mandelbrot-symbolics

Claude Heiland-Allen

2015–2019

Contents

1 c/bin/Makefile

```
 prefix ?= $(HOME)/opt
CC ?= gcc

PKGCONFIG := PKG_CONFIG_PATH="$(prefix)/lib/pkgconfig" pkg-config
COMPILE := $(CC) -std=c99 -Wall -Wextra -pedantic -fPIC -O3 -pipe -MMD `$(PKGCONFIG) --cflags mandelbrot-symbolics`
LIBS    := `$(PKGCONFIG) --libs mandelbrot-symbolics` -lgmp
OBJECTS := $(patsubst %.c,%.o,$(wildcard *.c))
DEPENDS := $(patsubst %.o,%.d,$(OBJECTS))
EXES    := $(patsubst %.o,%,$(OBJECTS))

all: $(EXES)

clean:
	@echo "CLEAN" ; rm -f $(OBJECTS) $(DEPENDS) $(EXES)

install: $(EXES)
	install -d "$(prefix)/bin"
	install -m 755 -t "$(prefix)/bin" $(EXES)

%: %.o
	@echo "EXE      $@" ; $(CC) -o $@ $< $(LIBS) || ( echo "ERROR   $(CC) -o $@ $< $(LIBS)" && false )

%.o: %.c
	@echo "O        $@" ; $(COMPILE) -o $@ -c $< || ( echo "ERROR   $(COMPILE) -o $@ $<" && false )

.SUFFIXES:
.PHONY: all clean install
.SECONDARY: $(OBJECTS)

-include $(DEPENDS)
```

2 c/bin/m-binangle-from-rational.c

```
#include <stdio.h>
#include <stdlib.h>
#include <mandelbrot-symbolics.h>

int main(int argc, char **argv) {
  if (! (argc > 1)) {
    return 1;
  }
  mpq_t q;
  mpq_init(q);
  mpq_set_str(q, argv[1], 10);
  mpq_canonicalize(q);
  m_binangle ba;
  m_binangle_init(&ba);
  m_binangle_from_rational(&ba, q);
  char *s = malloc(m_binangle_strlen(&ba));
  m_binangle_to_string(s, &ba);
  printf("%s\n", s);
  free(s);
  m_binangle_clear(&ba);
  mpq_clear(q);
  return 0;
}
```

3 c/bin/m-binangle-to-rational.c

```
#include <stdio.h>
#include <stdlib.h>
#include <mandelbrot-symbolics.h>

int main(int argc, char **argv) {
  if (! (argc > 1)) {
    return 1;
  }
  m_binangle ba;
  m_binangle_init(&ba);
  m_binangle_from_string(&ba, argv[1]);
  mpq_t q;
  mpq_init(q);
  m_binangle_to_rational(q, &ba);
  m_binangle_clear(&ba);
  mpq_out_str(stdout, 10, q);
  mpq_clear(q);
  putchar('\n');
  return 0;
}
```

4 c/include/mandelbrot-symbolics.h

```
#ifndef MANDELBROT_SYMBOLICS_H
#define MANDELBROT_SYMBOLICS_H 1

#include <stdbool.h>
#include <gmp.h>

extern void m_symbolics_init(void);
extern void m_symbolics_exit(void);

struct m_block {
  mpz_t bits;
  int length;
};
typedef struct m_block m_block;

extern void m_block_init(m_block *b);
extern void m_block_clear(m_block *b);
extern void m_block_set(m_block *o, const m_block *a);
extern void m_block_empty(m_block *b);
extern void m_block_append(m_block *o, const m_block *l, const m_block *r);
extern void m_block_concatmap(m_block *o, const m_block *i, const m_block *lo, const m_block *hi);
extern const char *m_block_from_string(m_block *b, const char *s);
extern void m_block_to_string(char *s, const m_block *b);

struct m_binangle {
  m_block pre;
  m_block per;
};
typedef struct m_binangle m_binangle;

extern void m_binangle_init(m_binangle *a);
extern void m_binangle_clear(m_binangle *a);
extern void m_binangle_set(m_binangle *o, const m_binangle *a);
extern void m_binangle_from_rational(m_binangle *a, const mpq_t q);
extern void m_binangle_to_rational(mpq_t q, const m_binangle *a);
extern const char *m_binangle_from_string(m_binangle *a, const char *s);
extern int m_binangle_strlen(const m_binangle *a);
extern void m_binangle_to_string(char *s, const m_binangle *a);
extern char *m_binangle_to_new_string(const m_binangle *a);
extern void m_binangle_canonicalize(m_binangle *a);
extern bool m_binangle_is_canonical(const m_binangle *a);
extern bool m_binangle_other_representation(m_binangle *a);
extern void m_binangle_tune(m_binangle *o, const m_binangle *i, const m_block *lo, const m_block *hi);
extern void m_binangle_bulb(m_binangle *lo, m_binangle *hi, const mpq_t q);

#endif
```

5 c/lib/Makefile

```
prefix ?= $(HOME)/opt
CC ?= gcc

COMPILE := $(CC) -std=c99 -Wall -Wextra -pedantic -fPIC -O3 -pipe -ggdb -MMD -I../include -c
LINK    := $(CC) -shared -ggdb
LIBRARY := libmandelbrot-symbolics
OBJECTS := $(patsubst %.c,%.o,$(wildcard *.c))
DEPENDS := $(patsubst %.o,%.d,$(OBJECTS))

all: $(LIBRARY).a $(LIBRARY).so pkgconfig/mandelbrot-symbolics.pc

clean:
	@echo "CLEAN" ; rm -f $(OBJECTS) $(DEPENDS) $(LIBRARY).a $(LIBRARY).so pkgconfig/mandelbrot-symbolics.pc

install: $(LIBRARY).a $(LIBRARY).so ../include/mandelbrot-symbolics.h pkgconfig/mandelbrot-symbolics.pc
	install -d "$(prefix)/include" "$(prefix)/lib" "$(prefix)/lib/pkgconfig"
	install -m 644 -t "$(prefix)/include" ../include/mandelbrot-symbolics.h
	install -m 644 -t "$(prefix)/lib" $(LIBRARY).a $(LIBRARY).so
	install -m 644 -t "$(prefix)/lib/pkgconfig" pkgconfig/mandelbrot-symbolics.pc

$(LIBRARY).a: $(OBJECTS)
	@echo "A	$@" ; ar -rs $@ $^ || ( echo "ERROR	ar -rs $@ $^" && false )

$(LIBRARY).so: $(OBJECTS)
	@echo "SO	$@" ; $(LINK) -o $@ $^ -lpari -lmpc -lmpfr -lgmp -lm || ( echo "ERROR	$(LINK) -o $@ $^ -lpari -lmpc -lmpfr -lgmp -lm" && false )

%.o: %.c
	@echo "O	$@" ; $(COMPILE) -o $@ $< || ( echo "ERROR	$(COMPILE) -o $@ $<" && false )

pkgconfig/mandelbrot-symbolics.pc: pkgconfig/mandelbrot-symbolics.pc.in
	@echo "PC	$@" ; ( echo "prefix=$(prefix)" ; cat pkgconfig/mandelbrot-symbolics.pc.in ) > pkgconfig/mandelbrot-symbolics.pc || ( echo 'ERROR	( echo "prefix=$(prefix)" ; cat pkgconfig/mandelbrot-symbolics.pc.in ) > pkgconfig/mandelbrot-symbolics.pc' && false )

.SUFFIXES:
.PHONY: all clean install

-include $(DEPENDS)
```

