clipping

Release 5.1.2

Azat Ibrakov

Sep 04, 2022

CONTENTS

Note: If object is not listed in documentation it should be considered as implementation detail that can change and should not be relied upon.

Boolean operations on geometries in the plane.

Based on algorithm by F. Martinez et al.

Reference: https://doi.org/10.1016/j.advengsoft.2013.04.004 http://www4.ujaen.es/~fmartin/bool_op.html

CHAPTER

ONE

GLOSSARY

Region — contour with points that lie within it.

Multiregion — sequence of two or more regions such that intersection of distinct regions is a discrete points set.

`clipping.planar.`**`intersect_segments`**(*first: ground.hints.Segment*, *second: ground.hints.Segment*, ***, *context: Optional[ground.base.Context] = None*) → Union[ground.hints.Empty, ground.hints.Multipoint, ground.hints.Segment]

Returns intersection of segments.

Time complexity: `O(1)`

Memory complexity: `O(1)`

Parameters

- **`first`** – first operand.
- **`second`** – second operand.
- **`context`** – geometric context.

Returns intersection of operands.

```
>>> from ground.base import get_context
>>> context = get_context()
>>> EMPTY = context.empty
>>> Multipoint = context.multipoint_cls
>>> Point = context.point_cls
>>> Segment = context.segment_cls
>>> (intersect_segments(Segment(Point(0, 0), Point(4, 0)),
...                     Segment(Point(6, 0), Point(10, 0)))
...  is EMPTY)
True
>>> (intersect_segments(Segment(Point(0, 0), Point(4, 0)),
...                     Segment(Point(4, 0), Point(8, 0)))
...  == Multipoint([Point(4, 0)]))
True
>>> (intersect_segments(Segment(Point(0, 0), Point(4, 0)),
...                     Segment(Point(2, 0), Point(6, 0)))
...  == Segment(Point(2, 0), Point(4, 0)))
True
```

`clipping.planar.subtract_segments`(*minuend: ground.hints.Segment*, *subtrahend: ground.hints.Segment*, ***, *context: Optional[ground.base.Context] = None*) → Union[ground.hints.Empty, ground.hints.Multisegment, ground.hints.Segment]

Returns difference of segments.

Time complexity: `O(1)`

Memory complexity: `O(1)`

Parameters

- **minuend** – segment to subtract from.
- **subtrahend** – segment to subtract.
- **context** – geometric context.

Returns difference of minuend with subtrahend.

```
>>> from ground.base import get_context
>>> context = get_context()
>>> EMPTY = context.empty
>>> Multisegment = context.multisegment_cls
>>> Point = context.point_cls
>>> Segment = context.segment_cls
>>> (subtract_segments(Segment(Point(0, 0), Point(4, 0)),
...                    Segment(Point(0, 0), Point(4, 0)))
...  is subtract_segments(Segment(Point(0, 0), Point(4, 0)),
...                       Segment(Point(0, 0), Point(6, 0)))
...  is EMPTY)
True
>>> (subtract_segments(Segment(Point(0, 0), Point(4, 0)),
...                    Segment(Point(2, 0), Point(6, 0)))
...  == Segment(Point(0, 0), Point(2, 0)))
True
>>> (subtract_segments(Segment(Point(0, 0), Point(4, 0)),
...                    Segment(Point(6, 0), Point(10, 0)))
...  == subtract_segments(Segment(Point(0, 0), Point(4, 0)),
...                       Segment(Point(4, 0), Point(8, 0)))
...  == Segment(Point(0, 0), Point(4, 0)))
True
>>> (subtract_segments(Segment(Point(0, 0), Point(4, 0)),
...                    Segment(Point(1, 0), Point(3, 0)))
...  == Multisegment([Segment(Point(0, 0), Point(1, 0)),
...                   Segment(Point(3, 0), Point(4, 0))]))
True
```

`clipping.planar.symmetric_subtract_segments`(*first: ground.hints.Segment*, *second: ground.hints.Segment*, ***, *context: Optional[ground.base.Context] = None*) → Union[ground.hints.Empty, ground.hints.Multisegment, ground.hints.Segment]

Returns symmetric difference of segments.

Time complexity: `O(1)`

Memory complexity: `O(1)`

Parameters

- **first** – first operand.
- **second** – second operand.
- **context** – geometric context.

Returns symmetric difference of operands.

```
>>> from ground.base import get_context
>>> context = get_context()
>>> EMPTY = context.empty
>>> Multisegment = context.multisegment_cls
>>> Point = context.point_cls
>>> Segment = context.segment_cls
>>> (symmetric_subtract_segments(Segment(Point(0, 0), Point(4, 0)),
...                              Segment(Point(0, 0), Point(4, 0)))
...  is EMPTY)
True
>>> (symmetric_subtract_segments(Segment(Point(0, 0), Point(4, 0)),
...                              Segment(Point(4, 0), Point(8, 0)))
...  == Segment(Point(0, 0), Point(8, 0)))
True
>>> (symmetric_subtract_segments(Segment(Point(0, 0), Point(4, 0)),
...                              Segment(Point(1, 0), Point(3, 0)))
...  == Multisegment([Segment(Point(0, 0), Point(1, 0)),
...                   Segment(Point(3, 0), Point(4, 0))]))
True
>>> (symmetric_subtract_segments(Segment(Point(0, 0), Point(4, 0)),
...                              Segment(Point(2, 0), Point(6, 0)))
...  == Multisegment([Segment(Point(0, 0), Point(2, 0)),
...                   Segment(Point(4, 0), Point(6, 0))]))
True
>>> (symmetric_subtract_segments(Segment(Point(0, 0), Point(4, 0)),
...                              Segment(Point(6, 0), Point(10, 0)))
...  == Multisegment([Segment(Point(0, 0), Point(4, 0)),
...                   Segment(Point(6, 0), Point(10, 0))]))
True
```

`clipping.planar.`**`unite_segments`**(*first: ground.hints.Segment*, *second: ground.hints.Segment*, *, *context: Optional[ground.base.Context] = None*) → Union[ground.hints.Multisegment, ground.hints.Segment]

Returns union of segments.

Time complexity: `O(1)`

Memory complexity: `O(1)`

Parameters

- **first** – first operand.
- **second** – second operand.
- **context** – geometric context.

Returns union of operands.

```
>>> from ground.base import get_context
>>> context = get_context()
>>> EMPTY = context.empty
>>> Multisegment = context.multisegment_cls
>>> Point = context.point_cls
>>> Segment = context.segment_cls
>>> (unite_segments(Segment(Point(0, 0), Point(4, 0)),
...                 Segment(Point(0, 0), Point(4, 0)))
...  == unite_segments(Segment(Point(0, 0), Point(4, 0)),
...                    Segment(Point(1, 0), Point(3, 0)))
...  == Segment(Point(0, 0), Point(4, 0)))
True
>>> (unite_segments(Segment(Point(0, 0), Point(4, 0)),
...                 Segment(Point(4, 0), Point(8, 0)))
...  == Segment(Point(0, 0), Point(8, 0)))
True
>>> (unite_segments(Segment(Point(0, 0), Point(4, 0)),
...                 Segment(Point(2, 0), Point(6, 0)))
...  == Segment(Point(0, 0), Point(6, 0)))
True
>>> (unite_segments(Segment(Point(0, 0), Point(4, 0)),
...                 Segment(Point(6, 0), Point(10, 0)))
...  == Multisegment([Segment(Point(0, 0), Point(4, 0)),
...                   Segment(Point(6, 0), Point(10, 0))]))
True
```

`clipping.planar.`**`complete_intersect_segment_with_multisegment`**(*segment: ground.hints.Segment*, *multisegment: ground.hints.Multisegment*, *, *context: Optional[ground.base.Context] = None*) → Union[ground.hints.Empty, ground.hints.Mix, ground.hints.Multipoint, ground.hints.Multisegment, ground.hints.Segment]

Returns intersection of segment with multisegment considering cases with geometries touching each other in points.

Time complexity: `O(segments_count * log segments_count)`

Memory complexity: `O(segments_count)`

where `segments_count = segments_count + intersections_count`, `segments_count = len(subtrahend.segments) + 1`, `intersections_count` — number of intersections between segments.

Parameters

- **`segment`** – first operand.
- **`multisegment`** – second operand.
- **`context`** – geometric context.

Returns intersection of operands.

```
>>> from ground.base import get_context
>>> context = get_context()
>>> EMPTY = context.empty
>>> Mix = context.mix_cls
>>> Multipoint = context.multipoint_cls
>>> Multisegment = context.multisegment_cls
>>> Point = context.point_cls
>>> Segment = context.segment_cls
>>> (complete_intersect_segment_with_multisegment(
...      Segment(Point(0, 0), Point(4, 0)),
...      Multisegment([Segment(Point(6, 0), Point(10, 0)),
...                    Segment(Point(6, 0), Point(6, 4))]))
...  is EMPTY)
True
>>> (complete_intersect_segment_with_multisegment(
...      Segment(Point(0, 0), Point(4, 0)),
...      Multisegment([Segment(Point(4, 0), Point(8, 0)),
...                    Segment(Point(4, 0), Point(4, 4))]))
...  == Multipoint([Point(4, 0)]))
True
>>> (complete_intersect_segment_with_multisegment(
...      Segment(Point(0, 0), Point(4, 0)),
...      Multisegment([Segment(Point(0, 0), Point(4, 0)),
...                    Segment(Point(0, 0), Point(0, 4))]))
...  == Segment(Point(0, 0), Point(4, 0)))
True
>>> (complete_intersect_segment_with_multisegment(
...      Segment(Point(0, 0), Point(4, 0)),
...      Multisegment([Segment(Point(0, 0), Point(1, 0)),
...                    Segment(Point(3, 0), Point(4, 0))]))
...  == Multisegment([Segment(Point(0, 0), Point(1, 0)),
...                   Segment(Point(3, 0), Point(4, 0))]))
True
>>> (complete_intersect_segment_with_multisegment(
...      Segment(Point(0, 0), Point(4, 0)),
...      Multisegment([Segment(Point(0, 0), Point(1, 0)),
...                    Segment(Point(2, 0), Point(2, 1)),
...                    Segment(Point(3, 0), Point(4, 0))]))
...  == Mix(Multipoint([Point(2, 0)]),
...         Multisegment([Segment(Point(0, 0), Point(1, 0)),
...                       Segment(Point(3, 0), Point(4, 0))]), EMPTY))
True
```

clipping.planar.**intersect_segment_with_multisegment**(*segment: ground.hints.Segment*, *multisegment: ground.hints.Multisegment*, ***, *context: Optional[ground.base.Context] = None*) → Union[ground.hints.Empty, ground.hints.Multisegment, ground.hints.Segment]

Returns intersection of segments.

Time complexity: `O(len(multisegment.segments))`

Memory complexity: `O(len(multisegment.segments))`

Parameters

- **segment** – first operand.
- **multisegment** – second operand.
- **context** – geometric context.

