,2]) = E([0,2,3])^2 \quad E([0,2])$$

$$0001001001000100100101 = (0001001001)^2 \quad 001$$

 =  ^{2} 

$$E([0,2,3]) = E([0]) \quad E([0,2])^3$$

$$0001001001 = 0 \quad (001)^3$$

 =   ^{3}

Patterns in DGtal

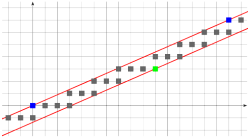
Class Pattern<Fraction>

```
  ...
typedef LighterSternBrocot<int32_t,int32_t> SB; // Stern-Brocot tree
typedef SB::Fraction Fraction; // the type for fractions
typedef Pattern<Fraction> MyPattern; // the type for patterns

DGtal::int32_t p = atoi( argv[ 1 ] );
DGtal::int32_t q = atoi( argv[ 2 ] );
MyPattern pattern( p, q );

bool sub = ( argc > 3 ) && ( std::string( argv[ 3 ] ) == "SUB" );
cout << ( ! sub ? pattern.rE() : pattern.rEs( "(|)" ) ) << endl;
```

```
bash> ./examples/arithmetic/pattern 11 17
0010010100101001010010100101
bash> ./examples/arithmetic/pattern 11 17 SUB
((00|1)|(0|0101)(0|0101)(0|0101)(0|0101)(0|0101))
```

+ positions of leaning points
+ greatest included subpattern given some $[AB]$
+ smallest covering subpattern given some $[AB]$

Digital straight lines

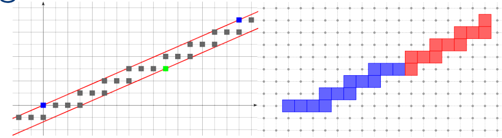


Class StandardDSLQ0<Fraction> , characteristics (a, b, μ)

```
#include "DGtal/arithmetic/StandardDSLQ0.h"
...
typedef ... Fraction;
typedef StandardDSLQ0<Fraction> DSL;
...
DSL D( 7, 16, 0 ); // (a, b, mu)
```

- get characteristics : `a()`, `b()`, `mu()`, `mup()`
- get slope `slope()` and pattern `pattern()`
- get first upper leaning point in quadrant `U()`, next lower `L()`
- get points from given abscissa or ordinate `lowestY( x )`, ...

Digital straight lines can be enumerated



```
typedef StandardDSLQ0<Fraction> DSL;
typedef DSL::ConstIterator ConstIterator;
DSL D( 7, 16, 0 ); // (a, b, mu)
board << CustomStyle( plow.className(), // in blue
                      new CustomColors( Color(0,0,255),
                                        Color(100,100,255) ) );
// segment [UL[
for ( ConstIterator it = D.begin( D.U() ),
        itend = D.end( D.L() ); it != itend; ++it )
  board << *it;
board << CustomStyle( plow.className(), // in red
                      new CustomColors( Color(255,0,0),
                                        Color(255,100,100) ) );
// segment [LU'[
for ( ConstIterator it = D.begin( D.L() ),
        itend = D.end( D.U() + D.v() ); it != itend; ++it )
  board << *it;
```

- A DSL is also a model of Class CPointPredicate

Fast extraction of subsegments

Knowing a DSL D, what are the characteristics of a subsegment $[A, B]$?

- standard recognition of the segment $[A, B]$ e.g. [Debled, Reveilles 1995]
  $\Rightarrow$ linear in its length
- `D.smartDSS(...)` recognition by going top-down the Stern-Brocot tree. [Said, L. 2009]
  $\Rightarrow$ linear in the sum of the quotients of output slope
- `D.reversedSmartDSS(...)` recognition by going bottom-up the Stern-Brocot tree. [Said, L. 2010]
  $\Rightarrow$ linear in the depth of output slope

	Speed-up factor wrt **ArithmeticDSS**			
	SmartDSS		**ReversedSmartDSS**	
N	$M = N/10$	$M = N/2$	$M = N/10$	$M = N/2$
30	1,2	1,5	1,1	1,4
400	2,3	6,8	2,2	6,8
1600	6,7	26,9	6,3	27,7
25600	70,9	378,3	75,5	441,9
409600	2195,0	22274,8	2574,1	27239,4