6 c/lib/m_binangle.c

```
#include <mandelbrot-symbolics.h>
#include <assert.h>
#include <pari/pari.h>

static int m_period_pari(const mpz_t den);

extern void m_binangle_init(m_binangle *a) {
  m_block_init(&a->pre);
  m_block_init(&a->per);
  a->per.length = 1;
}

extern void m_binangle_clear(m_binangle *a) {
  m_block_clear(&a->pre);
  m_block_clear(&a->per);
}

extern void m_binangle_from_rational(m_binangle *a, const mpq_t q) {
  mpq_t p;
  mpq_init(p);
  a->pre.length = mpz_scan1(mpq_denref(q), 0);
  mpz_fdiv_q_2exp(mpq_denref(p), mpq_denref(q), a->pre.length);
  mpz_fdiv_qr(a->pre.bits, mpq_numref(p), mpq_numref(q), mpq_denref(p));
  a->per.length = m_period_pari(mpq_denref(p));
  mpz_mul_2exp(a->per.bits, mpq_numref(p), a->per.length);
  mpz_sub(a->per.bits, a->per.bits, mpq_numref(p));
  mpz_fdiv_q(a->per.bits, a->per.bits, mpq_denref(p));
  mpq_clear(p);
}

extern void m_binangle_to_rational(mpq_t q, const m_binangle *a) {
  mpz_mul_2exp(mpq_numref(q), a->pre.bits, a->per.length);
  mpz_sub(mpq_numref(q), mpq_numref(q), a->pre.bits);
  mpz_add(mpq_numref(q), mpq_numref(q), a->per.bits);
  mpz_set_si(mpq_denref(q), 0);
  mpz_setbit(mpq_denref(q), a->per.length);
  mpz_sub_ui(mpq_denref(q), mpq_denref(q), 1);
  mpz_mul_2exp(mpq_denref(q), mpq_denref(q), a->pre.length);
  mpq_canonicalize(q);
}

void m_binangle_canonicalize(m_binangle *a)
{
  mpq_t q;
  mpq_init(q);
  m_binangle_to_rational(q, a);
  m_binangle_from_rational(a, q);
  mpq_clear(q);
}

extern int m_binangle_strlen(const m_binangle *a) {
  return 4 + a->pre.length + a->per.length;
}

extern void m_binangle_to_string(char *s, const m_binangle *a) {
```

```c
  int k = 0;
  s[k++] = '.';
  m_block_to_string(s + k, &a->pre);
  k += a->pre.length;
  s[k++] = '(';
  m_block_to_string(s + k, &a->per);
  k += a->per.length;
  s[k++] = ')';
  s[k] = 0;
}

extern const char *m_binangle_from_string(m_binangle *a, const char *s) {
  const char *t = s;
  if (*t != '.') return 0;
  t = m_block_from_string(&a->pre, t + 1);
  if (*t != '(') return 0;
  t = m_block_from_string(&a->per, t + 1);
  if (*t != ')') return 0;
  if (a->per.length <= 0) return 0;
  return t + 1;
}

extern void m_binangle_tune(m_binangle *o, const m_binangle *i, const m_block *lo, const m_block *hi) {
  m_block_concatmap(&o->pre, &i->pre, lo, hi);
  m_block_concatmap(&o->per, &i->per, lo, hi);
}

extern void m_binangle_set(m_binangle *o, const m_binangle *a)
{
  m_block_set(&o->pre, &a->pre);
  m_block_set(&o->per, &a->per);
}

extern char *m_binangle_to_new_string(const m_binangle *a)
{
  int bytes = m_binangle_strlen(a);
  char *o = malloc(bytes + 1);
  m_binangle_to_string(o, a);
  return o;
}

extern bool m_binangle_other_representation(m_binangle *a)
{
  if (a->per.length == 1)
  {
    int b = mpz_get_ui(a->per.bits) & 1;
    mpz_set_ui(a->per.bits, ! b);
    if (a->pre.length > 0)
    {
      if (b)
      {
        mpz_add_ui(a->pre.bits, a->pre.bits, 1);
      }
      else
      {
        mpz_sub_ui(a->pre.bits, a->pre.bits, 1);
      }
    }
    return true;
  }
  return false;
}

/* {{{ pari-gnump {{{ */

/*
Copyright © 2014 Andreas Enge <andreas.enge@inria.fr>

This file is part of pari-gnump.

Pari-gnump is free software; you can redistribute it and/or modify it
under the terms of the GNU General Public License as published by
the Free Software Foundation; either version 2 of the License, or (at
your option) any later version.

Pari-gnump is distributed in the hope that it will be useful, but
WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with Pari-gnump. If not, see <http://www.gnu.org/licenses/>.
*/


/****************************************************************************/
/*                                                                          */
/* Functions converting between pari and mpz                                */
/*                                                                          */
/****************************************************************************/

static void mpz_set_GEN (mpz_ptr z, GEN x)
   /* Sets z to x, which needs to be of type t_INT. */

{
   const long lx = lgefint (x) - 2;
   const long sign = signe (x);
   int i;

   assert (sizeof (long) == sizeof (mp_limb_t));

   if (typ (x) != t_INT) {
#if 0
      pari_err_TYPE (”mpz_set_GEN”, x);
#endif
   } else {
   if (sign == 0)
      mpz_set_ui (z, 0);
   else {
      mpz_realloc2 (z, lx * BITS_IN_LONG);
      z->_mp_size = sign * lx;
      for (i = 0; i < lx; i++)
         (z->_mp_d) [i] = *int_W (x, i);
    }
  }
}

/******************************************************************************/

static GEN mpz_get_GEN (mpz_srcptr z)
   /* Returns the GEN of type t_INT corresponding to z. */

{
   const long lz = z->_mp_size;
   const long lx = labs (lz);
   const long lx2 = lx + 2;
   int i;
   GEN x = cgeti (lx2);

   assert (sizeof (long) == sizeof (mp_limb_t));

   x [1] = evalsigne ((lz > 0 ? 1 : (lz < 0 ? -1 : 0))) | evallgefint (lx2);
   for (i = 0; i < lx; i++)
      *int_W (x, i) = (z->_mp_d) [i];

   return x;
}

/* }}} pari-gnump }}} */

static int m_period_pari(const mpz_t den) {
  m_symbolics_init();
  mpz_t n;
  mpz_init(n);
  pari_sp av = avma;
  mpz_set_GEN(n, order(gmodulsg(2, mpz_get_GEN(den))));
  avma = av;
  int p = 0;
  if (mpz_fits_sint_p(n)) {
    p = mpz_get_si(n);
  }
  mpz_clear(n);
  return p;
}
```

7 c/lib/m_block.c

```
#include <mandelbrot-symbolics.h>

extern void m_block_init(m_block *b) {
  mpz_init(b->bits);
  b->length = 0;
}

extern void m_block_clear(m_block *b) {
  mpz_clear(b->bits);
}

extern void m_block_empty(m_block *b) {
  mpz_set_si(b->bits, 0);
```

```c
  b->length = 0;
}

extern void m_block_append(m_block *o, const m_block *l, const m_block *r) {
  if (o == r) {
    mpz_t rbits;
    mpz_init(rbits);
    mpz_set(rbits, r->bits);
    mpz_mul_2exp(o->bits, l->bits, r->length);
    mpz_ior(o->bits, o->bits, rbits);
    o->length = l->length + r->length;
    mpz_clear(rbits);
  } else {
    mpz_mul_2exp(o->bits, l->bits, r->length);
    mpz_ior(o->bits, o->bits, r->bits);
    o->length = l->length + r->length;
  }
}

extern void m_block_concatmap(m_block *o, const m_block *i, const m_block *lo, const m_block *hi) {
  if (o == i || o == lo || o == hi) {
    m_block o2;
    m_block_init(&o2);
    m_block_empty(&o2);
    for (int k = 0; k < i->length; ++k) {
      m_block_append(&o2, &o2, mpz_tstbit(i->bits, i->length - 1 - k) ? hi : lo);
    }
    mpz_set(o->bits, o2.bits);
    o->length = o2.length;
    m_block_clear(&o2);
  } else {
    m_block_empty(o);
    for (int k = 0; k < i->length; ++k) {
      m_block_append(o, o, mpz_tstbit(i->bits, i->length - 1 - k) ? hi : lo);
    }
  }
}

extern const char *m_block_from_string(m_block *b, const char *s) {
  mpz_set_si(b->bits, 0);
  int i;
  for (i = 0; s[i] == '0' || s[i] == '1'; ++i) {
    mpz_mul_2exp(b->bits, b->bits, 1);
    if (s[i] == '1') {
      mpz_setbit(b->bits, 0);
    }
  }
  b->length = i;
  return s + i;
}

extern void m_block_to_string(char *s, const m_block *b) {
  for (int i = 0; i < b->length; ++i) {
    s[i] = '0' + mpz_tstbit(b->bits, b->length - 1 - i);
  }
  s[b->length] = 0;
}

extern void m_block_set(m_block *o, const m_block *a)
{
  mpz_set(o->bits, a->bits);
  o->length = a->length;
}
```

8 c/lib/m_symbolics.c

```
#include <mandelbrot-symbolics.h>
#include <stdbool.h>
#include <stdlib.h>
#include <pari/pari.h>

static bool m_symbolics_initted = false;
static bool m_symbolics_atexit = false;

extern void m_symbolics_init(void) {
  if (! m_symbolics_initted) {
    pari_init_opts(500000, 0, INIT_DFTm);// | INIT_noIMTm);
    m_symbolics_initted = true;
    if (! m_symbolics_atexit) {
      atexit(m_symbolics_exit);
      m_symbolics_atexit = true;
    }
  }
}

extern void m_symbolics_exit(void) {
  if (m_symbolics_initted) {
    pari_close_opts(INIT_DFTm);// | INIT_noIMTm);
    m_symbolics_initted = false;
  }
}
```

9 c/lib/pkgconfig/mandelbrot-symbolics.pc.in

```
exec_prefix=${prefix}
libdir=${exec_prefix}/lib
includedir=${prefix}/include

Name: mandelbrot-symbolics
Description: Symbolic algorithms related to the Mandelbrot set
Version: 0.1.0.0
URL: https://code.mathr.co.uk/mandelbrot-symbolics
Libs: -L${libdir} -lmandelbrot-symbolics
Libs.private: -lpari -lmpc -lmpfr -lgmp -lm
Cflags: -I${includedir}
```

10 COPYING

```
                    GNU GENERAL PUBLIC LICENSE
                       Version 3, 29 June 2007
```

Copyright (C) 2007 Free Software Foundation, Inc. <http://fsf.org/>
Everyone is permitted to copy and distribute verbatim copies
of this license document, but changing it is not allowed.

Preamble

The GNU General Public License is a free, copyleft license for software and other kinds of works.

The licenses for most software and other practical works are designed to take away your freedom to share and change the works. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change all versions of a program--to make sure it remains free software for all its users. We, the Free Software Foundation, use the GNU General Public License for most of our software; it applies also to any other work released this way by its authors. You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for them if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs, and that you know you can do these things.

To protect your rights, we need to prevent others from denying you these rights or asking you to surrender the rights. Therefore, you have certain responsibilities if you distribute copies of the software, or if you modify it: responsibilities to respect the freedom of others.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must pass on to the recipients the same freedoms that you received. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

Developers that use the GNU GPL protect your rights with two steps: (1) assert copyright on the software, and (2) offer you this License giving you legal permission to copy, distribute and/or modify it.

For the developers' and authors' protection, the GPL clearly explains that there is no warranty for this free software. For both users' and authors' sake, the GPL requires that modified versions be marked as changed, so that their problems will not be attributed erroneously to authors of previous versions.

Some devices are designed to deny users access to install or run modified versions of the software inside them, although the manufacturer can do so. This is fundamentally incompatible with the aim of protecting users' freedom to change the software. The systematic pattern of such abuse occurs in the area of products for individuals to use, which is precisely where it is most unacceptable. Therefore, we have designed this version of the GPL to prohibit the practice for those products. If such problems arise substantially in other domains, we stand ready to extend this provision to those domains in future versions of the GPL, as needed to protect the freedom of users.