Returns intersection of operands.

```
>>> from ground.base import get_context
>>> context = get_context()
>>> EMPTY = context.empty
>>> Multisegment = context.multisegment_cls
>>> Point = context.point_cls
>>> Segment = context.segment_cls
>>> (intersect_segment_with_multisegment(
...      Segment(Point(0, 0), Point(4, 0)),
...      Multisegment([Segment(Point(6, 0), Point(10, 0)),
...                    Segment(Point(6, 0), Point(6, 4))]))
...  is intersect_segment_with_multisegment(
...          Segment(Point(0, 0), Point(4, 0)),
...          Multisegment([Segment(Point(4, 0), Point(8, 0)),
...                        Segment(Point(4, 0), Point(4, 4))]))
... is EMPTY)
True
>>> (intersect_segment_with_multisegment(
...      Segment(Point(0, 0), Point(4, 0)),
...      Multisegment([Segment(Point(0, 0), Point(4, 0)),
...                    Segment(Point(0, 0), Point(0, 4))]))
...  == Segment(Point(0, 0), Point(4, 0)))
True
>>> (intersect_segment_with_multisegment(
...      Segment(Point(0, 0), Point(4, 0)),
...      Multisegment([Segment(Point(0, 0), Point(1, 0)),
...                    Segment(Point(3, 0), Point(4, 0))]))
...  == Multisegment([Segment(Point(0, 0), Point(1, 0)),
...                   Segment(Point(3, 0), Point(4, 0))]))
True
>>> (intersect_segment_with_multisegment(
...      Segment(Point(0, 0), Point(4, 0)),
...      Multisegment([Segment(Point(0, 0), Point(1, 0)),
...                    Segment(Point(2, 0), Point(2, 1)),
...                    Segment(Point(3, 0), Point(4, 0))]))
...  == Multisegment([Segment(Point(0, 0), Point(1, 0)),
...                   Segment(Point(3, 0), Point(4, 0))]))
True
```

clipping.planar.**subtract_multisegment_from_segment**(*minuend: ground.hints.Segment*, *subtrahend: ground.hints.Multisegment*, ***, *context: Optional[ground.base.Context] = None*) → Union[ground.hints.Empty, ground.hints.Multisegment, ground.hints.Segment]

Returns difference of segment with multisegment.

Time complexity: `O(segments_count * log segments_count)`

Memory complexity: `O(segments_count)`

where `segments_count = segments_count + intersections_count`, `segments_count = len(subtrahend.segments) + 1`, `intersections_count` — number of intersections between segments.

Parameters

- **minuend** – segment to subtract from.
- **subtrahend** – multisegment to subtract.
- **context** – geometric context.

Returns difference of minuend with subtrahend.

```
>>> from ground.base import get_context
>>> context = get_context()
>>> EMPTY = context.empty
>>> Multisegment = context.multisegment_cls
>>> Point = context.point_cls
>>> Segment = context.segment_cls
>>> (subtract_multisegment_from_segment(
...      Segment(Point(0, 0), Point(4, 0)),
...      Multisegment([Segment(Point(0, 0), Point(4, 0)),
...                    Segment(Point(0, 1), Point(0, 3))]))
...  is subtract_multisegment_from_segment(
...         Segment(Point(0, 0), Point(4, 0)),
...         Multisegment([Segment(Point(0, 0), Point(6, 0)),
...                       Segment(Point(0, 1), Point(0, 3))]))
...  is EMPTY)
True
>>> (subtract_multisegment_from_segment(
...      Segment(Point(0, 0), Point(4, 0)),
...      Multisegment([Segment(Point(2, 0), Point(4, 0)),
...                    Segment(Point(0, 1), Point(0, 3))]))
...  == Segment(Point(0, 0), Point(2, 0)))
True
>>> (subtract_multisegment_from_segment(
...      Segment(Point(0, 0), Point(4, 0)),
...      Multisegment([Segment(Point(3, 0), Point(4, 0)),
...                    Segment(Point(0, 0), Point(1, 0))]))
...  == Segment(Point(1, 0), Point(3, 0)))
True
>>> (subtract_multisegment_from_segment(
...      Segment(Point(0, 0), Point(4, 0)),
...      Multisegment([Segment(Point(6, 0), Point(10, 0)),
...                    Segment(Point(0, 1), Point(0, 3))]))
...  == subtract_multisegment_from_segment(
...         Segment(Point(0, 0), Point(4, 0)),
...         Multisegment([Segment(Point(4, 0), Point(8, 0)),
...                       Segment(Point(0, 1), Point(0, 3))]))
...  == Segment(Point(0, 0), Point(4, 0)))
True
>>> (subtract_multisegment_from_segment(
```

(continues on next page)

(continued from previous page)

```
...     Segment(Point(0, 0), Point(4, 0)),
...     Multisegment([Segment(Point(1, 0), Point(3, 0)),
...                   Segment(Point(0, 1), Point(0, 3))]))
...  == Multisegment([Segment(Point(0, 0), Point(1, 0)),
...                   Segment(Point(3, 0), Point(4, 0))]))
True
```

clipping.planar.**subtract_segment_from_multisegment**(*minuend: ground.hints.Multisegment*, *subtrahend: ground.hints.Segment*, ***, *context: Optional[ground.base.Context] = None*) → Union[ground.hints.Empty, ground.hints.Multisegment, ground.hints.Segment]

Returns difference of segment with multisegment.

Time complexity: `O(len(subtrahend.segments))`

Memory complexity: `O(len(subtrahend.segments))`

Parameters

- **minuend** – multisegment to subtract from.
- **subtrahend** – segment to subtract.
- **context** – geometric context.

Returns difference of minuend with subtrahend.

```
>>> from ground.base import get_context
>>> context = get_context()
>>> EMPTY = context.empty
>>> Multisegment = context.multisegment_cls
>>> Point = context.point_cls
>>> Segment = context.segment_cls
>>> (subtract_segment_from_multisegment(
...     Multisegment([Segment(Point(0, 0), Point(1, 0)),
...                   Segment(Point(3, 0), Point(4, 0))]),
...     Segment(Point(0, 0), Point(4, 0)))
...  is subtract_segment_from_multisegment(
...         Multisegment([Segment(Point(0, 0), Point(1, 0)),
...                       Segment(Point(3, 0), Point(4, 0))]),
...         Segment(Point(0, 0), Point(6, 0)))
...  is EMPTY)
True
>>> (subtract_segment_from_multisegment(
...     Multisegment([Segment(Point(0, 0), Point(2, 0)),
...                   Segment(Point(0, 1), Point(0, 3))]),
...     Segment(Point(0, 1), Point(0, 3)))
...  == subtract_segment_from_multisegment(
...         Multisegment([Segment(Point(0, 0), Point(2, 0)),
...                       Segment(Point(3, 0), Point(4, 0))]),
...         Segment(Point(2, 0), Point(4, 0)))
...  == Segment(Point(0, 0), Point(2, 0)))
```

(continues on next page)

(continued from previous page)

```
True
>>> (subtract_segment_from_multisegment(
...      Multisegment([Segment(Point(0, 0), Point(1, 0)),
...                    Segment(Point(3, 0), Point(4, 0))]),
...      Segment(Point(1, 0), Point(3, 0)))
...  == Multisegment([Segment(Point(0, 0), Point(1, 0)),
...                   Segment(Point(3, 0), Point(4, 0))]))
True
>>> (subtract_segment_from_multisegment(
...      Multisegment([Segment(Point(0, 0), Point(4, 0)),
...                    Segment(Point(0, 1), Point(0, 3))]),
...      Segment(Point(1, 0), Point(3, 0)))
...  == Multisegment([Segment(Point(0, 0), Point(1, 0)),
...                   Segment(Point(3, 0), Point(4, 0)),
...                   Segment(Point(0, 1), Point(0, 3))]))
True
```

clipping.planar.**symmetric_subtract_multisegment_from_segment**(*first: ground.hints.Segment*, *second: ground.hints.Multisegment*, ***, *context: Optional[ground.base.Context] = None*) → Union[ground.hints.Empty, ground.hints.Multisegment, ground.hints.Segment]

Returns symmetric difference of segment and multisegment.

Time complexity: `O(segments_count * log segments_count)`

Memory complexity: `O(segments_count)`

where `segments_count = segments_count + intersections_count`, `segments_count = len(second.segments) + 1`, `intersections_count` — number of intersections between segments.

Parameters

- **first** – first operand.
- **second** – second operand.
- **context** – geometric context.

Returns symmetric difference of operands.

```
>>> from ground.base import get_context
>>> context = get_context()
>>> EMPTY = context.empty
>>> Multisegment = context.multisegment_cls
>>> Point = context.point_cls
>>> Segment = context.segment_cls
>>> (symmetric_subtract_multisegment_from_segment(
...      Segment(Point(0, 0), Point(4, 0)),
...      Multisegment([Segment(Point(0, 0), Point(2, 0)),
...                    Segment(Point(2, 0), Point(4, 0))]))
...  is EMPTY)
True
```

(continues on next page)

(continued from previous page)

```
>>> (symmetric_subtract_multisegment_from_segment(
...      Segment(Point(0, 0), Point(4, 0)),
...      Multisegment([Segment(Point(0, 0), Point(4, 0)),
...                    Segment(Point(0, 1), Point(0, 3))]))
...  == Segment(Point(0, 1), Point(0, 3)))
True
>>> (symmetric_subtract_multisegment_from_segment(
...      Segment(Point(0, 0), Point(4, 0)),
...      Multisegment([Segment(Point(0, 0), Point(1, 0)),
...                    Segment(Point(3, 0), Point(4, 0))]))
...  == Segment(Point(1, 0), Point(3, 0)))
True
>>> (symmetric_subtract_multisegment_from_segment(
...      Segment(Point(0, 0), Point(4, 0)),
...      Multisegment([Segment(Point(4, 0), Point(8, 0)),
...                    Segment(Point(0, 1), Point(0, 3))]))
...  == Multisegment([Segment(Point(0, 0), Point(4, 0)),
...                   Segment(Point(0, 1), Point(0, 3)),
...                   Segment(Point(4, 0), Point(8, 0))]))
True
>>> (symmetric_subtract_multisegment_from_segment(
...      Segment(Point(0, 0), Point(4, 0)),
...      Multisegment([Segment(Point(1, 0), Point(3, 0)),
...                    Segment(Point(0, 1), Point(0, 3))]))
...  == Multisegment([Segment(Point(0, 0), Point(1, 0)),
...                   Segment(Point(0, 1), Point(0, 3)),
...                   Segment(Point(3, 0), Point(4, 0))]))
True
```

clipping.planar.**unite_segment_with_multisegment**(*first: ground.hints.Segment*, *second: ground.hints.Multisegment*, ***, *context: Optional[ground.base.Context] = None*) → Union[ground.hints.Multisegment, ground.hints.Segment]

Returns symmetric difference of segment and multisegment.

Time complexity: `O(len(second.segments))`

Memory complexity: `O(len(second.segments))`

Parameters

- **first** – first operand.
- **second** – second operand.
- **context** – geometric context.

Returns symmetric difference of operands.

```
>>> from ground.base import get_context
>>> context = get_context()
>>> EMPTY = context.empty
>>> Multisegment = context.multisegment_cls
```

(continues on next page)

(continued from previous page)

```
>>> Point = context.point_cls
>>> Segment = context.segment_cls
>>> (unite_segment_with_multisegment(
...      Segment(Point(0, 0), Point(4, 0)),
...      Multisegment([Segment(Point(0, 0), Point(2, 0)),
...                    Segment(Point(2, 0), Point(4, 0))]))
...  == unite_segment_with_multisegment(
...          Segment(Point(0, 0), Point(4, 0)),
...          Multisegment([Segment(Point(0, 0), Point(1, 0)),
...                        Segment(Point(3, 0), Point(4, 0))]))
...  == Segment(Point(0, 0), Point(4, 0)))
True
>>> (unite_segment_with_multisegment(
...      Segment(Point(0, 0), Point(4, 0)),
...      Multisegment([Segment(Point(0, 0), Point(4, 0)),
...                    Segment(Point(0, 1), Point(0, 3))]))
...  == Multisegment([Segment(Point(0, 1), Point(0, 3)),
...                   Segment(Point(0, 0), Point(4, 0))]))
True
>>> (unite_segment_with_multisegment(
...      Segment(Point(0, 0), Point(4, 0)),
...      Multisegment([Segment(Point(4, 0), Point(8, 0)),
...                    Segment(Point(0, 1), Point(0, 3))]))
...  == Multisegment([Segment(Point(4, 0), Point(8, 0)),
...                   Segment(Point(0, 1), Point(0, 3)),
...                   Segment(Point(0, 0), Point(4, 0))]))
True
>>> (unite_segment_with_multisegment(
...      Segment(Point(1, 0), Point(3, 0)),
...      Multisegment([Segment(Point(0, 0), Point(4, 0)),
...                    Segment(Point(0, 1), Point(0, 3))]))
...  == Multisegment([Segment(Point(0, 0), Point(1, 0)),
...                   Segment(Point(3, 0), Point(4, 0)),
...                   Segment(Point(0, 1), Point(0, 3)),
...                   Segment(Point(1, 0), Point(3, 0))]))
True
```

clipping.planar.**complete_intersect_segment_with_polygon**(*segment: ground.hints.Segment*, *polygon: ground.hints.Polygon*, ***, *context: Optional[ground.base.Context] = None*) → Union[ground.hints.Empty, ground.hints.Mix, ground.hints.Multipoint, ground.hints.Multisegment, ground.hints.Segment]

Returns intersection of segment with polygon considering cases with geometries touching each other in points.

