Zero-Dimensional Regular Chains in BPAS

1 `ZeroDimensionalRegularChain` Class Description

The `ZeroDimensionalRegularChain` class handles the special case of regular chains of dimension zero (where there are the same number of equations as variables in the chain). The zero sets of zero-dimensional regular chains are therefore sets of points. Because zero-dimensional regular chains are particular kinds of regular chains, the `ZeroDimensionalRegularChain` class inherits from the `RegularChain` class. This special case allows for specialized algorithms that are more efficient than those needed for positive dimension, as well as being of mathematical interest in its own right, as we discuss briefly below. Specialized versions of `intersect` and `regularize` are provided by `ZeroDimensionalRegularChain`, which behave in precisely the same way as the arbitrary dimension routines, except that the underlying regular chain has `ZeroDimensionalRegularChain` type. The usage of these routines is illustrated below.

A zero-dimensional regular chain with polynomials in $\mathbb{K}[x_1, \ldots, x_n]$ actually encodes a tower of field extensions of the base field $\mathbb{K}$. In case that a zero-dimensional regular chain is strongly normalized, so that the initials of all polynomials in the chain are in the base field, then the regular chain defines a direct product of fields. Thus, in addition to being a special case of arbitrary dimension regular chains, the zero-dimensional case is of independent mathematical interest. We will not consider the mathematics here, except to note that for zero-dimensional regular chains T, $W(T) = V(T)$ since sets of points are Zariski-closed. For an explanation of the concept of regular chain see the `RegularChain` class description.

The most important differences between the `ZeroDimensionalRegularChain` class and the `RegularChain` class have to do with the fact that being zero-dimensional imposes a strict condition (all variables are algebraic variables) that most `RegularChain` objects do not satisfy. This impacts the constructors for the class and certain routines, such as `lower` and `upper`. The zero-dimensionality requirement is handled by not allowing the direct construction of zero-dimensional regular chains with an underlying triangular set of fixed type. To allow this would allow the creation of `ZeroDimensionalRegularChain` objects that are not actually zero-dimensional because not all the variables are algebraic. Accordingly, only variable type `ZeroDimensionalRegularChain` objects can be created directly from the constructors of the class, and each of these

constructors forces the created object to be genuinely zero-dimensional.

This leads to an important restriction on how `ZeroDimensionalRegularChain` objects are created with the class. Because the objects are required to be zero-dimensional, it is not possible to add a polynomial to a variable type `ZeroDimensionalRegularChain` that does not have exactly one new variable v, *i.e.*, v is neither in the algebraic variables list nor the transcendentals list. Accordingly, the order in which polynomials are added to the chain matters. As such, *polynomials with the least significant variables must be added first.* This is seen in the examples of the usage of `intersect` and `regularize` below.

Even though this is the strictly mathematically correct way to handle zero-dimensional regular chains, there are situations where it is convenient to treat a positive-dimensional regular chain as zero-dimensional, but only when it is meaningful to do so. This is particularly important in the algorithms of the `RegularChain` class, so that the `ZeroDimensionalRegularChain` routines can be called without forcing a copy of the entire `RegularChain` object. For this reason, the copy and move constructors that take a fixed type `RegularChain` as input allow the creation of technically positive-dimensional `ZeroDimensionalRegularChain` objects that are morally zero-dimensional (no free variables appear in the polynomials of the chain).

The other main situation where `ZeroDimensionalRegularChain` routines behave differently is for `upper` and `lower`. Here, `lower` is required to return a zero-dimensional regular chain, since lower on a zero-dimensional regular chain is always zero-dimensional. The routine `upper`, however, is required to return a regular chain, since upper on a zero-dimensional regular chain may be positive-dimensional. In addition to this, the behaviour of `lower` is different depending on whether the underlying `TriangularSet` of the `ZeroDimensionalRegularChain` is fixed or variable. If it is fixed, the `ZeroDimensionalRegularChain` returned by lower may only be morally zero-dimensional.

2 intersect

The method `intersect` of the `ZeroDimensionalRegularChain` class allows the computation of the intersection of the points of a zero-dimensional regular chain and the variety of a polynomial p. For a polynomial $p \in \mathbb{Q}[x_1, \ldots, x_n]$, encoded as an `SMQP` object `p`, and a zero-dimensional regular chain T over the ambient space defined by the variable ordering $x_1 < x_2 < \cdots < x_n$, encoded as a `ZeroDimensionalRegularChain<RN,SMQP>` object `T`, we can compute the intersection of the variety $V(p)$ and the quasi-component $W(T)$ by calling

```
vector<ZeroDimensionalRegularChain<RN,SMQP>> dec;
dec = T.intersect(p);
```

For example, if $p = xy$, which means that $x = 0$ or $y = 0$ on $V(p)$, and the zero-dimensional regular chain in $\mathbb{Q}[y, x]$ is $T = \{y^2 + yx, x^2 + x\}$, which picks out the three points $(x, y) = (0, 0), (0, -1), (-1, -1)$, then we can compute the intersection of $V(p)$ and $W(T)$ with the following code

```
ZeroDimensionalRegularChain<RN,SMQP> T;
T += SMQP("x^2+x");
T += SMQP("y^2-y*x");
vector<ZeroDimensionalRegularChain<RN,SMQP>> dec;
SMQP p("x*y");
dec = T.intersect(p);
for (auto d : dec)
   d.display();
```

which produces the output

```
 /
 | y = 0
<
 | x + 1 = 0
 \
 /
 | y = 0
<
 | x = 0
 \
```

so that the intersection is just the points $(x, y) = (0, 0), (0, -1)$, as expected.

3 regularize

The method `regularize` is a routine that will take a polynomial p and a regular chain T and decompose T into regular chains of two types: components on which p is regular modulo $\mathrm{sat}(T)$, the regular case; and components on which p is zero modulo $\mathrm{sat}(T)$, the singular case. The return type of `regularize` is a `vector` of `PolyChainPair<PolyType,RegularChainType>` objects. If `A` is a `PolyChainPair` object, then we can access the polynomial as `A.poly` and the regular chain as `A.chain`. For the singular components returned by `regularize`, `A.poly` is zero, and for the regular components it is non-zero.

For example, reconsidering the example for `intersect` above, suppose that $T = \{y^2 + yx, x^2 + x\}$ and suppose that we want to determine the regular and singular components of T for $p = xy$. Then we can do so with the following code:

```
vector<Symbol> R = {'x','y','z'};
vector<PolyChainPair<SMQP,ZeroDimensionalRegularChain<RN,SMQP>>>
   components;
components = T.regularize(p);
cout << "Regular Components" << endl;
for (auto c : components) {
   if (!c.poly.isZero())
      c.chain.display();
```

```
}
cout << "Singular Components" << endl;
for (auto c : components) {
   if (c.poly.isZero())
      c.chain.display();
}
```

which produces the output

```
Regular Components
 /
 | y + 1 = 0
<
 | x + 1 = 0
 \
Singular Components
 /
 | y = 0
<
 | x + 1 = 0
 \
 /
 | y = 0
<
 | x = 0
 \
```

Thus, the regular components are those where $p \neq 0$ but the polynomials of T are zero (but the initials are nonzero), *i.e.*, where p is regular modulo $\mathrm{sat}(T)$, which is just $(x, y) = (-1, -1)$ in this case. The singular components are then those where both $p = xy = 0$ and the polynomials of T are zero (but the initials are nonzero), *i.e.*, the intersection of $V(p)$ and $W(T)$. Since the singular components are always the intersection of $V(p)$ and $W(T)$ in dimension zero, `regularize` is actually called by `intersect` to compute intersections.