Finally, every program is threatened constantly by software patents. States should not allow patents to restrict development and use of software on general-purpose computers, but in those that do, we wish to avoid the special danger that patents applied to a free program could make it effectively proprietary. To prevent this, the GPL assures that patents cannot be used to render the program non-free.

The precise terms and conditions for copying, distribution and modification follow.

TERMS AND CONDITIONS

0. Definitions.

”This License” refers to version 3 of the GNU General Public License.

”Copyright” also means copyright-like laws that apply to other kinds of works, such as semiconductor masks.

”The Program” refers to any copyrightable work licensed under this License. Each licensee is addressed as ”you”. ”Licensees” and ”recipients” may be individuals or organizations.

To ”modify” a work means to copy from or adapt all or part of the work in a fashion requiring copyright permission, other than the making of an exact copy. The resulting work is called a ”modified version” of the earlier work or a work ”based on” the earlier work.

A ”covered work” means either the unmodified Program or a work based on the Program.

To ”propagate” a work means to do anything with it that, without permission, would make you directly or secondarily liable for infringement under applicable copyright law, except executing it on a computer or modifying a private copy. Propagation includes copying, distribution (with or without modification), making available to the public, and in some countries other activities as well.

To ”convey” a work means any kind of propagation that enables other parties to make or receive copies. Mere interaction with a user through a computer network, with no transfer of a copy, is not conveying.

An interactive user interface displays ”Appropriate Legal Notices” to the extent that it includes a convenient and prominently visible feature that (1) displays an appropriate copyright notice, and (2) tells the user that there is no warranty for the work (except to the extent that warranties are provided), that licensees may convey the work under this License, and how to view a copy of this License. If the interface presents a list of user commands or options, such as a menu, a prominent item in the list meets this criterion.

1. Source Code.

The ”source code” for a work means the preferred form of the work for making modifications to it. ”Object code” means any non-source form of a work.

A ”Standard Interface” means an interface that either is an official standard defined by a recognized standards body, or, in the case of interfaces specified for a particular programming language, one that is widely used among developers working in that language.

The ”System Libraries” of an executable work include anything, other than the work as a whole, that (a) is included in the normal form of packaging a Major Component, but which is not part of that Major Component, and (b) serves only to enable use of the work with that Major Component, or to implement a Standard Interface for which an implementation is available to the public in source code form. A ”Major Component”, in this context, means a major essential component (kernel, window system, and so on) of the specific operating system (if any) on which the executable work runs, or a compiler used to produce the work, or an object code interpreter used to run it.

The ”Corresponding Source” for a work in object code form means all the source code needed to generate, install, and (for an executable work) run the object code and to modify the work, including scripts to control those activities. However, it does not include the work's System Libraries, or general-purpose tools or generally available free programs which are used unmodified in performing those activities but which are not part of the work. For example, Corresponding Source includes interface definition files associated with source files for the work, and the source code for shared libraries and dynamically linked subprograms that the work is specifically designed to require, such as by intimate data communication or control flow between those subprograms and other parts of the work.

The Corresponding Source need not include anything that users can regenerate automatically from other parts of the Corresponding Source.

The Corresponding Source for a work in source code form is that same work.

2. Basic Permissions.

All rights granted under this License are granted for the term of copyright on the Program, and are irrevocable provided the stated conditions are met. This License explicitly affirms your unlimited permission to run the unmodified Program. The output from running a covered work is covered by this License only if the output, given its content, constitutes a covered work. This License acknowledges your rights of fair use or other equivalent, as provided by copyright law.

You may make, run and propagate covered works that you do not convey, without conditions so long as your license otherwise remains in force. You may convey covered works to others for the sole purpose of having them make modifications exclusively for you, or provide you with facilities for running those works, provided that you comply with the terms of this License in conveying all material for which you do not control copyright. Those thus making or running the covered works for you must do so exclusively on your behalf, under your direction and control, on terms that prohibit them from making any copies of your copyrighted material outside their relationship with you.

```
  Conveying under any other circumstances is permitted solely under
the conditions stated below.  Sublicensing is not allowed; section 10
makes it unnecessary.

  3. Protecting Users' Legal Rights From Anti-Circumvention Law.

  No covered work shall be deemed part of an effective technological
measure under any applicable law fulfilling obligations under article
11 of the WIPO copyright treaty adopted on 20 December 1996, or
similar laws prohibiting or restricting circumvention of such
measures.

  When you convey a covered work, you waive any legal power to forbid
circumvention of technological measures to the extent such circumvention
is effected by exercising rights under this License with respect to
the covered work, and you disclaim any intention to limit operation or
modification of the work as a means of enforcing, against the work's
users, your or third parties' legal rights to forbid circumvention of
technological measures.

  4. Conveying Verbatim Copies.

  You may convey verbatim copies of the Program's source code as you
receive it, in any medium, provided that you conspicuously and
appropriately publish on each copy an appropriate copyright notice;
keep intact all notices stating that this License and any
non-permissive terms added in accord with section 7 apply to the code;
keep intact all notices of the absence of any warranty; and give all
recipients a copy of this License along with the Program.

  You may charge any price or no price for each copy that you convey,
and you may offer support or warranty protection for a fee.

  5. Conveying Modified Source Versions.

  You may convey a work based on the Program, or the modifications to
produce it from the Program, in the form of source code under the
terms of section 4, provided that you also meet all of these conditions:

    a) The work must carry prominent notices stating that you modified
    it, and giving a relevant date.

    b) The work must carry prominent notices stating that it is
    released under this License and any conditions added under section
    7.  This requirement modifies the requirement in section 4 to
    "keep intact all notices".

    c) You must license the entire work, as a whole, under this
    License to anyone who comes into possession of a copy.  This
    License will therefore apply, along with any applicable section 7
    additional terms, to the whole of the work, and all its parts,
    regardless of how they are packaged.  This License gives no
    permission to license the work in any other way, but it does not
    invalidate such permission if you have separately received it.

    d) If the work has interactive user interfaces, each must display
    Appropriate Legal Notices; however, if the Program has interactive
    interfaces that do not display Appropriate Legal Notices, your
    work need not make them do so.

  A compilation of a covered work with other separate and independent
works, which are not by their nature extensions of the covered work,
and which are not combined with it such as to form a larger program,
in or on a volume of a storage or distribution medium, is called an
"aggregate" if the compilation and its resulting copyright are not
used to limit the access or legal rights of the compilation's users
beyond what the individual works permit.  Inclusion of a covered work
in an aggregate does not cause this License to apply to the other
parts of the aggregate.

  6. Conveying Non-Source Forms.

  You may convey a covered work in object code form under the terms
of sections 4 and 5, provided that you also convey the
machine-readable Corresponding Source under the terms of this License,
in one of these ways:

    a) Convey the object code in, or embodied in, a physical product
    (including a physical distribution medium), accompanied by the
    Corresponding Source fixed on a durable physical medium
    customarily used for software interchange.

    b) Convey the object code in, or embodied in, a physical product
    (including a physical distribution medium), accompanied by a
    written offer, valid for at least three years and valid for as
    long as you offer spare parts or customer support for that product
    model, to give anyone who possesses the object code either (1) a
    copy of the Corresponding Source for all the software in the
    product that is covered by this License, on a durable physical
    medium customarily used for software interchange, for a price no
    more than your reasonable cost of physically performing this
    conveying of source, or (2) access to copy the
    Corresponding Source from a network server at no charge.

    c) Convey individual copies of the object code with a copy of the
    written offer to provide the Corresponding Source.  This
    alternative is allowed only occasionally and noncommercially, and
    only if you received the object code with such an offer, in accord
    with subsection 6b.

    d) Convey the object code by offering access from a designated
    place (gratis or for a charge), and offer equivalent access to the
    Corresponding Source in the same way through the same place at no
    further charge.  You need not require recipients to copy the
    Corresponding Source along with the object code.  If the place to
    copy the object code is a network server, the Corresponding Source
    may be on a different server (operated by you or a third party)
    that supports equivalent copying facilities, provided you maintain
    clear directions next to the object code saying where to find the
    Corresponding Source.  Regardless of what server hosts the
    Corresponding Source, you remain obligated to ensure that it is
    available for as long as needed to satisfy these requirements.

    e) Convey the object code using peer-to-peer transmission, provided
```

you inform other peers where the object code and Corresponding Source of the work are being offered to the general public at no charge under subsection 6d.

A separable portion of the object code, whose source code is excluded from the Corresponding Source as a System Library, need not be included in conveying the object code work.

A "User Product" is either (1) a "consumer product", which means any tangible personal property which is normally used for personal, family, or household purposes, or (2) anything designed or sold for incorporation into a dwelling. In determining whether a product is a consumer product, doubtful cases shall be resolved in favor of coverage. For a particular product received by a particular user, "normally used" refers to a typical or common use of that class of product, regardless of the status of the particular user or of the way in which the particular user actually uses, or expects or is expected to use, the product. A product is a consumer product regardless of whether the product has substantial commercial, industrial or non-consumer uses, unless such uses represent the only significant mode of use of the product.

"Installation Information" for a User Product means any methods, procedures, authorization keys, or other information required to install and execute modified versions of a covered work in that User Product from a modified version of its Corresponding Source. The information must suffice to ensure that the continued functioning of the modified object code is in no case prevented or interfered with solely because modification has been made.

If you convey an object code work under this section in, or with, or specifically for use in, a User Product, and the conveying occurs as part of a transaction in which the right of possession and use of the User Product is transferred to the recipient in perpetuity or for a fixed term (regardless of how the transaction is characterized), the Corresponding Source conveyed under this section must be accompanied by the Installation Information. But this requirement does not apply if neither you nor any third party retains the ability to install modified object code on the User Product (for example, the work has been installed in ROM).