Time complexity: `O(segments_count * log segments_count)`

Memory complexity: `O(segments_count)`

where `segments_count = start_segments_count + intersections_count`, `start_segments_count = polygon_edges_count + 1`, `polygon_edges_count = len(polygon.border.vertices) + sum(len(hole.vertices) for hole in polygon.holes)`, `intersections_count` — number of intersections between segment and polygon edges.

Parameters

- **segment** – first operand.
- **polygon** – second operand.
- **context** – geometric context.

Returns intersection of operands.

```
>>> from ground.base import get_context
>>> context = get_context()
>>> EMPTY = context.empty
>>> Contour = context.contour_cls
>>> Mix = context.mix_cls
>>> Multipoint = context.multipoint_cls
>>> Multisegment = context.multisegment_cls
>>> Point = context.point_cls
>>> Polygon = context.polygon_cls
>>> Segment = context.segment_cls
>>> square = Contour([Point(0, 0), Point(4, 0), Point(4, 4),
...                   Point(0, 4)])
>>> inner_square = Contour([Point(1, 1), Point(3, 1), Point(3, 3),
...                         Point(1, 3)])
>>> clockwise_inner_square = Contour([Point(1, 1), Point(1, 3),
...                                   Point(3, 3), Point(3, 1)])
>>> (complete_intersect_segment_with_polygon(
...      Segment(Point(0, 0), Point(1, 0)), Polygon(inner_square, []))
...  is EMPTY)
True
>>> (complete_intersect_segment_with_polygon(
...      Segment(Point(0, 0), Point(1, 1)), Polygon(inner_square, []))
...  == Multipoint([Point(1, 1)]))
True
>>> (complete_intersect_segment_with_polygon(
...      Segment(Point(0, 0), Point(1, 1)), Polygon(square, []))
...  == Segment(Point(0, 0), Point(1, 1)))
True
>>> (complete_intersect_segment_with_polygon(
...      Segment(Point(0, 0), Point(4, 4)),
...      Polygon(square, [clockwise_inner_square]))
...  == Multisegment([Segment(Point(0, 0), Point(1, 1)),
...                   Segment(Point(3, 3), Point(4, 4))]))
True
>>> (complete_intersect_segment_with_polygon(
...      Segment(Point(1, 1), Point(4, 4)),
...      Polygon(square, [clockwise_inner_square]))
...  == Mix(Multipoint([Point(1, 1)]), Segment(Point(3, 3), Point(4, 4)),
...         EMPTY))
True
```

clipping.planar.**intersect_segment_with_polygon**(*segment: ground.hints.Segment*, *polygon: ground.hints.Polygon*, ***, *context: Optional[ground.base.Context] = None*) → Union[ground.hints.Empty, ground.hints.Multisegment, ground.hints.Segment]

Returns intersection of segment with polygon.

Time complexity: `O(segments_count * log segments_count)`

Memory complexity: `O(segments_count)`

where `segments_count = start_segments_count + intersections_count`, `start_segments_count = polygon_edges_count + 1`, `polygon_edges_count = len(polygon.border.vertices) + sum(len(hole.vertices) for hole in polygon.holes)`, `intersections_count` — number of intersections between segment and polygon edges.

Parameters

- **segment** – first operand.
- **polygon** – second operand.
- **context** – geometric context.

Returns intersection of operands.

```
>>> from ground.base import get_context
>>> context = get_context()
>>> EMPTY = context.empty
>>> Contour = context.contour_cls
>>> Multisegment = context.multisegment_cls
>>> Point = context.point_cls
>>> Polygon = context.polygon_cls
>>> Segment = context.segment_cls
>>> square = Contour([Point(0, 0), Point(4, 0), Point(4, 4),
...                   Point(0, 4)])
>>> inner_square = Contour([Point(1, 1), Point(3, 1), Point(3, 3),
...                         Point(1, 3)])
>>> clockwise_inner_square = Contour([Point(1, 1), Point(1, 3),
...                                   Point(3, 3), Point(3, 1)])
>>> (intersect_segment_with_polygon(Segment(Point(0, 0), Point(1, 0)),
...                                 Polygon(inner_square, []))
...  is intersect_segment_with_polygon(Segment(Point(0, 0), Point(1, 1)),
...                                    Polygon(inner_square, []))
...  is EMPTY)
True
>>> (intersect_segment_with_polygon(Segment(Point(0, 0), Point(1, 1)),
...                                 Polygon(square, []))
...  == Segment(Point(0, 0), Point(1, 1)))
True
>>> (intersect_segment_with_polygon(Segment(Point(1, 1), Point(4, 4)),
...                                 Polygon(square,
...                                         [clockwise_inner_square]))
...  == Segment(Point(3, 3), Point(4, 4)))
True
>>> (intersect_segment_with_polygon(Segment(Point(0, 0), Point(4, 4)),
...                                 Polygon(square,
...                                         [clockwise_inner_square]))
...  == Multisegment([Segment(Point(0, 0), Point(1, 1)),
...                   Segment(Point(3, 3), Point(4, 4))]))
True
```

`clipping.planar.subtract_polygon_from_segment`(*minuend: ground.hints.Segment*, *subtrahend: ground.hints.Polygon*, ***, *context: Optional[ground.base.Context] = None*) → Union[ground.hints.Empty, ground.hints.Multisegment, ground.hints.Segment]

Returns difference of segment with polygon.

Time complexity: `O(segments_count * log segments_count)`

Memory complexity: `O(segments_count)`

where `segments_count = start_segments_count + intersections_count`, `start_segments_count = subtrahend_edges_count + 1`, `subtrahend_edges_count = sum(len(polygon.border.vertices) + sum(len(hole.vertices) for hole in polygon.holes) for polygon in subtrahend.polygons)`, `intersections_count` — number of intersections between segment and polygon edges.

Parameters

- **minuend** – segment to subtract from.
- **subtrahend** – polygon to subtract.
- **context** – geometric context.

Returns difference of minuend with subtrahend.

```
>>> from ground.base import get_context
>>> context = get_context()
>>> EMPTY = context.empty
>>> Contour = context.contour_cls
>>> Multisegment = context.multisegment_cls
>>> Point = context.point_cls
>>> Polygon = context.polygon_cls
>>> Segment = context.segment_cls
>>> square = Contour([Point(0, 0), Point(4, 0), Point(4, 4), Point(0, 4)])
>>> inner_square = Contour([Point(1, 1), Point(3, 1), Point(3, 3),
...                         Point(1, 3)])
>>> clockwise_inner_square = Contour([Point(1, 1), Point(1, 3),
...                                   Point(3, 3), Point(3, 1)])
>>> (subtract_polygon_from_segment(Segment(Point(0, 0), Point(1, 1)),
...                                Polygon(square, []))
...  is EMPTY)
True
>>> (subtract_polygon_from_segment(Segment(Point(0, 0), Point(1, 0)),
...                                Polygon(inner_square, []))
...  == Segment(Point(0, 0), Point(1, 0)))
True
>>> (subtract_polygon_from_segment(Segment(Point(0, 0), Point(1, 1)),
...                                Polygon(inner_square, []))
...  == Segment(Point(0, 0), Point(1, 1)))
True
>>> (subtract_polygon_from_segment(Segment(Point(1, 1), Point(4, 4)),
...                                Polygon(square,
...                                        [clockwise_inner_square]))
...  == subtract_polygon_from_segment(Segment(Point(0, 0), Point(4, 4)),
...                                   Polygon(square,
```

(continues on next page)

(continued from previous page)

```
...                                            [clockwise_inner_square]))
...  == Segment(Point(1, 1), Point(3, 3)))
True
>>> (subtract_polygon_from_segment(Segment(Point(0, 0), Point(4, 4)),
...                               Polygon(inner_square, []))
...  == Multisegment([Segment(Point(0, 0), Point(1, 1)),
...                   Segment(Point(3, 3), Point(4, 4))]))
True
```

clipping.planar.**symmetric_subtract_polygon_from_segment**(*segment: ground.hints.Segment*, *polygon: ground.hints.Polygon*, ***, *context: Optional[ground.base.Context] = None*) → Union[ground.hints.Mix, ground.hints.Polygon]

Returns symmetric difference of segment with polygon.

Time complexity: `O(segments_count * log segments_count)`

Memory complexity: `O(segments_count)`

where `segments_count = start_segments_count + intersections_count`, `start_segments_count = polygon_edges_count + 1`, `polygon_edges_count = len(polygon.border.vertices) + sum(len(hole.vertices) for hole in polygon.holes)`, `intersections_count` — number of intersections between segment and polygon edges.

Parameters

- **segment** – first operand.
- **polygon** – second operand.
- **context** – geometric context.

Returns symmetric difference of operands.

```
>>> from ground.base import get_context
>>> context = get_context()
>>> EMPTY = context.empty
>>> Contour = context.contour_cls
>>> Mix = context.mix_cls
>>> Multisegment = context.multisegment_cls
>>> Point = context.point_cls
>>> Polygon = context.polygon_cls
>>> Segment = context.segment_cls
>>> square = Contour([Point(0, 0), Point(4, 0), Point(4, 4),
...                   Point(0, 4)])
>>> inner_square = Contour([Point(1, 1), Point(3, 1), Point(3, 3),
...                         Point(1, 3)])
>>> clockwise_inner_square = Contour([Point(1, 1), Point(1, 3),
...                                   Point(3, 3), Point(3, 1)])
>>> (symmetric_subtract_polygon_from_segment(
...      Segment(Point(0, 0), Point(1, 1)), Polygon(square, []))
...  == Polygon(square, []))
True
>>> (symmetric_subtract_polygon_from_segment(
...      Segment(Point(0, 0), Point(1, 1)), Polygon(inner_square, []))
```

(continues on next page)

(continued from previous page)

```
...  == Mix(EMPTY, Segment(Point(0, 0), Point(1, 1)),
...         Polygon(inner_square, [])))
True
>>> (symmetric_subtract_polygon_from_segment(
...      Segment(Point(1, 1), Point(8, 8)),
...      Polygon(square, [clockwise_inner_square]))
...  == Mix(EMPTY, Multisegment([Segment(Point(1, 1), Point(3, 3)),
...                               Segment(Point(4, 4), Point(8, 8))]),
...         Polygon(square, [clockwise_inner_square])))
True
```

clipping.planar.**unite_segment_with_polygon**(*segment: ground.hints.Segment*, *polygon: ground.hints.Polygon*, ***, *context: Optional[ground.base.Context] = None*) → Union[ground.hints.Mix, ground.hints.Polygon]

Returns union of segment with polygon.

Time complexity: `O(segments_count * log segments_count)`

Memory complexity: `O(segments_count)`

where `segments_count = start_segments_count + intersections_count`, `start_segments_count = polygon_edges_count + 1`, `polygon_edges_count = len(polygon.border.vertices) + sum(len(hole.vertices) for hole in polygon.holes)`, `intersections_count` — number of intersections between segment and polygon edges.

Parameters

- **segment** – first operand.
- **polygon** – second operand.
- **context** – geometric context.

Returns union of operands.

```
>>> from ground.base import get_context
>>> context = get_context()
>>> EMPTY = context.empty
>>> Contour = context.contour_cls
>>> Mix = context.mix_cls
>>> Multisegment = context.multisegment_cls
>>> Point = context.point_cls
>>> Polygon = context.polygon_cls
>>> Segment = context.segment_cls
>>> square = Contour([Point(0, 0), Point(4, 0), Point(4, 4),
...                   Point(0, 4)])
>>> inner_square = Contour([Point(1, 1), Point(3, 1), Point(3, 3),
...                         Point(1, 3)])
>>> clockwise_inner_square = Contour([Point(1, 1), Point(1, 3),
...                                   Point(3, 3), Point(3, 1)])
>>> (unite_segment_with_polygon(Segment(Point(0, 0), Point(1, 1)),
...                             Polygon(square, []))
...  == Polygon(square, []))
True
>>> (unite_segment_with_polygon(Segment(Point(0, 0), Point(1, 1)),
```

(continues on next page)

(continued from previous page)

```
...                                Polygon(inner_square, []))
...  == Mix(EMPTY, Segment(Point(0, 0), Point(1, 1)),
...         Polygon(inner_square, [])))
True
>>> (unite_segment_with_polygon(Segment(Point(1, 1), Point(8, 8)),
...                                Polygon(square, [clockwise_inner_square]))
...  == Mix(EMPTY, Multisegment([Segment(Point(1, 1), Point(3, 3)),
...                               Segment(Point(4, 4), Point(8, 8))]),
...         Polygon(square, [clockwise_inner_square])))
True
```

clipping.planar.**complete_intersect_segment_with_multipolygon**(*segment: ground.hints.Segment*, *multipolygon: ground.hints.Multipolygon*, ***, *context: Optional[ground.base.Context] = None*) → Union[ground.hints.Empty, ground.hints.Mix, ground.hints.Multipoint, ground.hints.Multisegment, ground.hints.Segment]

Returns intersection of segment with multipolygon considering cases with geometries touching each other in points.

Time complexity: `O(segments_count * log segments_count)`

Memory complexity: `O(segments_count)`

where `segments_count = start_segments_count + intersections_count`, `start_segments_count = multipolygon_edges_count + 1`, `multipolygon_edges_count = sum(len(polygon.border.vertices) + sum(len(hole.vertices) for hole in polygon.holes) for polygon in multipolygon.polygons)`, `intersections_count` — number of intersections between segment and multipolygon edges.

Parameters

- **segment** – first operand.
- **multipolygon** – second operand.
- **context** – geometric context.

Returns intersection of operands.

```
>>> from ground.base import get_context
>>> context = get_context()
>>> EMPTY = context.empty
>>> Mix = context.mix_cls
>>> Contour = context.contour_cls
>>> Multipoint = context.multipoint_cls
>>> Multipolygon = context.multipolygon_cls
>>> Multisegment = context.multisegment_cls
>>> Point = context.point_cls
>>> Polygon = context.polygon_cls
>>> Segment = context.segment_cls
```

(continues on next page)

(continued from previous page)

```
>>> first_square = Contour([Point(0, 0), Point(4, 0), Point(4, 4),
...                         Point(0, 4)])
>>> second_square = Contour([Point(4, 0), Point(8, 0), Point(8, 4),
...                          Point(4, 4)])
>>> third_square = Contour([Point(4, 4), Point(8, 4), Point(8, 8),
...                         Point(4, 8)])
>>> first_inner_square = Contour([Point(1, 1), Point(3, 1), Point(3, 3),
...                               Point(1, 3)])
>>> clockwise_first_inner_square = Contour([Point(1, 1), Point(1, 3),
...                                         Point(3, 3), Point(3, 1)])
>>> (complete_intersect_segment_with_multipolygon(
...      Segment(Point(0, 0), Point(2, 0)),
...      Multipolygon([Polygon(first_inner_square, []),
...                    Polygon(second_square, [])]))
...  is EMPTY)
True
>>> (complete_intersect_segment_with_multipolygon(
...      Segment(Point(0, 0), Point(4, 0)),
...      Multipolygon([Polygon(first_inner_square, []),
...                    Polygon(second_square, [])]))
...  == Multipoint([Point(4, 0)]))
True
>>> (complete_intersect_segment_with_multipolygon(
...      Segment(Point(0, 0), Point(2, 2)),
...      Multipolygon([Polygon(first_inner_square, []),
...                    Polygon(second_square, [])]))
...  == Segment(Point(1, 1), Point(2, 2)))
True
>>> (complete_intersect_segment_with_multipolygon(
...      Segment(Point(0, 0), Point(4, 4)),
...      Multipolygon([Polygon(first_square,
...                            [clockwise_first_inner_square]),
...                    Polygon(third_square, [])]))
...  == Multisegment([Segment(Point(0, 0), Point(1, 1)),
...                   Segment(Point(3, 3), Point(4, 4))]))
True
>>> (complete_intersect_segment_with_multipolygon(
...      Segment(Point(1, 1), Point(8, 8)),
...      Multipolygon([Polygon(first_square,
...                            [clockwise_first_inner_square]),
...                    Polygon(third_square, [])]))
...  == Mix(Multipoint([Point(1, 1)]),
...         Multisegment([Segment(Point(3, 3), Point(4, 4)),
...                       Segment(Point(4, 4), Point(8, 8))]), EMPTY))
True
```

`clipping.planar.`**`intersect_segment_with_multipolygon`**(*segment: ground.hints.Segment*, *multipolygon: ground.hints.Multipolygon*, ***, *context: Optional[ground.base.Context] = None*) → Union[ground.hints.Empty, ground.hints.Multisegment, ground.hints.Segment]

Returns intersection of segment with multipolygon.

Time complexity: `O(segments_count * log segments_count)`

Memory complexity: `O(segments_count)`

where `segments_count = start_segments_count + intersections_count`, `start_segments_count = multipolygon_edges_count + 1`, `multipolygon_edges_count = sum(len(polygon.border.vertices) + sum(len(hole.vertices) for hole in polygon.holes) for polygon in multipolygon.polygons)`, `intersections_count` — number of intersections between segment and multipolygon edges.

Parameters

- **segment** – first operand.
- **multipolygon** – second operand.
- **context** – geometric context.

Returns intersection of operands.

```
>>> from ground.base import get_context
>>> context = get_context()
>>> EMPTY = context.empty
>>> Contour = context.contour_cls
>>> Multipolygon = context.multipolygon_cls
>>> Multisegment = context.multisegment_cls
>>> Point = context.point_cls
>>> Polygon = context.polygon_cls
>>> Segment = context.segment_cls
>>> first_square = Contour([Point(0, 0), Point(4, 0), Point(4, 4),
...                         Point(0, 4)])
>>> second_square = Contour([Point(4, 0), Point(8, 0), Point(8, 4),
...                          Point(4, 4)])
>>> third_square = Contour([Point(4, 4), Point(8, 4), Point(8, 8),
...                         Point(4, 8)])
>>> first_inner_square = Contour([Point(1, 1), Point(3, 1), Point(3, 3),
...                               Point(1, 3)])
>>> clockwise_first_inner_square = Contour([Point(1, 1), Point(1, 3),
...                                         Point(3, 3), Point(3, 1)])
>>> (intersect_segment_with_multipolygon(
...      Segment(Point(0, 0), Point(2, 0)),
...      Multipolygon([Polygon(first_inner_square, []),
...                    Polygon(second_square, [])]))
...  is intersect_segment_with_multipolygon(
...          Segment(Point(0, 0), Point(4, 0)),
...          Multipolygon([Polygon(first_inner_square, []),
...                        Polygon(second_square, [])]))
...  is EMPTY)
True
>>> (intersect_segment_with_multipolygon(
...      Segment(Point(0, 0), Point(2, 2)),
...      Multipolygon([Polygon(first_inner_square, []),
...                    Polygon(second_square, [])]))
...  == Segment(Point(1, 1), Point(2, 2)))
True
```

(continues on next page)

(continued from previous page)

```
>>> (intersect_segment_with_multipolygon(
...      Segment(Point(0, 0), Point(4, 4)),
...      Multipolygon([Polygon(first_square,
...                            [clockwise_first_inner_square]),
...                    Polygon(third_square, [])]))
...   == Multisegment([Segment(Point(0, 0), Point(1, 1)),
...                    Segment(Point(3, 3), Point(4, 4))]))
True
>>> (intersect_segment_with_multipolygon(
...      Segment(Point(1, 1), Point(8, 8)),
...      Multipolygon([Polygon(first_square,
...                            [clockwise_first_inner_square]),
...                    Polygon(third_square, [])]))
...   == Multisegment([Segment(Point(3, 3), Point(4, 4)),
...                    Segment(Point(4, 4), Point(8, 8))]))
True
```

`clipping.planar.`**`subtract_multipolygon_from_segment`**(*minuend: ground.hints.Segment*, *subtrahend: ground.hints.Multipolygon*, ***, *context: Optional[ground.base.Context] = None*) → Union[ground.hints.Empty, ground.hints.Multisegment, ground.hints.Segment]

Returns difference of segment with multipolygon.

Time complexity: `O(segments_count * log segments_count)`

Memory complexity: `O(segments_count)`

where `segments_count = start_segments_count + intersections_count`, `start_segments_count = subtrahend_edges_count + 1`, `subtrahend_edges_count = sum(len(polygon.border.vertices) + sum(len(hole.vertices) for hole in polygon.holes) for polygon in subtrahend.polygons)`, `intersections_count` — number of intersections between segment and multipolygon edges.

Parameters

- **`minuend`** – segment to subtract from.
- **`subtrahend`** – multipolygon to subtract.
- **`context`** – geometric context.

Returns difference of minuend with subtrahend.

```
>>> from ground.base import get_context
>>> context = get_context()
>>> EMPTY = context.empty
>>> Contour = context.contour_cls
>>> Multipolygon = context.multipolygon_cls
>>> Multisegment = context.multisegment_cls
>>> Point = context.point_cls
>>> Polygon = context.polygon_cls
>>> Segment = context.segment_cls
>>> first_square = Contour([Point(0, 0), Point(4, 0), Point(4, 4),
...                         Point(0, 4)])
```

(continues on next page)

(continued from previous page)

```
>>> second_square = Contour([Point(4, 0), Point(8, 0), Point(8, 4),
...                          Point(4, 4)])
>>> third_square = Contour([Point(4, 4), Point(8, 4), Point(8, 8),
...                         Point(4, 8)])
>>> first_inner_square = Contour([Point(1, 1), Point(3, 1), Point(3, 3),
...                               Point(1, 3)])
>>> clockwise_first_inner_square = Contour([Point(1, 1), Point(1, 3),
...                                         Point(3, 3), Point(3, 1)])
>>> (subtract_multipolygon_from_segment(
...      Segment(Point(0, 0), Point(4, 0)),
...      Multipolygon([Polygon(first_square, []),
...                    Polygon(third_square, [])]))
...  is EMPTY)
True
>>> (subtract_multipolygon_from_segment(
...      Segment(Point(0, 0), Point(4, 4)),
...      Multipolygon([Polygon(first_square,
...                            [clockwise_first_inner_square]),
...                    Polygon(third_square, [])]))
...  == Segment(Point(1, 1), Point(3, 3)))
True
>>> (subtract_multipolygon_from_segment(
...      Segment(Point(0, 0), Point(4, 4)),
...      Multipolygon([Polygon(first_inner_square, []),
...                    Polygon(second_square, [])]))
...  == Multisegment([Segment(Point(0, 0), Point(1, 1)),
...                   Segment(Point(3, 3), Point(4, 4))]))
True
```

`clipping.planar.`**`symmetric_subtract_multipolygon_from_segment`**(*segment: ground.hints.Segment*, *multipolygon: ground.hints.Multipolygon*, *, *context: Optional[ground.base.Context] = None*) → Union[ground.hints.Mix, ground.hints.Multipolygon]

Returns symmetric difference of segment with multipolygon.