The requirement to provide Installation Information does not include a requirement to continue to provide support service, warranty, or updates for a work that has been modified or installed by the recipient, or for the User Product in which it has been modified or installed. Access to a network may be denied when the modification itself materially and adversely affects the operation of the network or violates the rules and protocols for communication across the network.

Corresponding Source conveyed, and Installation Information provided, in accord with this section must be in a format that is publicly documented (and with an implementation available to the public in source code form), and must require no special password or key for unpacking, reading or copying.

7. Additional Terms.

"Additional permissions" are terms that supplement the terms of this

License by making exceptions from one or more of its conditions. Additional permissions that are applicable to the entire Program shall be treated as though they were included in this License, to the extent that they are valid under applicable law. If additional permissions apply only to part of the Program, that part may be used separately under those permissions, but the entire Program remains governed by this License without regard to the additional permissions.

When you convey a copy of a covered work, you may at your option remove any additional permissions from that copy, or from any part of it. (Additional permissions may be written to require their own removal in certain cases when you modify the work.) You may place additional permissions on material, added by you to a covered work, for which you have or can give appropriate copyright permission.

Notwithstanding any other provision of this License, for material you add to a covered work, you may (if authorized by the copyright holders of that material) supplement the terms of this License with terms:

a) Disclaiming warranty or limiting liability differently from the terms of sections 15 and 16 of this License; or

b) Requiring preservation of specified reasonable legal notices or author attributions in that material or in the Appropriate Legal Notices displayed by works containing it; or

c) Prohibiting misrepresentation of the origin of that material, or requiring that modified versions of such material be marked in reasonable ways as different from the original version; or

d) Limiting the use for publicity purposes of names of licensors or authors of the material; or

e) Declining to grant rights under trademark law for use of some trade names, trademarks, or service marks; or

f) Requiring indemnification of licensors and authors of that material by anyone who conveys the material (or modified versions of it) with contractual assumptions of liability to the recipient, for any liability that these contractual assumptions directly impose on those licensors and authors.

All other non-permissive additional terms are considered ”further restrictions” within the meaning of section 10. If the Program as you received it, or any part of it, contains a notice stating that it is governed by this License along with a term that is a further restriction, you may remove that term. If a license document contains a further restriction but permits relicensing or conveying under this License, you may add to a covered work material governed by the terms of that license document, provided that the further restriction does not survive such relicensing or conveying.

If you add terms to a covered work in accord with this section, you must place, in the relevant source files, a statement of the additional terms that apply to those files, or a notice indicating where to find the applicable terms.

Additional terms, permissive or non-permissive, may be stated in the form of a separately written license, or stated as exceptions; the above requirements apply either way.

8. Termination.

You may not propagate or modify a covered work except as expressly provided under this License. Any attempt otherwise to propagate or modify it is void, and will automatically terminate your rights under this License (including any patent licenses granted under the third paragraph of section 11).

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, you do not qualify to receive new licenses for the same material under section 10.

9. Acceptance Not Required for Having Copies.

You are not required to accept this License in order to receive or run a copy of the Program. Ancillary propagation of a covered work occurring solely as a consequence of using peer-to-peer transmission to receive a copy likewise does not require acceptance. However, nothing other than this License grants you permission to propagate or modify any covered work. These actions infringe copyright if you do not accept this License. Therefore, by modifying or propagating a covered work, you indicate your acceptance of this License to do so.

10. Automatic Licensing of Downstream Recipients.

Each time you convey a covered work, the recipient automatically receives a license from the original licensors, to run, modify and propagate that work, subject to this License. You are not responsible for enforcing compliance by third parties with this License.

An ”entity transaction” is a transaction transferring control of an organization, or substantially all assets of one, or subdividing an organization, or merging organizations. If propagation of a covered work results from an entity transaction, each party to that transaction who receives a copy of the work also receives whatever licenses to the work the party's predecessor in interest had or could give under the previous paragraph, plus a right to possession of the

```
Corresponding Source of the work from the predecessor in interest, if
the predecessor has it or can get it with reasonable efforts.

  You may not impose any further restrictions on the exercise of the
rights granted or affirmed under this License.  For example, you may
not impose a license fee, royalty, or other charge for exercise of
rights granted under this License, and you may not initiate litigation
(including a cross-claim or counterclaim in a lawsuit) alleging that
any patent claim is infringed by making, using, selling, offering for
sale, or importing the Program or any portion of it.

  11. Patents.

  A ”contributor” is a copyright holder who authorizes use under this
License of the Program or a work on which the Program is based.  The
work thus licensed is called the contributor's ”contributor version”.

  A contributor's ”essential patent claims” are all patent claims
owned or controlled by the contributor, whether already acquired or
hereafter acquired, that would be infringed by some manner, permitted
by this License, of making, using, or selling its contributor version,
but do not include claims that would be infringed only as a
consequence of further modification of the contributor version.  For
purposes of this definition, ”control” includes the right to grant
patent sublicenses in a manner consistent with the requirements of
this License.

  Each contributor grants you a non-exclusive, worldwide, royalty-free
patent license under the contributor's essential patent claims, to
make, use, sell, offer for sale, import and otherwise run, modify and
propagate the contents of its contributor version.

  In the following three paragraphs, a ”patent license” is any express
agreement or commitment, however denominated, not to enforce a patent
(such as an express permission to practice a patent or covenant not to
sue for patent infringement).  To ”grant” such a patent license to a
party means to make such an agreement or commitment not to enforce a
patent against the party.

  If you convey a covered work, knowingly relying on a patent license,
and the Corresponding Source of the work is not available for anyone
to copy, free of charge and under the terms of this License, through a
publicly available network server or other readily accessible means,
then you must either (1) cause the Corresponding Source to be so
available, or (2) arrange to deprive yourself of the benefit of the
patent license for this particular work, or (3) arrange, in a manner
consistent with the requirements of this License, to extend the patent
license to downstream recipients.  ”Knowingly relying” means you have
actual knowledge that, but for the patent license, your conveying the
covered work in a country, or your recipient's use of the covered work
in a country, would infringe one or more identifiable patents in that
country that you have reason to believe are valid.

  If, pursuant to or in connection with a single transaction or
arrangement, you convey, or propagate by procuring conveyance of, a
covered work, and grant a patent license to some of the parties
receiving the covered work authorizing them to use, propagate, modify
```

or convey a specific copy of the covered work, then the patent license you grant is automatically extended to all recipients of the covered work and works based on it.

A patent license is ”discriminatory” if it does not include within the scope of its coverage, prohibits the exercise of, or is conditioned on the non−exercise of one or more of the rights that are specifically granted under this License. You may not convey a covered work if you are a party to an arrangement with a third party that is in the business of distributing software, under which you make payment to the third party based on the extent of your activity of conveying the work, and under which the third party grants, to any of the parties who would receive the covered work from you, a discriminatory patent license (a) in connection with copies of the covered work conveyed by you (or copies made from those copies), or (b) primarily for and in connection with specific products or compilations that contain the covered work, unless you entered into that arrangement, or that patent license was granted, prior to 28 March 2007.

Nothing in this License shall be construed as excluding or limiting any implied license or other defenses to infringement that may otherwise be available to you under applicable patent law.

12. No Surrender of Others’ Freedom.

If conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot convey a covered work so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not convey it at all. For example, if you agree to terms that obligate you to collect a royalty for further conveying from those to whom you convey the Program, the only way you could satisfy both those terms and this License would be to refrain entirely from conveying the Program.

13. Use with the GNU Affero General Public License.

Notwithstanding any other provision of this License, you have permission to link or combine any covered work with a work licensed under version 3 of the GNU Affero General Public License into a single combined work, and to convey the resulting work. The terms of this License will continue to apply to the part which is the covered work, but the special requirements of the GNU Affero General Public License, section 13, concerning interaction through a network will apply to the combination as such.

14. Revised Versions of this License.

The Free Software Foundation may publish revised and/or new versions of the GNU General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies that a certain numbered version of the GNU General Public License ”or any later version” applies to it, you have the option of following the terms and conditions either of that numbered version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of the GNU General Public License, you may choose any version ever published by the Free Software Foundation.

If the Program specifies that a proxy can decide which future versions of the GNU General Public License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Program.

Later license versions may give you additional or different permissions. However, no additional obligations are imposed on any author or copyright holder as a result of your choosing to follow a later version.

15. Disclaimer of Warranty.

THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. Limitation of Liability.

IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MODIFIES AND/OR CONVEYS THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

17. Interpretation of Sections 15 and 16.

If the disclaimer of warranty and limitation of liability provided above cannot be given local legal effect according to their terms, reviewing courts shall apply local law that most closely approximates an absolute waiver of all civil liability in connection with the Program, unless a warranty or assumption of liability accompanies a copy of the Program in return for a fee.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively

```
state the exclusion of warranty; and each file should have at least
the "copyright" line and a pointer to where the full notice is found.

    <one line to give the program's name and a brief idea of what it does.>
    Copyright (C) <year>  <name of author>

    This program is free software: you can redistribute it and/or modify
    it under the terms of the GNU General Public License as published by
    the Free Software Foundation, either version 3 of the License, or
    (at your option) any later version.

    This program is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
    GNU General Public License for more details.

    You should have received a copy of the GNU General Public License
    along with this program.  If not, see <http://www.gnu.org/licenses/>.

Also add information on how to contact you by electronic and paper mail.

  If the program does terminal interaction, make it output a short
notice like this when it starts in an interactive mode:

    <program>  Copyright (C) <year>  <name of author>
    This program comes with ABSOLUTELY NO WARRANTY; for details type ‘show w’.
    This is free software, and you are welcome to redistribute it
    under certain conditions; type ‘show c’ for details.

The hypothetical commands ‘show w’ and ‘show c’ should show the appropriate
parts of the General Public License.  Of course, your program's commands
might be different; for a GUI interface, you would use an "about box".

  You should also get your employer (if you work as a programmer) or school,
if any, to sign a "copyright disclaimer" for the program, if necessary.
For more information on this, and how to apply and follow the GNU GPL, see
<http://www.gnu.org/licenses/>.

  The GNU General Public License does not permit incorporating your program
into proprietary programs.  If your program is a subroutine library, you
may consider it more useful to permit linking proprietary applications with
the library.  If this is what you want to do, use the GNU Lesser General
Public License instead of this License.  But first, please read
<http://www.gnu.org/philosophy/why-not-lgpl.html>.
```

11 .gitignore

```
*.a
*.d
*.o
*.pc
*.so
c/bin/m-binangle-from-rational
c/bin/m-binangle-to-rational
dist
.cabal-sandbox
cabal.sandbox.config
```

12 hs/lib/Mandelbrot/Symbolics/AngledAddress.hs

```haskell
module Mandelbrot.Symbolics.AngledAddress
  ( AngledAddress(..)
  , angledAddress
  , addressAngles
  , splitAddress
  , joinAddress
  , stripAddress
  ) where

import Control.Monad
  ( guard
  )
import Data.Bits
  ( shiftL
  , shiftR
  , (.&.)
  , (.|.)
  )
import Data.List
  ( elemIndex
  )

import Data.Strict.Tuple
  ( Pair((:!:))
  )

import Mandelbrot.Symbolics.ExternalAngle
  ( ExternalAngle
  , Tuning(..)
  )
import Mandelbrot.Symbolics.InternalAddress
  ( InternalAddress(..)
  , internalAddress
  )
import Mandelbrot.Symbolics.InternalAngle
  ( InternalAngle
  )
import Mandelbrot.Symbolics.Kneading
  ( Kneading
  , kneading
  , unwrap
  )
import Mandelbrot.Symbolics.Period
  ( Period(preperiod, period)
  )
import Mandelbrot.Symbolics.Rational
  ( (%)
  , numerator
  , denominator
  , zero
  , one
  , wrap
  , double
  , doubleOdd
  )
```

```haskell
-- | Angled internal addresses have internal angles between each period in an
--   internal address.
data AngledAddress
  = Unangled !Int
  | Angled !Int !InternalAngle AngledAddress
  deriving (Read, Show, Eq, Ord)

-- | The period of an angled internal address.
instance Period AngledAddress where
  preperiod _ = 0
  period (Unangled p) = p
  period (Angled _ _ a) = period a

{-
-- | Builds a valid 'AngledAddress' from a list, checking the
--   precondition that only the last 'Maybe Angle' should be 'Nothing',
--   and the 'Integer' must be strictly increasing.
angledFromList :: [(Int, Maybe InternalAngle)] -> Maybe AngledAddress
angledFromList = fromList' 0
  where
    fromList' x [(n, Nothing)] | n > x = Just (Unangled n)
    fromList' x ((n, Just r) : xs) | n > x && zero < r && r < one = Angled n r `fmap` fromList' n xs
    fromList' _ _ = Nothing
-}

unsafeAngledFromList :: [(Int, Maybe InternalAngle)] -> AngledAddress
unsafeAngledFromList = fromList' 0
  where
    fromList' x [(n, Nothing)] | n > x = Unangled n
    fromList' x ((n, Just r) : xs) | n > x && zero < r && r < one = Angled n r (fromList' n xs)
    fromList' x xs = error $ "AngledAddress.unsafeAngledFromList " ++ show (x, xs)

-- | Convert an 'AngledAddress' to a list.
angledToList :: AngledAddress -> [(Int, Maybe InternalAngle)]
angledToList (Unangled n) = [(n, Nothing)]
angledToList (Angled n r a) = (n, Just r) : angledToList a

denominators :: InternalAddress -> Kneading -> [Int]
denominators (InternalAddress xs) v = denominators' xs
  where
    denominators' (s0:ss@(s1:_)) =
      let rr = r s0 s1
      in  (((s1 - rr) `div` s0) + if (s0 ==) . head . dropWhile (< s0) . iterate p $ rr then 1 else 2) : denominators' ss
    denominators' _ = []
    r s s' = case s' `mod` s of
      0 -> s
      t -> t
    p = rho v

rho :: Kneading -> Int -> Int
rho v = rho'
  where
    rho' r
```

```
      | r >= 1 && r `mod` n /= 0 = ((1 + r) +) . length . takeWhile id . zipWith↙
          ↳ (==) (unwrap v) . drop r $ (unwrap v)
      | otherwise = rho' (r + 1)
    n = period v

numerators :: ExternalAngle -> InternalAddress -> [Int] -> [Int]
numerators r (InternalAddress a) qs = zipWith num a qs
  where
    num s q = length . filter (<= r) . map (rs !!) $ [0 .. q - 2]
      where
        rs = iterate (\t -> foldr (.) id (replicate s (if even (denominator t) ↙
            ↳ then double else doubleOdd)) $ t) (wrap r)

-- | The angled internal address corresponding to an external angle.
angledAddress :: ExternalAngle -> Maybe AngledAddress
angledAddress r0 = do
  let r = wrap r0
      k = kneading r
  i@(InternalAddress is) <- internalAddress k
  let d = denominators i k
      n = numerators r i d
  return . unsafeAngledFromList . zip is . (++ [Nothing]) . map Just . zipWith ↙
      ↳ (\a b -> fromIntegral a % fromIntegral b) n $ d

-- | Split an angled internal address at the last island.
splitAddress :: AngledAddress -> (AngledAddress, [InternalAngle])
splitAddress a =
  let (ps0, rs0) = unzip $ angledToList a
      ps1 = reverse ps0
      rs1 = reverse (Nothing : init rs0)
      prs1 = zip ps1 rs1
      f ((p, Just r):qrs@((q, _):_)) acc
        | p == fromIntegral (denominator r) * q = f qrs (r : acc)
      f prs acc = g prs acc
      g prs acc =
        let (ps2, rs2) = unzip prs
            ps3 = reverse ps2
            rs3 = reverse (Nothing : init rs2)
            prs3 = zip ps3 rs3
            aa = unsafeAngledFromList prs3
        in (aa, acc)
  in f prs1 []

-- | The inverse of 'splitAddress'.
joinAddress :: AngledAddress -> [InternalAngle] -> AngledAddress
joinAddress (Unangled p) [] = Unangled p
joinAddress (Unangled p) (r:rs) = Angled p r (joinAddress (Unangled $ p * ↙
    ↳ fromIntegral (denominator r)) rs)
joinAddress (Angled p r a) rs = Angled p r (joinAddress a rs)

-- | Discard angle information from an internal address.
stripAddress :: AngledAddress -> InternalAddress
stripAddress = InternalAddress . map fst . angledToList


-- | The pair of external angles whose rays land at the root of the
--   hyperbolic component described by the angled internal address.
```

```haskell
addressAngles :: AngledAddress -> Maybe (ExternalAngle, ExternalAngle)
addressAngles = externalAngles' 1 (zero, one)

externalAngles' :: Int -> (ExternalAngle, ExternalAngle) -> AngledAddress -> Maybe (ExternalAngle, ExternalAngle)
externalAngles' p0 lohi a0@(Unangled p)
  | p0 /= p = case wakees lohi p of
      [lh] -> externalAngles' p lh a0
      _ -> Nothing
  | otherwise = Just lohi
externalAngles' p0 lohi a0@(Angled p r a)
  | p0 /= p = case wakees lohi p of
      [lh] -> externalAngles' p lh a0
      _ -> Nothing
  | otherwise = do
      let num = numerator r
          den = denominator r
          ws = wakees (zero, one) (fromIntegral den)
          nums = [ num' | num' <- [ 1.. den - 1 ], let r' = num' % den :: ExternalAngle, denominator r' == den ]
          nws, nnums :: Int
          nws = length ws
          nnums = length nums
      guard (nws == nnums)
      i <- elemIndex num nums
      (l,h) <- safeIndex ws i
      externalAngles' (p * fromIntegral den) (if p > 1 then (tune lohi l, tune lohi h) else (l, h)) a

wakees :: (ExternalAngle, ExternalAngle) -> Int -> [(ExternalAngle, ExternalAngle)]
wakees (lo, hi) q =
  let gaps (l :!: h) n
        | n == 0 = [(l :!: h)]
        | n > 0 = let gs = gaps (l :!: h) (n - 1)
                      cs = candidates n gs
                  in  accumulate cs gs
        | otherwise = error $ "AngledAddress.gaps: !(n >= 0) " ++ show n
      candidates n gs =
        let den = (1 `shiftL` n) - 1
        in  [ r
            | (l :!: h) <- gs
            , num <- [ ceiling' l n .. floor' h n ]
            , fullperiod n num
            , let r = num % den
            , l < r, r < h
            ]
      accumulate [] ws = ws
      accumulate (l : h : lhs) ws =
        let (ls, ms@((ml :!: _):_)) = break (l `inside`) ws
            (_s, (_ :!: rh):rs) = break (h `inside`) ms
        in  ls ++ [(ml :!: l)] ++ accumulate lhs ((h :!: rh) : rs)
      accumulate _ _ = error "AngledAddress.accumulate !even"
      inside x (l :!: h) = l < x && x < h
      fullperiod bs = \n -> and [ (((n `shiftR` b) .|. (n `shiftL` (bs - b))) .&. mask) /= n | b <- factors ]
        where
```

```
          factors = [ b | b <- [ bs - 1, bs - 2 .. 1 ], bs `mod` b == 0 ]
          mask = (1 `shiftL` bs) - 1
  in  chunk2 . candidates q . gaps (lo :!: hi) $ (q - 1)

chunk2 :: [t] -> [(t, t)]
chunk2 [] = []
chunk2 (x:y:zs) = (x, y) : chunk2 zs
chunk2 _ = error "AngledAddress.chunk2 !even"

safeIndex :: [a] -> Int -> Maybe a
safeIndex [] _ = Nothing
safeIndex (x:xs) i
  | i < 0 = Nothing
  | i > 0 = safeIndex xs (i - 1)
  | otherwise = Just x

-- | ceiling' x y = ceiling $ x * (2^y - 1)
ceiling' :: ExternalAngle -> Int -> Integer
ceiling' x y = ((numerator x `shiftL` y) - numerator x + denominator x - 1) `div` denominator x

-- | floor' x y = floor $ x * (2^y - 1)
floor' :: ExternalAngle -> Int -> Integer
floor' x y = ((numerator x `shiftL` y) - numerator x) `div` denominator x
```