Time complexity: `O(segments_count * log segments_count)`

Memory complexity: `O(segments_count)`

where `segments_count = start_segments_count + intersections_count`, `start_segments_count = multipolygon_edges_count + 1`, `multipolygon_edges_count = sum(len(polygon.border.vertices) + sum(len(hole.vertices) for hole in polygon.holes) for polygon in multipolygon.polygons)`, `intersections_count` — number of intersections between segment and multipolygon edges.

Parameters

- **`segment`** – first operand.
- **`multipolygon`** – second operand.
- **`context`** – geometric context.

Returns symmetric difference of operands.

```
>>> from ground.base import get_context
>>> context = get_context()
>>> EMPTY = context.empty
>>> Contour = context.contour_cls
>>> Mix = context.mix_cls
>>> Multipolygon = context.multipolygon_cls
>>> Multisegment = context.multisegment_cls
>>> Point = context.point_cls
>>> Polygon = context.polygon_cls
>>> Segment = context.segment_cls
>>> first_square = Contour([Point(0, 0), Point(4, 0), Point(4, 4),
...                         Point(0, 4)])
>>> second_square = Contour([Point(4, 0), Point(8, 0), Point(8, 4),
...                          Point(4, 4)])
>>> third_square = Contour([Point(4, 4), Point(8, 4), Point(8, 8),
...                         Point(4, 8)])
>>> first_inner_square = Contour([Point(1, 1), Point(3, 1), Point(3, 3),
...                               Point(1, 3)])
>>> clockwise_first_inner_square = Contour([Point(1, 1), Point(1, 3),
...                                         Point(3, 3), Point(3, 1)])
>>> (symmetric_subtract_multipolygon_from_segment(
...      Segment(Point(0, 0), Point(4, 4)),
...      Multipolygon([Polygon(first_square, []),
...                    Polygon(third_square, [])]))
...  == Multipolygon([Polygon(first_square, []),
...                   Polygon(third_square, [])]))
True
>>> (symmetric_subtract_multipolygon_from_segment(
...      Segment(Point(0, 0), Point(4, 4)),
...      Multipolygon([Polygon(first_square,
...                            [clockwise_first_inner_square]),
...                    Polygon(third_square, [])]))
...  == Mix(EMPTY, Segment(Point(1, 1), Point(3, 3)),
...         Multipolygon([Polygon(first_square,
...                               [clockwise_first_inner_square]),
...                       Polygon(third_square, [])])))
True
>>> (symmetric_subtract_multipolygon_from_segment(
...      Segment(Point(0, 0), Point(4, 4)),
...      Multipolygon([Polygon(first_inner_square, []),
...                    Polygon(second_square, [])]))
...  == Mix(EMPTY, Multisegment([Segment(Point(0, 0), Point(1, 1)),
...                              Segment(Point(3, 3), Point(4, 4))]),
...         Multipolygon([Polygon(first_inner_square, []),
...                       Polygon(second_square, [])])))
True
```

`clipping.planar.`**`unite_segment_with_multipolygon`**(*segment: ground.hints.Segment*, *multipolygon: ground.hints.Multipolygon*, ***, *context: Optional[ground.base.Context] = None*) → Union[ground.hints.Mix, ground.hints.Multipolygon]

Returns union of segment with multipolygon.

Time complexity: `O(segments_count * log segments_count)`

Memory complexity: `O(segments_count)`

where `segments_count = start_segments_count + intersections_count`, `start_segments_count = multipolygon_edges_count + 1`, `multipolygon_edges_count = sum(len(polygon.border.vertices) + sum(len(hole.vertices) for hole in polygon.holes) for polygon in multipolygon.polygons)`, `intersections_count` — number of intersections between segment and multipolygon edges.

Parameters

- **segment** – first operand.
- **multipolygon** – second operand.
- **context** – geometric context.

Returns union of operands.

```
>>> from ground.base import get_context
>>> context = get_context()
>>> EMPTY = context.empty
>>> Contour = context.contour_cls
>>> Mix = context.mix_cls
>>> Multipolygon = context.multipolygon_cls
>>> Multisegment = context.multisegment_cls
>>> Point = context.point_cls
>>> Polygon = context.polygon_cls
>>> Segment = context.segment_cls
>>> first_square = Contour([Point(0, 0), Point(4, 0), Point(4, 4),
...                         Point(0, 4)])
>>> second_square = Contour([Point(4, 0), Point(8, 0), Point(8, 4),
...                          Point(4, 4)])
>>> third_square = Contour([Point(4, 4), Point(8, 4), Point(8, 8),
...                         Point(4, 8)])
>>> first_inner_square = Contour([Point(1, 1), Point(3, 1), Point(3, 3),
...                               Point(1, 3)])
>>> clockwise_first_inner_square = Contour([Point(1, 1), Point(1, 3),
...                                         Point(3, 3), Point(3, 1)])
>>> (unite_segment_with_multipolygon(
...      Segment(Point(0, 0), Point(4, 4)),
...      Multipolygon([Polygon(first_square, []),
...                    Polygon(third_square, [])]))
...  == Multipolygon([Polygon(first_square, []),
...                   Polygon(third_square, [])]))
True
>>> (unite_segment_with_multipolygon(
...      Segment(Point(0, 0), Point(4, 4)),
...      Multipolygon([Polygon(first_square,
...                            [clockwise_first_inner_square]),
...                    Polygon(third_square, [])]))
...  == Mix(EMPTY, Segment(Point(1, 1), Point(3, 3)),
...         Multipolygon([Polygon(first_square,
...                               [clockwise_first_inner_square]),
...                       Polygon(third_square, [])])))
True
```

(continues on next page)

(continued from previous page)

```
>>> (unite_segment_with_multipolygon(
...      Segment(Point(0, 0), Point(4, 4)),
...      Multipolygon([Polygon(first_inner_square, []),
...                    Polygon(second_square, [])]))
...  == Mix(EMPTY, Multisegment([Segment(Point(0, 0), Point(1, 1)),
...                               Segment(Point(3, 3), Point(4, 4))]),
...         Multipolygon([Polygon(first_inner_square, []),
...                       Polygon(second_square, [])])))
True
```

clipping.planar.**segments_to_multisegment**(*segments: Sequence[ground.hints.Segment]*, ***, *context: Optional[ground.base.Context] = None*) → Union[ground.hints.Empty, ground.hints.Segment, ground.hints.Multisegment]

Returns multisegment from given segments.

Time complexity: `O(segments_count * log segments_count)`

Memory complexity: `O(segments_count)`

where `segments_count = segments_count + intersections_count`, `segments_count = len(segments)`, `intersections_count` — number of intersections between segments.

Parameters

- **segments** – target segments.
- **context** – geometric context.

Returns multisegment from segments.

```
>>> from ground.base import get_context
>>> context = get_context()
>>> Multisegment = context.multisegment_cls
>>> Point = context.point_cls
>>> Segment = context.segment_cls
>>> (segments_to_multisegment([Segment(Point(0, 0), Point(1, 0)),
...                            Segment(Point(0, 1), Point(1, 0))])
...  == Multisegment([Segment(Point(0, 0), Point(1, 0)),
...                   Segment(Point(0, 1), Point(1, 0))]))
True
>>> (segments_to_multisegment([Segment(Point(0, 0), Point(2, 0)),
...                            Segment(Point(1, 0), Point(3, 0))])
...  == Multisegment([Segment(Point(0, 0), Point(1, 0)),
...                   Segment(Point(1, 0), Point(2, 0)),
...                   Segment(Point(2, 0), Point(3, 0))]))
True
```

clipping.planar.**complete_intersect_multisegments**(*first: ground.hints.Multisegment*, *second: ground.hints.Multisegment*, ***, *context: Optional[ground.base.Context] = None*) → Union[ground.hints.Empty, ground.hints.Mix, ground.hints.Multipoint, ground.hints.Multisegment, ground.hints.Segment]

Returns intersection of multisegments considering cases with segments touching each other in points.

Time complexity: `O(segments_count * log segments_count)`

Memory complexity: `O(segments_count)`

where `segments_count = segments_count + intersections_count`, `segments_count = len(first.segments) + len(second.segments)`, `intersections_count` — number of intersections between multisegments.

Parameters

- **first** – first operand.
- **second** – second operand.
- **context** – geometric context.

Returns intersection of operands.

```
>>> from ground.base import get_context
>>> context = get_context()
>>> EMPTY = context.empty
>>> Mix = context.mix_cls
>>> Multipoint = context.multipoint_cls
>>> Multisegment = context.multisegment_cls
>>> Point = context.point_cls
>>> Segment = context.segment_cls
>>> (complete_intersect_multisegments(
...      Multisegment([Segment(Point(0, 0), Point(1, 0)),
...                    Segment(Point(0, 1), Point(1, 0))]),
...      Multisegment([Segment(Point(0, 0), Point(1, 0)),
...                    Segment(Point(0, 1), Point(1, 0))]))
...  == Multisegment([Segment(Point(0, 0), Point(1, 0)),
...                   Segment(Point(0, 1), Point(1, 0))]))
True
>>> (complete_intersect_multisegments(
...      Multisegment([Segment(Point(0, 0), Point(1, 0)),
...                    Segment(Point(0, 1), Point(1, 1))]),
...      Multisegment([Segment(Point(0, 0), Point(2, 0)),
...                    Segment(Point(0, 0), Point(2, 2))]))
...  == Mix(Multipoint([Point(1, 1)]), Segment(Point(0, 0), Point(1, 0)),
...         EMPTY))
True
```

`clipping.planar.`**`intersect_multisegments`**(*first: ground.hints.Multisegment*, *second: ground.hints.Multisegment*, *, *context: Optional[ground.base.Context] = None*) → Union[ground.hints.Empty, ground.hints.Segment, ground.hints.Multisegment]

Returns intersection of multisegments.

Time complexity: `O(segments_count * log segments_count)`

Memory complexity: `O(segments_count)`

where `segments_count = segments_count + intersections_count`, `segments_count = len(first.segments) + len(second.segments)`, `intersections_count` — number of intersections between multisegments.

Parameters

- **first** – first operand.
- **second** – second operand.
- **context** – geometric context.

Returns intersection of operands.

```
>>> from ground.base import get_context
>>> context = get_context()
>>> Multisegment = context.multisegment_cls
>>> Point = context.point_cls
>>> Segment = context.segment_cls
>>> (intersect_multisegments(
...      Multisegment([Segment(Point(0, 0), Point(1, 0)),
...                    Segment(Point(0, 1), Point(1, 0))]),
...      Multisegment([Segment(Point(0, 0), Point(1, 0)),
...                    Segment(Point(0, 1), Point(1, 0))]))
...  == Multisegment([Segment(Point(0, 0), Point(1, 0)),
...                   Segment(Point(0, 1), Point(1, 0))]))
True
>>> (intersect_multisegments(
...      Multisegment([Segment(Point(0, 0), Point(1, 0)),
...                    Segment(Point(0, 1), Point(1, 1))]),
...      Multisegment([Segment(Point(0, 0), Point(2, 0)),
...                    Segment(Point(0, 0), Point(2, 2))]))
...  == Segment(Point(0, 0), Point(1, 0)))
True
```

`clipping.planar.`**`subtract_multisegments`**(*minuend: ground.hints.Multisegment*, *subtrahend: ground.hints.Multisegment*, ***, *context: Optional[ground.base.Context] = None*) → Union[ground.hints.Empty, ground.hints.Segment, ground.hints.Multisegment]

Returns difference of multisegments.