13 hs/lib/Mandelbrot/Symbolics/Block.hs

```
module Mandelbrot.Symbolics.Block
  ( Block(..)
  , (!)
  , (!<)
  , concatMap
  , compact
  , rotate
  , splitAt
  , take
  , drop
  , toList
  , toListReversed
  , fromList
  , singleton
  ) where

import Prelude hiding
  ( concatMap
  , splitAt
  , take
  , drop
  , (!!)
  )
import Data.Bits
  ( shiftL
  , shiftR
  , setBit
  , testBit
  , bit
  , (.&.)
  , (.|.)
  )
import Data.List
  ( foldl'
  )
import Data.Monoid
  ( Monoid(..)
  )
import Data.Semigroup
  ( Semigroup(..)
  )

data Block = Block !Integer !Int
  deriving (Eq, Read, Show)

instance Semigroup Block where
  Block xs x <> Block ys y = Block ((xs `shiftL` y) .|. ys) (x + y)

instance Monoid Block where
  mempty = Block 0 0

(!) :: Block -> Int -> Bool
Block b l ! i = b `testBit` (l - i - 1)

(!<) :: Block -> Int -> Bool
Block b _ !< i = b `testBit` i

concatMap :: (Bool -> Block) -> Block -> Block
concatMap f b@(Block _ l) = mconcat [ f (b ! i) | i <- [ 0 .. l - 1 ] ]

compact :: Block -> Block
compact b@(Block _ l) = head
  [ p
  | m <- [1 .. l]
  , l `mod` m == 0
  , let p = take m b
  , b == mconcat (replicate (l `div` m) p)
  ]

rotate :: Block -> Int -> Block
rotate (Block _ 0) _ = mempty
rotate (Block b l) i = Block (((b `shiftL` j) .&. mask) .|. (b `shiftR` k)) l
  where
    j = i `mod` l
    k = l - i
    mask = bit l - 1

splitAt :: Int -> Block -> (Block, Block)
splitAt i (Block b l) = (Block (b `shiftR` m) i, Block (b .&. mask) m)
  where
    m = l - i
    mask = bit m - 1

take :: Int -> Block -> Block
take i = fst . splitAt i

drop :: Int -> Block -> Block
```

```haskell
drop i = snd . splitAt i

toList :: Block -> [Bool]
toList b@(Block _ l) = [ b ! i | i <- [ 0 .. l - 1 ] ]

toListReversed :: Block -> [Bool]
toListReversed b@(Block _ l) = [ b !< i | i <- [ 0 .. l - 1 ] ]

fromList :: [Bool] -> Block
fromList = foldl' add mempty
  where
    add (Block b l) True  = Block ((b `shiftL` 1) `setBit` 0) (l + 1)
    add (Block b l) False = Block ( b `shiftL` 1            ) (l + 1)

singleton :: Bool -> Block
singleton False = Block 0 1
singleton True  = Block 1 1
```

14 hs/lib/Mandelbrot/Symbolics/ExternalAngle.hs

```haskell
{-# LANGUAGE FlexibleContexts #-}
{-# LANGUAGE TypeFamilies #-}
{- |
External angles.
-}
module Mandelbrot.Symbolics.ExternalAngle
  ( ExternalAngle(..)
  , BinaryAngle(..)
  , binaryAngle
  , binary
  , rational
  , bits
  , Tuning(..)
  , otherRep
  ) where

import Prelude hiding
  ( Rational
  , concatMap
  , splitAt
  , take
  )
import Data.Bits
  ( shiftL
  , bit
  )
import Data.Monoid
  ( (<>)
  )
import Mandelbrot.Symbolics.Block
  ( Block(Block)
  , singleton
  , concatMap
  , compact
  , splitAt
  , take
  , toList
  , toListReversed
  , fromList
  )
import Mandelbrot.Symbolics.Period
  ( Period(periods, safePeriods)
  )
import Mandelbrot.Symbolics.Rational
  ( Q(..)
  , Rational
  )

newtype ExternalAngle = ExternalAngle Rational
  deriving (Read, Show, Eq, Ord)

instance Q ExternalAngle where
  type Z ExternalAngle = Integer
  n % d = ExternalAngle (n % d)
  n %! d = ExternalAngle (n %! d)
  numerator (ExternalAngle r) = numerator r
  denominator (ExternalAngle r) = denominator r

instance Period ExternalAngle where
  periods = periods . binary
  safePeriods maxPeriod = go 0
    where
      go n r
        | n > maxPeriod = Nothing
        | even (denominator r) = go (n + 1) (double r)
        | otherwise = go' 1 (doubleOdd r)
        where
          go' m r'
            | n + m > maxPeriod = Nothing
            | r' == r = Just (n, m)
            | otherwise = go' (m + 1) (doubleOdd r')

data BinaryAngle = BinaryAngle !Block !Block
  deriving (Eq, Read, Show)

instance Ord BinaryAngle where
  compare x y
    | x == y = EQ
    | otherwise = compare (bits x) (bits y)

instance Period BinaryAngle where
  periods (BinaryAngle (Block _ pp) (Block _ p)) = (pp, p)

-- | Tuning transformation for external angles.
--   Probably only valid for angle pairs representing hyperbolic components.
class Tuning t where
  tune :: (t, t) -> t -> t

instance Tuning ExternalAngle where
  tune (lo, hi) t = rational (tune (binary lo, binary hi) (binary t))

instance Tuning BinaryAngle where
  tune (BinaryAngle _ lo, BinaryAngle _ hi) (BinaryAngle pre per)
    = binaryAngle (concatMap t pre) (concatMap t per)
    where
      t False = lo
      t True  = hi

binaryAngle :: Block -> Block -> BinaryAngle
binaryAngle pre@(Block _ pp) per@(Block _ p)
  = BinaryAngle pre' (compact (common <> per'))
  where
    match
      = length . takeWhile id
      $ zipWith (==) (toListReversed pre) (toListReversed per)
    (pre', common) = splitAt (pp - match) pre
    per' = take (p - match) per

bits :: BinaryAngle -> [Bool]
bits (BinaryAngle pre per) = toList pre ++ cycle (toList per)

binary :: ExternalAngle -> BinaryAngle
binary a0 = (\(pp, p) -> BinaryAngle (fromList pp) (b p)) . binary' . wrap $ a0
  where
    b p = if a0 == one then singleton True else fromList p
    binary' a
      | even (denominator a) =
          let (pre, per) = binary' (double a)
          in  ((a >= half) : pre, per)
      | otherwise = ([], (a >= half) : binary'' (doubleOdd a))
      where
        binary'' a'
          | a' == a = []
          | otherwise = (a' >= half) : binary'' (doubleOdd a')

rational :: BinaryAngle -> ExternalAngle
rational (BinaryAngle (Block pre pp) (Block per p))
  = ((pre `shiftL` p) - pre + per) % ((bit p - 1) `shiftL` pp)

otherRep :: BinaryAngle -> Maybe BinaryAngle
otherRep (BinaryAngle (Block pre pp) (Block 0 1)) = Just $ binaryAngle (Block (pre - 1) pp) (Block 1 1)
otherRep (BinaryAngle (Block pre pp) (Block 1 1)) = Just $ binaryAngle (Block (pre + 1) pp) (Block 0 1)
otherRep _ = Nothing
```

15 hs/lib/Mandelbrot/Symbolics.hs

```haskell
module Mandelbrot.Symbolics
  ( module Mandelbrot.Symbolics.AngledAddress
  , module Mandelbrot.Symbolics.Block
  , module Mandelbrot.Symbolics.ExternalAngle
  , module Mandelbrot.Symbolics.InternalAddress
  , module Mandelbrot.Symbolics.InternalAngle
  , module Mandelbrot.Symbolics.Kneading
  , module Mandelbrot.Symbolics.Misiurewicz
  , module Mandelbrot.Symbolics.Period
  , module Mandelbrot.Symbolics.Rational
  ) where

import Mandelbrot.Symbolics.AngledAddress
import Mandelbrot.Symbolics.Block
import Mandelbrot.Symbolics.ExternalAngle
import Mandelbrot.Symbolics.InternalAddress
import Mandelbrot.Symbolics.InternalAngle
import Mandelbrot.Symbolics.Kneading
import Mandelbrot.Symbolics.Misiurewicz
import Mandelbrot.Symbolics.Period
import Mandelbrot.Symbolics.Rational
```

16 hs/lib/Mandelbrot/Symbolics/InternalAddress.hs

```
module Mandelbrot.Symbolics.InternalAddress
  ( InternalAddress(..)
  , internalAddress
  , associated
  , upper
  , lower
  ) where

import Data.Monoid
  ( (<>)
  )

import Mandelbrot.Symbolics.Block
  ( Block(..)
  , (!)
  , compact
  , singleton
  , toList
  )
import Mandelbrot.Symbolics.Kneading
  ( Kneading(..)
  )
import Mandelbrot.Symbolics.Period
  ( Period(periods)
  )

newtype InternalAddress = InternalAddress [Int]
  deriving (Read, Show, Eq, Ord)

instance Period InternalAddress where
  periods (InternalAddress xs) = (0, last xs)

-- | Construct an 'InternalAddress' from a kneading sequence.
internalAddress :: Kneading -> Maybe InternalAddress
internalAddress (StarPeriodic   (Block _ 0))          = Just . InternalAddress $ ↙
    ↳ [1]
internalAddress (StarPeriodic v@(Block _ n)) | v ! 0 = Just . InternalAddress $ ↙
    ↳ address (n + 1) (unpack v ++ [Nothing])
internalAddress (Periodic     v@(Block _ n)) | v ! 0 = Just . InternalAddress $ ↙
    ↳ address n (unpack v)
internalAddress _ = Nothing

unpack :: Block -> [Maybe Bool]
unpack = map Just . toList

address :: Int -> [Maybe Bool] -> [Int]
address p v = takeWhile (<= p) $ address' v

address' :: [Maybe Bool] -> [Int]
address' v = address'' 1 [Just True]
  where
    address'' sk vk = sk : address'' sk' vk'
      where
        sk' = (1 +) . length . takeWhile id . zipWith (==) v . cycle $ vk
        vk' = take sk' (cycle v)


-- | A star-periodic kneading sequence's upper and lower associated
--   kneading sequences.
associated :: Kneading -> Maybe (Kneading, Kneading)
associated (StarPeriodic k@(Block _ n)) = do
  let a1 = compact $ k <> singleton False
      a2 = compact $ k <> singleton True
  InternalAddress xs <- internalAddress (Periodic a2)
  let (a, abar) = if (n + 1) `elem` xs then (a2, a1) else (a1, a2)
  return (Periodic a, Periodic abar)
associated _ = Nothing

-- | The upper associated kneading sequence.
upper :: Kneading -> Maybe Kneading
upper = fmap fst . associated

-- | The lower associated kneading sequence.
lower :: Kneading -> Maybe Kneading
lower = fmap snd . associated
```

17 hs/lib/Mandelbrot/Symbolics/InternalAngle.hs

```
{-# LANGUAGE TypeFamilies #-}
{- |
Internal angles.
-}
module Mandelbrot.Symbolics.InternalAngle
  ( InternalAngle(..)
  ) where

import Prelude hiding
  ( Rational
  )
import Mandelbrot.Symbolics.Rational
  ( Q(..)
  , Rational
  )

newtype InternalAngle = InternalAngle Rational
  deriving (Read, Show, Eq, Ord)

instance Q InternalAngle where
  type Z InternalAngle = Integer
  n % d = InternalAngle (n % d)
  n %! d = InternalAngle (n %! d)
  numerator (InternalAngle r) = numerator r
  denominator (InternalAngle r) = denominator r
```

18 hs/lib/Mandelbrot/Symbolics/Kneading.hs

```haskell
{- |
Kneading.
-}
module Mandelbrot.Symbolics.Kneading
  ( Kneading(..)
  , kneading
  , unwrap
  ) where

import Prelude hiding
  ( Rational
  )
import Data.Maybe
  ( catMaybes
  )
import Data.Monoid
  ( mempty
  )

import Mandelbrot.Symbolics.Block
  ( Block(..)
  , compact
  , toList
  , fromList
  )
import Mandelbrot.Symbolics.ExternalAngle
  ( ExternalAngle
  )
import Mandelbrot.Symbolics.Period
  ( Period(periods)
  )
import Mandelbrot.Symbolics.Rational
  ( denominator
  , wrap
  , double
  , doubleOdd
  , preimages
  , zero
  )

-- | Kneading sequences.
data Kneading
  = PrePeriodic !Block !Block
  | StarPeriodic !Block -- shorter by one bit, with implicit final Star
  | Periodic !Block
  deriving (Read, Show, Eq)

instance Period Kneading where
  periods (PrePeriodic (Block _ pp) (Block _ p)) = (pp, p)
  periods (StarPeriodic (Block _ p)) = (0, p + 1)
  periods (Periodic (Block _ p)) = (0, p)

-- | The kneading sequence for an external angle.
kneading :: ExternalAngle -> Kneading
kneading a0'
```

```
  | a0 == zero = StarPeriodic mempty
  | otherwise = case span (even . denominator . fst) . kneading' $ a0 of
      (pre, ak1@(a1,_):aks) -> case takeWhile ((a1 /=) . fst) aks of
        aks' ->
          let per = map snd $ ak1 : aks'
          in  case (null pre, last per) of
                (True, Nothing) -> StarPeriodic (fromList (catMaybes per))
                (True, _) -> Periodic (compact (fromList (catMaybes per)))
                (False, _) -> PrePeriodic (fromList (catMaybes (map snd pre))) (↙
                      ↪ compact (fromList (catMaybes per)))
      ppp -> error $ "kneading: " ++ show (a0', ppp)
  where
    a0 = wrap a0'
    (lo, hi) = preimages a0
    kneading' a
      | even (denominator a) = (a, knead a) : kneading' (double a)
      | otherwise = kneading'' a
    kneading'' a = (a, knead a) : kneading'' (doubleOdd a)
    knead a
      | a == lo          = Nothing
      | a == hi          = Nothing
      | lo < a && a < hi = Just True
      | hi < a || a < lo = Just False
      | otherwise = error $ "kneading.knead: " ++ show (a, lo, hi)

unwrap :: Kneading -> [Maybe Bool]
unwrap (PrePeriodic pre per) = map Just (toList pre) ++ cycle (map Just (toList ↙
    ↪ per))
unwrap (StarPeriodic per) = cycle (map Just (toList per) ++ [Nothing])
unwrap (Periodic per) = cycle (map Just (toList per))
```

19 hs/lib/Mandelbrot/Symbolics/Misiurewicz.hs

```
module Mandelbrot.Symbolics.Misiurewicz
  ( angleCount
  , externalAngles
  ) where

import Prelude hiding
  ( splitAt
  , take
  )
import qualified Prelude as P

import Data.List
  ( nub
  , sort
  )
import Data.Maybe
  ( mapMaybe
  )
import Data.Monoid
  ( (<>)
  , mconcat
  )

import Mandelbrot.Symbolics.AngledAddress
```

```haskell
  ( angledAddress
  , addressAngles
  )
import Mandelbrot.Symbolics.Block
  ( Block(..)
  , splitAt
  , take
  )
import Mandelbrot.Symbolics.ExternalAngle
  ( ExternalAngle(..)
  , BinaryAngle(..)
  , binaryAngle
  , rational
  , binary
  )
import Mandelbrot.Symbolics.Kneading
  ( Kneading(PrePeriodic)
  , kneading
  )
import Mandelbrot.Symbolics.Period
  ( Period(periods, period)
  )

angleCount :: ExternalAngle -> Maybe Int
angleCount r
  | pp == 0   = Just 2
  | q  == 1   = Nothing -- either 1 or 2
  | otherwise = Just q
  where
    (pp, p) = periods r
    q = p `div` period (kneading r)

externalAngles :: BinaryAngle -> [BinaryAngle]
externalAngles = rays
  where
    periodic :: Int -> BinaryAngle -> Maybe BinaryAngle
    periodic = preperiodic 0
    preperiodic :: Int -> Int -> BinaryAngle -> Maybe BinaryAngle
    preperiodic preperiod' period' (BinaryAngle pre@(Block _ pp) per@(Block _ p)) =
      let n = preperiod' + period' - pp
          k = (n + p - 1) `div` p  -- ceiling (n / p)
          (pre', per') = splitAt preperiod' (pre <> mconcat (replicate k per))
      in  check $ binaryAngle pre' (take period' per')
      where
        check t@(BinaryAngle (Block _ pp') (Block _ p'))
          | pp' == preperiod' && p' == period' = Just t
          | otherwise = Nothing
    rays :: BinaryAngle -> [BinaryAngle]
    rays t
      | pp == 0 && p == 1 = [BinaryAngle (Block 0 0) (Block 0 1), BinaryAngle (Block 0 0) (Block 1 1)]
      | pp == 0 = case fmap (map binary . sort . (\(a,b) -> [a,b])) . (addressAngles =<<) . angledAddress . rational $ t of
          Just xs -> xs
          Nothing -> []
      | pp > 0 = case kneading (rational t) of
```

```
          PrePeriodic _ (Block _ kp) -> case p `divMod` kp of
            (n, m)
              | m /= 0 -> error $ "rays Preperiodic: " ++ show (pp, p, kp, n, m, t)
              | n > 1 -> case dropWhile ((n /=) . length) . iterate (rays' n pp p) $ [t] of
                  h:_ -> h
                  [] -> error $ "Misiurewicz.rays: [] " ++ show (pp, p, kp, n, m, t)
              | n == 1 -> rays'' pp p t
              | otherwise -> error $ "Misiureiwicz.rays: n<1 " ++ show (pp, p, kp, n, m, t)
          k -> error $ "Misiurewicz.rays: !pp " ++ show (pp, p, t, k)
      | otherwise = error $ "Misiurewicz.rays: pp<0 " ++ show (pp, p, t)
      where
        (pp, p) = periods t
    rays' :: Int -> Int -> Int -> [BinaryAngle] -> [BinaryAngle]
    rays' n pp p ts
      | not (null ts)
        = map binary
        . sort
        . P.take (n `min` (length ts + 2))
        . nub
        . map rational
        . mapMaybe (preperiodic pp p)
        . concat
        . mapMaybe
            ( fmap (map binary . (\(a,b) -> [a,b]))
            . (addressAngles =<<)
            . (angledAddress =<<)
            . fmap rational
            )
        $ [ periodic m t | m <- [2 * pp + p ..], t <- ts ]
      | otherwise = error "Misiurewicz.rays': null ts"
    rays'' :: Int -> Int -> BinaryAngle -> [BinaryAngle]
    rays'' pp p t
      = map binary
      . sort
      . nub
      . map rational
      . mapMaybe (preperiodic pp p)
      . concat
      . mapMaybe
          ( fmap (map binary . (\(a,b) -> [a,b]))
          . (addressAngles =<<)
          . (angledAddress =<<)
          . fmap rational
          )
      $ [ periodic m t | m <- [2 * (pp + p) .. 3 * (pp + p)] ]
```