Time complexity: `O(segments_count * log segments_count)`

Memory complexity: `O(segments_count)`

where `segments_count = segments_count + intersections_count`, `segments_count = len(first.segments) + len(second.segments)`, `intersections_count` — number of intersections between multisegments.

Parameters

- **minuend** – multisegment to subtract from.
- **subtrahend** – multisegment to subtract.
- **context** – geometric context.

Returns difference between minuend and subtrahend.

```
>>> from ground.base import get_context
>>> context = get_context()
>>> EMPTY = context.empty
>>> Multisegment = context.multisegment_cls
>>> Point = context.point_cls
```

(continues on next page)

(continued from previous page)

```
>>> Segment = context.segment_cls
>>> (subtract_multisegments(
...      Multisegment([Segment(Point(0, 0), Point(1, 0)),
...                    Segment(Point(0, 1), Point(1, 0))]),
...      Multisegment([Segment(Point(0, 0), Point(1, 0)),
...                    Segment(Point(0, 1), Point(1, 0))]))
...  is EMPTY)
True
>>> (subtract_multisegments(
...      Multisegment([Segment(Point(0, 0), Point(1, 0)),
...                    Segment(Point(0, 1), Point(1, 1))]),
...      Multisegment([Segment(Point(0, 0), Point(2, 0)),
...                    Segment(Point(0, 0), Point(2, 2))]))
...  == Segment(Point(0, 1), Point(1, 1)))
True
```

`clipping.planar.`**`symmetric_subtract_multisegments`**(*first: ground.hints.Multisegment*, *second: ground.hints.Multisegment*, ***, *context: Optional[ground.base.Context] = None*) → Union[ground.hints.Empty, ground.hints.Segment, ground.hints.Multisegment]

Returns symmetric difference of multisegments.

Time complexity: `O(segments_count * log segments_count)`

Memory complexity: `O(segments_count)`

where `segments_count = segments_count + intersections_count`, `segments_count = len(first.segments) + len(second.segments)`, `intersections_count` — number of intersections between multisegments.

Parameters

- **first** – first operand.
- **second** – second operand.
- **context** – geometric context.

Returns symmetric difference of operands.

```
>>> from ground.base import get_context
>>> context = get_context()
>>> EMPTY = context.empty
>>> Multisegment = context.multisegment_cls
>>> Point = context.point_cls
>>> Segment = context.segment_cls
>>> (symmetric_subtract_multisegments(
...      Multisegment([Segment(Point(0, 0), Point(1, 0)),
...                    Segment(Point(0, 1), Point(1, 0))]),
...      Multisegment([Segment(Point(0, 0), Point(1, 0)),
...                    Segment(Point(0, 1), Point(1, 0))]))
...  is EMPTY)
True
>>> (symmetric_subtract_multisegments(
...      Multisegment([Segment(Point(0, 0), Point(1, 0)),
```

(continues on next page)

(continued from previous page)

```
...                        Segment(Point(0, 1), Point(1, 1))]),
...         Multisegment([Segment(Point(0, 0), Point(2, 0)),
...                        Segment(Point(0, 0), Point(2, 2))]))
...     == Multisegment([Segment(Point(0, 0), Point(1, 1)),
...                       Segment(Point(0, 1), Point(1, 1)),
...                       Segment(Point(1, 0), Point(2, 0)),
...                       Segment(Point(1, 1), Point(2, 2))]))
True
```

clipping.planar.**unite_multisegments**(*first: ground.hints.Multisegment*, *second: ground.hints.Multisegment*, **, context: Optional[ground.base.Context] = None*) → ground.hints.Multisegment

Returns union of multisegments.

Time complexity: `O(segments_count * log segments_count)`

Memory complexity: `O(segments_count)`

where `segments_count = segments_count + intersections_count`, `segments_count = len(first.segments) + len(second.segments)`, `intersections_count` — number of intersections between multisegments.

Parameters

- **first** – first operand.
- **second** – second operand.
- **context** – geometric context.

Returns union of operands.

```
>>> from ground.base import get_context
>>> context = get_context()
>>> Multisegment = context.multisegment_cls
>>> Point = context.point_cls
>>> Segment = context.segment_cls
>>> (unite_multisegments(Multisegment([Segment(Point(0, 0), Point(1, 0)),
...                                    Segment(Point(0, 1), Point(1, 0))]),
...                      Multisegment([Segment(Point(0, 0), Point(1, 0)),
...                                    Segment(Point(0, 1), Point(1, 0))]))
...     == Multisegment([Segment(Point(0, 0), Point(1, 0)),
...                       Segment(Point(0, 1), Point(1, 0))]))
True
>>> (unite_multisegments(Multisegment([Segment(Point(0, 0), Point(1, 0)),
...                                    Segment(Point(0, 1), Point(1, 1))]),
...                      Multisegment([Segment(Point(0, 0), Point(2, 0)),
...                                    Segment(Point(0, 0), Point(2, 2))]))
...     == Multisegment([Segment(Point(0, 0), Point(1, 0)),
...                       Segment(Point(0, 0), Point(1, 1)),
...                       Segment(Point(0, 1), Point(1, 1)),
...                       Segment(Point(1, 0), Point(2, 0)),
...                       Segment(Point(1, 1), Point(2, 2))]))
True
```

clipping.planar.**complete_intersect_multisegment_with_polygon**(*multisegment: ground.hints.Multisegment*, *polygon: ground.hints.Polygon*, ***, *context: Optional[ground.base.Context] = None*) → Union[ground.hints.Empty, ground.hints.Mix, ground.hints.Multipoint, ground.hints.Multisegment, ground.hints.Segment]

Returns intersection of multisegment with polygon considering cases with geometries touching each other in points.

Time complexity: `O(segments_count * log segments_count)`

Memory complexity: `O(segments_count)`

where `segments_count = start_segments_count + intersections_count`, `start_segments_count = len(multisegment.segments) + polygon_edges_count`, `polygon_edges_count = len(polygon.border.vertices) + sum(len(hole.vertices) for hole in polygon.holes)`, `intersections_count` — number of intersections between multisegment and polygon edges.

Parameters

- **multisegment** – multisegment to intersect with.
- **polygon** – polygon to intersect with.
- **context** – geometric context.

Returns intersection of multisegment with polygon.

```
>>> from ground.base import get_context
>>> context = get_context()
>>> EMPTY = context.empty
>>> Mix = context.mix_cls
>>> Contour = context.contour_cls
>>> Multipoint = context.multipoint_cls
>>> Multisegment = context.multisegment_cls
>>> Point = context.point_cls
>>> Polygon = context.polygon_cls
>>> Segment = context.segment_cls
>>> (complete_intersect_multisegment_with_polygon(
...      Multisegment([Segment(Point(0, 0), Point(1, 0)),
...                    Segment(Point(0, 1), Point(1, 0))]),
...      Polygon(Contour([Point(0, 0), Point(1, 0), Point(0, 1)]), []))
...  == Multisegment([Segment(Point(0, 0), Point(1, 0)),
...                   Segment(Point(0, 1), Point(1, 0))]))
True
>>> (complete_intersect_multisegment_with_polygon(
...      Multisegment([Segment(Point(0, 0), Point(1, 0)),
...                    Segment(Point(1, 1), Point(2, 2))]),
...      Polygon(Contour([Point(0, 0), Point(1, 0), Point(1, 1),
...                       Point(0, 1)]), []))
...  == Mix(Multipoint([Point(1, 1)]), Segment(Point(0, 0), Point(1, 0)),
```

(continues on next page)

(continued from previous page)

```
...         EMPTY))
True
```

clipping.planar.**intersect_multisegment_with_polygon**(*multisegment: ground.hints.Multisegment, polygon: ground.hints.Polygon, *, context: Optional[ground.base.Context] = None*) → Union[ground.hints.Empty, ground.hints.Multisegment, ground.hints.Segment]

Returns intersection of multisegment with polygon.

Time complexity: `O(segments_count * log segments_count)`

Memory complexity: `O(segments_count)`

where `segments_count = start_segments_count + intersections_count`, `start_segments_count = len(multisegment.segments) + polygon_edges_count`, `polygon_edges_count = len(polygon.border.vertices) + sum(len(hole.vertices) for hole in polygon.holes)`, `intersections_count` — number of intersections between multisegment and polygon edges.

Parameters

- **multisegment** – multisegment to intersect with.
- **polygon** – polygon to intersect with.
- **context** – geometric context.

Returns intersection of multisegment with polygon.

```
>>> from ground.base import get_context
>>> context = get_context()
>>> Contour = context.contour_cls
>>> Multisegment = context.multisegment_cls
>>> Point = context.point_cls
>>> Polygon = context.polygon_cls
>>> Segment = context.segment_cls
>>> (intersect_multisegment_with_polygon(
...      Multisegment([Segment(Point(0, 0), Point(1, 0)),
...                    Segment(Point(0, 1), Point(1, 0))]),
...      Polygon(Contour([Point(0, 0), Point(1, 0), Point(0, 1)]), []))
...  == Multisegment([Segment(Point(0, 0), Point(1, 0)),
...                   Segment(Point(0, 1), Point(1, 0))]))
True
>>> (intersect_multisegment_with_polygon(
...      Multisegment([Segment(Point(0, 0), Point(1, 0)),
...                    Segment(Point(1, 1), Point(2, 2))]),
...      Polygon(Contour([Point(0, 0), Point(1, 0), Point(1, 1),
...                       Point(0, 1)]), []))
...  == Segment(Point(0, 0), Point(1, 0)))
True
```

clipping.planar.**subtract_polygon_from_multisegment**(*minuend: ground.hints.Multisegment*, *subtrahend: ground.hints.Polygon*, ***, *context: Optional[ground.base.Context] = None*) → Union[ground.hints.Empty, ground.hints.Multisegment, ground.hints.Segment]

Returns difference of multisegment with polygon.

Time complexity: `O(segments_count * log segments_count)`

Memory complexity: `O(segments_count)`

where `segments_count = start_segments_count + intersections_count`, `start_segments_count = len(minuend.segments) + subtrahend_edges_count`, `subtrahend_edges_count = len(polygon.border.vertices) + sum(len(hole.vertices) for hole in polygon.holes)`, `intersections_count` — number of intersections between multisegment and polygon edges.

Parameters

- **minuend** – multisegment to subtract from.
- **subtrahend** – polygon to subtract.
- **context** – geometric context.

Returns difference of minuend with subtrahend.

```
>>> from ground.base import get_context
>>> context = get_context()
>>> EMPTY = context.empty
>>> Contour = context.contour_cls
>>> Multipoint = context.multipoint_cls
>>> Polygon = context.polygon_cls
>>> Multisegment = context.multisegment_cls
>>> Point = context.point_cls
>>> Polygon = context.polygon_cls
>>> Segment = context.segment_cls
>>> (subtract_polygon_from_multisegment(
...      Multisegment([Segment(Point(0, 0), Point(1, 0)),
...                    Segment(Point(0, 1), Point(1, 0))]),
...      Polygon(Contour([Point(0, 0), Point(1, 0), Point(0, 1)]), []))
...  is EMPTY)
True
>>> (subtract_polygon_from_multisegment(
...      Multisegment([Segment(Point(0, 0), Point(1, 0)),
...                    Segment(Point(1, 1), Point(2, 2))]),
...      Polygon(Contour([Point(0, 0), Point(1, 0), Point(1, 1),
...                       Point(0, 1)]), []))
...  == Segment(Point(1, 1), Point(2, 2)))
True
```

clipping.planar.**symmetric_subtract_polygon_from_multisegment**(*multisegment: ground.hints.Multisegment*, *polygon: ground.hints.Polygon*, ***, *context: Optional[ground.base.Context] = None*) → Union[ground.hints.Mix, ground.hints.Polygon]

Returns symmetric difference of multisegment with polygon.