20 hs/lib/Mandelbrot/Symbolics/Period.hs

```
module Mandelbrot.Symbolics.Period
  ( Period(..)
  , Periods(..)
  ) where

```

```haskell
class Period t where
  preperiod :: t -> Int
  preperiod = fst . periods
  period :: t -> Int
  period = snd . periods
  periods :: t -> (Int, Int)
  periods t = (preperiod t, period t)
  safePeriods :: Int -> t -> Maybe (Int, Int)
  safePeriods _ = Just . periods

newtype Periods = Periods (Int, Int)
  deriving (Eq, Read, Show)

instance Period Periods where
  periods (Periods p) = p
```

21 hs/lib/Mandelbrot/Symbolics/Rational.hs

```haskell
{-# LANGUAGE TypeFamilies #-}
{-# LANGUAGE FlexibleContexts #-}
{-# LANGUAGE ConstrainedClassMethods #-}

{- |
Rational numbers with operations useful in Mandelbrot set symbolic algorithms.
-}
module Mandelbrot.Symbolics.Rational
  ( Q(..)
  , Ratio(..)
  , Rational
  ) where

import Prelude hiding (Rational)
import qualified Data.Ratio as Ratio

-- | Rational numbers with ruff-specific operations.
class Q r where
  {-# MINIMAL (%), numerator, denominator #-}

  type Z r

  infixl 7 %, %!

  -- | Safe constructor.  Reduces to canonical form.
  (%) :: Z r -> Z r -> r
  -- | Extract numerator.
  numerator :: r -> Z r
  -- | Extract denominator.
  denominator :: r -> Z r

  -- | Unsafe constructor.
  --   Precondition: numerator `gcd` denominator == 1 && denominator > 0
  {-# INLINE (%!) #-}
  (%!) :: Z r -> Z r -> r
  (%!) = (%)

  -- | 0.
  {-# INLINE zero #-}
```

```
zero :: Integral (Z r) => r
zero = 0 %! 1

-- | 1/2.
{-# INLINE half #-}
half :: Integral (Z r) => r
half = 1 %! 2

-- | 1.
{-# INLINE one #-}
one  :: Integral (Z r) => r
one  = 1 %! 1

-- | Convert to Prelude.Rational.
{-# INLINE fromQ #-}
fromQ :: Integral (Z r) => r -> Ratio.Rational
fromQ x = toInteger (numerator x) %! toInteger (denominator x)

-- | Convert from Prelude.Rational.
{-# INLINE toQ #-}
toQ :: Integral (Z r) => Ratio.Rational -> r
toQ x = fromInteger (Ratio.numerator x) %! fromInteger (Ratio.denominator x)

-- | Wrap into [0,1).
{-# INLINE wrap #-}
wrap :: Integral (Z r) => r -> r
wrap x = (numerator x `mod` denominator x) %! denominator x

-- | Doubling map to [0,1).
{-# INLINE doubleWrap #-}
doubleWrap :: Integral (Z r) => r -> r
doubleWrap = {-# SCC "doubleWrap" #-} double . wrap

-- | Doubling map from [0,1) to [0,1).
--   Precondition: 0 <= x && x < 1
{-# INLINE double #-}
double :: Integral (Z r) => r -> r
double x = {-# SCC "double" #-} case () of
 _| even d    -> (if n < d' then n  else n - d') %  d'
  | otherwise -> (if n' < d then n' else n' - d) %! d
  where
    d = denominator x
    d' = d `div` 2
    n = numerator x
    n' = 2 * n

-- | Doubling map from [0,1) to [0,1) for odd denominator.
--   Precondition: 0 <= x && x < 1 && odd (denominator x)
{-# INLINE doubleOdd #-}
doubleOdd :: Integral (Z r) => r -> r
doubleOdd x = {-# SCC "doubleOdd" #-} (if n' < d then n' else n' - d) %! d
  where
    d = denominator x
    n = numerator x
    n' = 2 * n

-- | Doubling map pre-images from [0,1) to [0,1)x[0,1).
```

```
  --    Precondition: 0 <= x && x < 1
  {-# INLINE preimages #-}
  preimages :: Integral (Z r) => r -> (r, r)
  preimages x = (n % d', (n + d) % d')
    where
      n = numerator x
      d = denominator x
      d' = 2 * d


instance Integral a => Q (Ratio.Ratio a) where
  {-# SPECIALIZE instance Q Ratio.Rational #-}
  type Z (Ratio.Ratio a) = a
  {-# INLINE (%) #-}
  (%) = (Ratio.%)
  {-# INLINE numerator #-}
  numerator = Ratio.numerator
  {-# INLINE denominator #-}
  denominator = Ratio.denominator


-- | Ratio data structure with exposed constructor for optimisations.
data Ratio a = !a :% !a deriving (Eq)

-- | Rational type.
type Rational = Ratio Integer

instance Integral a => Q (Ratio a) where
  {-# SPECIALIZE instance Q Rational #-}
  type Z (Ratio a) = a
  {-# INLINE (%) #-}
  x % y = reduce (x * signum y) (abs y)
    where reduce x' y' = (x' `quot` d) :% (y' `quot` d) where d = gcd x' y'
  {-# INLINE (%!) #-}
  x %! y = x :% y
  {-# INLINE numerator #-}
  numerator (x :% _) = x
  {-# INLINE denominator #-}
  denominator (_ :% y) = y

instance Integral a => Ord (Ratio a) where
  {-# SPECIALIZE instance Ord Rational #-}
  (x:%y) <= (x':%y') = x * y' <= x' * y
  (x:%y) <  (x':%y') = x * y' <  x' * y

instance (Integral a, Read a) => Read (Ratio a) where
  readsPrec p = map (\(x,y) -> (toQ x, y)) . readsPrec p

instance (Integral a, Show a) => Show (Ratio a) where
  showsPrec p = showsPrec p . fromQ
```

22 mandelbrot-symbolics.cabal

```
name:                mandelbrot-symbolics
version:             0.1.0.0
synopsis:            symbolic algorithms related to the Mandelbrot set
description:         Symbolic algorithms related to the Mandelbrot set:
```

```
                     computations on external angles, kneading sequences,
                     (angled) internal addresses, etc.
homepage:            https://code.mathr.co.uk/mandelbrot-symbolics
license:             GPL-3
license-file:        COPYING
author:              Claude Heiland-Allen
maintainer:          claude@mathr.co.uk
copyright:           (c) 2018 Claude Heiland-Allen
category:            Math
build-type:          Simple
cabal-version:       >=1.10
extra-source-files:
  README.md
  TODO.md

library
  exposed-modules:
    Mandelbrot.Symbolics,
    Mandelbrot.Symbolics.AngledAddress,
    Mandelbrot.Symbolics.Block,
    Mandelbrot.Symbolics.ExternalAngle,
    Mandelbrot.Symbolics.InternalAddress,
    Mandelbrot.Symbolics.InternalAngle,
    Mandelbrot.Symbolics.Kneading,
    Mandelbrot.Symbolics.Misiurewicz,
    Mandelbrot.Symbolics.Period,
    Mandelbrot.Symbolics.Rational
  other-extensions:
    FlexibleContexts,
    FlexibleInstances,
    ConstrainedClassMethods,
    TypeFamilies,
    TypeSynonymInstances
  build-depends:
    base   >=4.7 && <4.14,
    strict >=0.3 && <0.4
  hs-source-dirs:      hs/lib
  default-language:    Haskell2010
  ghc-options:         -Wall
  ghc-prof-options:    -prof -auto-all -caf-all

source-repository head
  type:     git
  location: https://code.mathr.co.uk/mandelbrot-symbolics.git

source-repository this
  type:     git
  location: https://code.mathr.co.uk/mandelbrot-symbolics.git
  tag:      v0.1.0.0
```

23 README.md

```
mandelbrot-symbolics
====================

Symbolic algorithms related to the Mandelbrot set: computations on external
angles, kneading sequences, (angled) internal addresses, etc.
```

24 Setup.hs

```
import Distribution.Simple
main = defaultMain
```

25 TODO.md

```
TODO
====

AngledAddress
-------------

Port from 'ExternalAngle' to 'BinaryAngle'.

Use 'bulb' instead of 'wakees' where appropriate.

Move and rename 'wakees' to 'lavaurs' in a new module 'Lavaurs'.

Add 'ConciseAddress' and operations.


ExternalAngle
-------------

Optimize 'Ord' instance for 'BinaryAngle' and make it safe for non-canonical
forms.

Add 'otherRepresentation' for ...1(0) -> ...0(1) with special case for .(0).


InternalAngle
-------------

Add 'bulb' and friends.


Misiurewicz
-----------

Add navigating by spokes.


Parse
-----

Ensure base 10 digits where appropriate.

Validation and canonicalization of results.


Rational
--------

Instances for 'Ratio': 'Num', 'Fractional', 'Real', 'RealFrac'.

```

```
Integer
-------

Optimize 'odd' and 'even' to bitops.

Optimize masking - try masking before 'shiftL' where appropriate.
```