Time complexity: `O(segments_count * log segments_count)`

Memory complexity: `O(segments_count)`

where `segments_count = start_segments_count + intersections_count`, `start_segments_count = len(multisegment.segments) + polygon_edges_count`, `polygon_edges_count = len(polygon.border.vertices) + sum(len(hole.vertices) for hole in polygon.holes)`, `intersections_count` — number of intersections between multisegment and polygon edges.

Parameters

- **multisegment** – first operand.
- **polygon** – second operand.
- **context** – geometric context.

Returns symmetric difference of operands.

```
>>> from ground.base import get_context
>>> context = get_context()
>>> EMPTY = context.empty
>>> Contour = context.contour_cls
>>> Mix = context.mix_cls
>>> Polygon = context.polygon_cls
>>> Multisegment = context.multisegment_cls
>>> Point = context.point_cls
>>> Polygon = context.polygon_cls
>>> Segment = context.segment_cls
>>> (symmetric_subtract_polygon_from_multisegment(
...      Multisegment([Segment(Point(0, 0), Point(1, 0)),
...                    Segment(Point(0, 1), Point(1, 0))]),
...      Polygon(Contour([Point(0, 0), Point(1, 0), Point(0, 1)]), []))
...  == Polygon(Contour([Point(0, 0), Point(1, 0), Point(0, 1)]), []))
True
>>> (symmetric_subtract_polygon_from_multisegment(
...      Multisegment([Segment(Point(0, 0), Point(1, 0)),
...                    Segment(Point(1, 1), Point(2, 2))]),
...      Polygon(Contour([Point(0, 0), Point(1, 0), Point(1, 1),
...                       Point(0, 1)]), []))
...  == Mix(EMPTY, Segment(Point(1, 1), Point(2, 2)),
...         Polygon(Contour([Point(0, 0), Point(1, 0), Point(1, 1),
...                          Point(0, 1)]), [])))
True
>>> (symmetric_subtract_polygon_from_multisegment(
...      Multisegment([Segment(Point(0, 0), Point(1, 0)),
...                    Segment(Point(1, 0), Point(2, 0)),
...                    Segment(Point(1, 1), Point(2, 2))]),
...      Polygon(Contour([Point(0, 0), Point(1, 0), Point(1, 1),
...                       Point(0, 1)]), []))
...  == Mix(EMPTY, Multisegment([Segment(Point(1, 0), Point(2, 0)),
...                              Segment(Point(1, 1), Point(2, 2))]),
...         Polygon(Contour([Point(0, 0), Point(1, 0), Point(1, 1),
```

(continues on next page)

(continued from previous page)

```
...                         Point(0, 1)]), []))
True
```

clipping.planar.**unite_multisegment_with_polygon**(*multisegment: ground.hints.Multisegment*, *polygon: ground.hints.Polygon*, ***, *context: Optional[ground.base.Context] = None*) → Union[ground.hints.Mix, ground.hints.Polygon]

Returns union of multisegment with polygon.

Time complexity: `O(segments_count * log segments_count)`

Memory complexity: `O(segments_count)`

where `segments_count = start_segments_count + intersections_count`, `start_segments_count = len(multisegment.segments) + polygon_edges_count`, `polygon_edges_count = len(polygon.border.vertices) + sum(len(hole.vertices) for hole in polygon.holes)`, `intersections_count` — number of intersections between multisegment and polygon edges.

Parameters

- **multisegment** – first operand.
- **polygon** – second operand.
- **context** – geometric context.

Returns union of operands.

```
>>> from ground.base import get_context
>>> context = get_context()
>>> EMPTY = context.empty
>>> Contour = context.contour_cls
>>> Mix = context.mix_cls
>>> Polygon = context.polygon_cls
>>> Multisegment = context.multisegment_cls
>>> Point = context.point_cls
>>> Polygon = context.polygon_cls
>>> Segment = context.segment_cls
>>> (unite_multisegment_with_polygon(
...      Multisegment([Segment(Point(0, 0), Point(1, 0)),
...                    Segment(Point(0, 1), Point(1, 0))]),
...      Polygon(Contour([Point(0, 0), Point(1, 0), Point(0, 1)]), []))
...  == Polygon(Contour([Point(0, 0), Point(1, 0), Point(0, 1)]), []))
True
>>> (unite_multisegment_with_polygon(
...      Multisegment([Segment(Point(0, 0), Point(1, 0)),
...                    Segment(Point(1, 1), Point(2, 2))]),
...      Polygon(Contour([Point(0, 0), Point(1, 0), Point(1, 1),
...                       Point(0, 1)]), []))
...  == Mix(EMPTY, Segment(Point(1, 1), Point(2, 2)),
...         Polygon(Contour([Point(0, 0), Point(1, 0), Point(1, 1),
...                          Point(0, 1)]), [])))
True
>>> (unite_multisegment_with_polygon(
...      Multisegment([Segment(Point(0, 0), Point(1, 0)),
```

(continues on next page)

(continued from previous page)

```
...                      Segment(Point(1, 0), Point(2, 0)),
...                      Segment(Point(1, 1), Point(2, 2))]),
...     Polygon(Contour([Point(0, 0), Point(1, 0), Point(1, 1),
...                         Point(0, 1)]), []))
...  == Mix(EMPTY, Multisegment([Segment(Point(1, 0), Point(2, 0)),
...                              Segment(Point(1, 1), Point(2, 2))]),
...         Polygon(Contour([Point(0, 0), Point(1, 0), Point(1, 1),
...                            Point(0, 1)]), [])))
True
```

`clipping.planar.`**`complete_intersect_multisegment_with_multipolygon`**(*multisegment: ground.hints.Multisegment*, *multipolygon: ground.hints.Multipolygon*, ***, *context: Optional[ground.base.Context] = None*) → Union[ground.hints.Empty, ground.hints.Mix, ground.hints.Multipoint, ground.hints.Multisegment, ground.hints.Segment]

Returns intersection of multisegment with multipolygon considering cases with geometries touching each other in points.

Time complexity: `O(segments_count * log segments_count)`

Memory complexity: `O(segments_count)`

where `segments_count = start_segments_count + intersections_count`, `start_segments_count = len(multisegment.segments) + multipolygon_edges_count`, `multipolygon_edges_count = sum(len(polygon.border.vertices) + sum(len(hole.vertices) for hole in polygon.holes) for polygon in multipolygon.polygons)`, `intersections_count` — number of intersections between multisegment and multipolygon edges.

Parameters

- **multisegment** – multisegment to intersect with.
- **multipolygon** – multipolygon to intersect with.
- **context** – geometric context.

Returns intersection of multisegment with multipolygon.

```
>>> from ground.base import get_context
>>> context = get_context()
>>> EMPTY = context.empty
>>> Mix = context.mix_cls
>>> Contour = context.contour_cls
>>> Multipoint = context.multipoint_cls
>>> Multipolygon = context.multipolygon_cls
>>> Multisegment = context.multisegment_cls
>>> Point = context.point_cls
>>> Polygon = context.polygon_cls
>>> Segment = context.segment_cls
```

(continues on next page)

(continued from previous page)

```
>>> first_square = Contour([Point(0, 0), Point(4, 0), Point(4, 4),
...                          Point(0, 4)])
>>> second_square = Contour([Point(4, 0), Point(8, 0), Point(8, 4),
...                           Point(4, 4)])
>>> third_square = Contour([Point(4, 4), Point(8, 4), Point(8, 8),
...                          Point(4, 8)])
>>> first_inner_square = Contour([Point(1, 1), Point(3, 1), Point(3, 3),
...                                Point(1, 3)])
>>> clockwise_first_inner_square = Contour([Point(1, 1), Point(1, 3),
...                                          Point(3, 3), Point(3, 1)])
>>> (complete_intersect_multisegment_with_multipolygon(
...      Multisegment([Segment(Point(0, 0), Point(2, 0)),
...                    Segment(Point(0, 0), Point(0, 2))]),
...      Multipolygon([Polygon(first_inner_square, []),
...                    Polygon(second_square, [])]))
...  is EMPTY)
True
>>> (complete_intersect_multisegment_with_multipolygon(
...      Multisegment([Segment(Point(0, 0), Point(4, 0)),
...                    Segment(Point(0, 0), Point(0, 4))]),
...      Multipolygon([Polygon(first_inner_square, []),
...                    Polygon(second_square, [])]))
...  == Multipoint([Point(4, 0)]))
True
>>> (complete_intersect_multisegment_with_multipolygon(
...      Multisegment([Segment(Point(0, 0), Point(2, 0)),
...                    Segment(Point(0, 0), Point(2, 2))]),
...      Multipolygon([Polygon(first_inner_square, []),
...                    Polygon(second_square, [])]))
...  == Segment(Point(1, 1), Point(2, 2)))
True
>>> (complete_intersect_multisegment_with_multipolygon(
...      Multisegment([Segment(Point(0, 0), Point(4, 0)),
...                    Segment(Point(0, 0), Point(4, 4))]),
...      Multipolygon([Polygon(first_square, []),
...                    Polygon(third_square, [])]))
...  == Multisegment([Segment(Point(0, 0), Point(4, 0)),
...                   Segment(Point(0, 0), Point(4, 4))]))
True
>>> (complete_intersect_multisegment_with_multipolygon(
...      Multisegment([Segment(Point(0, 0), Point(4, 0)),
...                    Segment(Point(0, 0), Point(4, 4))]),
...      Multipolygon([Polygon(first_square,
...                            [clockwise_first_inner_square]),
...                    Polygon(third_square, [])]))
...  == Multisegment([Segment(Point(0, 0), Point(4, 0)),
...                   Segment(Point(0, 0), Point(1, 1)),
...                   Segment(Point(3, 3), Point(4, 4))]))
True
>>> (complete_intersect_multisegment_with_multipolygon(
...      Multisegment([Segment(Point(0, 0), Point(4, 0)),
...                    Segment(Point(0, 0), Point(4, 4))]),
```

(continues on next page)

(continued from previous page)

```
...     Multipolygon([Polygon(first_inner_square, []),
...                   Polygon(second_square, [])]))
...  == Mix(Multipoint([Point(4, 0), Point(4, 4)]),
...        Segment(Point(1, 1), Point(3, 3)), EMPTY))
True
```

clipping.planar.**intersect_multisegment_with_multipolygon**(*multisegment: ground.hints.Multisegment*, *multipolygon: ground.hints.Multipolygon*, *, *context: Optional[ground.base.Context] = None*) → Union[ground.hints.Empty, ground.hints.Multisegment, ground.hints.Segment]

Returns intersection of multisegment with multipolygon.

Time complexity: `O(segments_count * log segments_count)`

Memory complexity: `O(segments_count)`

where `segments_count = start_segments_count + intersections_count`, `start_segments_count = len(multisegment.segments) + multipolygon_edges_count`, `multipolygon_edges_count = sum(len(polygon.border.vertices) + sum(len(hole.vertices) for hole in polygon.holes) for polygon in multipolygon.polygons)`, `intersections_count` — number of intersections between multisegment and multipolygon edges.

Parameters

- **multisegment** – multisegment to intersect with.
- **multipolygon** – multipolygon to intersect with.
- **context** – geometric context.

Returns intersection of multisegment with multipolygon.

```
>>> from ground.base import get_context
>>> context = get_context()
>>> EMPTY = context.empty
>>> Contour = context.contour_cls
>>> Multipolygon = context.multipolygon_cls
>>> Multisegment = context.multisegment_cls
>>> Point = context.point_cls
>>> Polygon = context.polygon_cls
>>> Segment = context.segment_cls
>>> first_square = Contour([Point(0, 0), Point(4, 0), Point(4, 4),
...                          Point(0, 4)])
>>> second_square = Contour([Point(4, 0), Point(8, 0), Point(8, 4),
...                           Point(4, 4)])
>>> third_square = Contour([Point(4, 4), Point(8, 4), Point(8, 8),
...                          Point(4, 8)])
>>> first_inner_square = Contour([Point(1, 1), Point(3, 1), Point(3, 3),
...                                Point(1, 3)])
>>> clockwise_first_inner_square = Contour([Point(1, 1), Point(1, 3),
...                                          Point(3, 3), Point(3, 1)])
>>> (intersect_multisegment_with_multipolygon(
...     Multisegment([Segment(Point(0, 0), Point(4, 0)),
```

(continues on next page)

(continued from previous page)

```
...                     Segment(Point(0, 0), Point(0, 4))]),
...         Multipolygon([Polygon(first_inner_square, []),
...                       Polygon(second_square, [])]))
...     is EMPTY)
True
>>> (intersect_multisegment_with_multipolygon(
...         Multisegment([Segment(Point(0, 0), Point(4, 0)),
...                       Segment(Point(0, 0), Point(4, 4))]),
...         Multipolygon([Polygon(first_inner_square, []),
...                       Polygon(second_square, [])]))
...     == Segment(Point(1, 1), Point(3, 3)))
True
>>> (intersect_multisegment_with_multipolygon(
...         Multisegment([Segment(Point(0, 0), Point(4, 0)),
...                       Segment(Point(0, 0), Point(4, 4))]),
...         Multipolygon([Polygon(first_square, []),
...                       Polygon(third_square, [])]))
...     == Multisegment([Segment(Point(0, 0), Point(4, 0)),
...                      Segment(Point(0, 0), Point(4, 4))]))
True
>>> (intersect_multisegment_with_multipolygon(
...         Multisegment([Segment(Point(0, 0), Point(4, 0)),
...                       Segment(Point(0, 0), Point(4, 4))]),
...         Multipolygon([Polygon(first_square,
...                               [clockwise_first_inner_square]),
...                       Polygon(third_square, [])]))
...     == Multisegment([Segment(Point(0, 0), Point(4, 0)),
...                      Segment(Point(0, 0), Point(1, 1)),
...                      Segment(Point(3, 3), Point(4, 4))]))
True
```

`clipping.planar.`**`subtract_multipolygon_from_multisegment`**(*minuend: ground.hints.Multisegment*, *subtrahend: ground.hints.Multipolygon*, *, *context: Optional[ground.base.Context] = None*) → Union[ground.hints.Empty, ground.hints.Multisegment, ground.hints.Segment]

Returns difference of multisegment with multipolygon.

Time complexity: `O(segments_count * log segments_count)`

Memory complexity: `O(segments_count)`

where `segments_count = start_segments_count + intersections_count`, `start_segments_count = len(minuend.segments) + multipolygon_edges_count`, `subtrahend_edges_count = sum(len(polygon.border.vertices) + sum(len(hole.vertices) for hole in polygon.holes) for polygon in subtrahend.polygons)`, `intersections_count` — number of intersections between multisegment and multipolygon edges.

Parameters

- **`minuend`** – multisegment to subtract from.
- **`subtrahend`** – multipolygon to subtract.
- **`context`** – geometric context.

Returns difference of minuend with subtrahend.

```
>>> from ground.base import get_context
>>> context = get_context()
>>> EMPTY = context.empty
>>> Contour = context.contour_cls
>>> Multipolygon = context.multipolygon_cls
>>> Multisegment = context.multisegment_cls
>>> Point = context.point_cls
>>> Polygon = context.polygon_cls
>>> Segment = context.segment_cls
>>> first_square = Contour([Point(0, 0), Point(4, 0), Point(4, 4),
...                         Point(0, 4)])
>>> second_square = Contour([Point(4, 0), Point(8, 0), Point(8, 4),
...                          Point(4, 4)])
>>> third_square = Contour([Point(4, 4), Point(8, 4), Point(8, 8),
...                         Point(4, 8)])
>>> first_inner_square = Contour([Point(1, 1), Point(3, 1), Point(3, 3),
...                               Point(1, 3)])
>>> clockwise_first_inner_square = Contour([Point(1, 1), Point(1, 3),
...                                         Point(3, 3), Point(3, 1)])
>>> (subtract_multipolygon_from_multisegment(
...      Multisegment([Segment(Point(0, 0), Point(4, 0)),
...                    Segment(Point(0, 0), Point(4, 4))]),
...      Multipolygon([Polygon(first_square, []),
...                    Polygon(third_square, [])]))
...  is EMPTY)
True
>>> (subtract_multipolygon_from_multisegment(
...      Multisegment([Segment(Point(0, 0), Point(4, 0)),
...                    Segment(Point(0, 0), Point(4, 4))]),
...      Multipolygon([Polygon(first_square, [clockwise_first_inner_square]),
...                    Polygon(third_square, [])]))
...  == Segment(Point(1, 1), Point(3, 3)))
True
>>> (subtract_multipolygon_from_multisegment(
...      Multisegment([Segment(Point(0, 0), Point(4, 0)),
...                    Segment(Point(0, 0), Point(4, 4))]),
...      Multipolygon([Polygon(first_inner_square, []),
...                    Polygon(second_square, [])]))
...  == Multisegment([Segment(Point(0, 0), Point(1, 1)),
...                   Segment(Point(0, 0), Point(4, 0)),
...                   Segment(Point(3, 3), Point(4, 4))]))
True
```

clipping.planar.**symmetric_subtract_multipolygon_from_multisegment**(*multisegment: ground.hints.Multisegment, multipolygon: ground.hints.Multipolygon, *, context: Optional[ground.base.Context] = None*) → Union[ground.hints.Mix, ground.hints.Multipolygon]

Returns symmetric difference of multisegment with multipolygon.

Time complexity: `O(segments_count * log segments_count)`

Memory complexity: `O(segments_count)`

where `segments_count = start_segments_count + intersections_count`, `start_segments_count = len(multisegment.segments) + multipolygon_edges_count`, `multipolygon_edges_count = len(multipolygon.border.vertices) + sum(len(hole.vertices) for hole in multipolygon.holes)`, `intersections_count` — number of intersections between multisegment and multipolygon edges.

Parameters

- **multisegment** – first operand.
- **multipolygon** – second operand.
- **context** – geometric context.

Returns symmetric difference of operands.

```
>>> from ground.base import get_context
>>> context = get_context()
>>> EMPTY = context.empty
>>> Contour = context.contour_cls
>>> Mix = context.mix_cls
>>> Multipolygon = context.multipolygon_cls
>>> Multisegment = context.multisegment_cls
>>> Point = context.point_cls
>>> Polygon = context.polygon_cls
>>> Segment = context.segment_cls
>>> first_square = Contour([Point(0, 0), Point(4, 0), Point(4, 4),
...                         Point(0, 4)])
>>> second_square = Contour([Point(4, 0), Point(8, 0), Point(8, 4),
...                          Point(4, 4)])
>>> third_square = Contour([Point(4, 4), Point(8, 4), Point(8, 8),
...                         Point(4, 8)])
>>> first_inner_square = Contour([Point(1, 1), Point(3, 1), Point(3, 3),
...                               Point(1, 3)])
>>> clockwise_first_inner_square = Contour([Point(1, 1), Point(1, 3),
...                                         Point(3, 3), Point(3, 1)])
>>> (symmetric_subtract_multipolygon_from_multisegment(
...      Multisegment([Segment(Point(0, 0), Point(4, 0)),
...                    Segment(Point(0, 0), Point(4, 4))]),
...      Multipolygon([Polygon(first_square, []),
...                    Polygon(third_square, [])]))
...  == Multipolygon([Polygon(first_square, []),
...                   Polygon(third_square, [])]))
True
>>> (symmetric_subtract_multipolygon_from_multisegment(
...      Multisegment([Segment(Point(0, 0), Point(4, 0)),
...                    Segment(Point(0, 0), Point(4, 4))]),
...      Multipolygon([Polygon(first_square, [clockwise_first_inner_square]),
...                    Polygon(third_square, [])]))
...  == Mix(EMPTY, Segment(Point(1, 1), Point(3, 3)),
...         Multipolygon([Polygon(first_square,
```

(continues on next page)

(continued from previous page)

```
...                                [clockwise_first_inner_square]),
...                        Polygon(third_square, [])])))
True
>>> (symmetric_subtract_multipolygon_from_multisegment(
...      Multisegment([Segment(Point(0, 0), Point(4, 0)),
...                    Segment(Point(0, 0), Point(4, 4))]),
...      Multipolygon([Polygon(first_inner_square, []),
...                    Polygon(second_square, [])]))
...  == Mix(EMPTY, Multisegment([Segment(Point(0, 0), Point(4, 0)),
...                              Segment(Point(0, 0), Point(1, 1)),
...                              Segment(Point(3, 3), Point(4, 4))]),
...         Multipolygon([Polygon(first_inner_square, []),
...                        Polygon(second_square, [])])))
True
```

clipping.planar.**unite_multisegment_with_multipolygon**(*multisegment: ground.hints.Multisegment*, *multipolygon: ground.hints.Multipolygon*, ***, *context: Optional[ground.base.Context] = None*) → Union[ground.hints.Mix, ground.hints.Multipolygon]

Returns union of multisegment with multipolygon.

Time complexity: `O(segments_count * log segments_count)`

Memory complexity: `O(segments_count)`

where `segments_count = start_segments_count + intersections_count`, `start_segments_count = len(multisegment.segments) + multipolygon_edges_count`, `multipolygon_edges_count = len(multipolygon.border.vertices) + sum(len(hole.vertices) for hole in multipolygon.holes)`, `intersections_count` — number of intersections between multisegment and multipolygon edges.

Parameters

- **multisegment** – first operand.
- **multipolygon** – second operand.
- **context** – geometric context.

Returns union of operands.

```
>>> from ground.base import get_context
>>> context = get_context()
>>> EMPTY = context.empty
>>> Contour = context.contour_cls
>>> Mix = context.mix_cls
>>> Multipolygon = context.multipolygon_cls
>>> Multisegment = context.multisegment_cls
>>> Point = context.point_cls
>>> Polygon = context.polygon_cls
>>> Segment = context.segment_cls
>>> first_square = Contour([Point(0, 0), Point(4, 0), Point(4, 4),
...                         Point(0, 4)])
>>> second_square = Contour([Point(4, 0), Point(8, 0), Point(8, 4),
...                          Point(4, 4)])
```

(continues on next page)

(continued from previous page)

```
>>> third_square = Contour([Point(4, 4), Point(8, 4), Point(8, 8),
...                         Point(4, 8)])
>>> first_inner_square = Contour([Point(1, 1), Point(3, 1), Point(3, 3),
...                               Point(1, 3)])
>>> clockwise_first_inner_square = Contour([Point(1, 1), Point(1, 3),
...                                         Point(3, 3), Point(3, 1)])
>>> (unite_multisegment_with_multipolygon(
...      Multisegment([Segment(Point(0, 0), Point(4, 0)),
...                    Segment(Point(0, 0), Point(4, 4))]),
...      Multipolygon([Polygon(first_square, []),
...                    Polygon(third_square, [])]))
...  == Multipolygon([Polygon(first_square, []),
...                   Polygon(third_square, [])]))
True
>>> (unite_multisegment_with_multipolygon(
...      Multisegment([Segment(Point(0, 0), Point(4, 0)),
...                    Segment(Point(0, 0), Point(4, 4))]),
...      Multipolygon([Polygon(first_square, [clockwise_first_inner_square]),
...                    Polygon(third_square, [])]))
...  == Mix(EMPTY, Segment(Point(1, 1), Point(3, 3)),
...         Multipolygon([Polygon(first_square,
...                               [clockwise_first_inner_square]),
...                       Polygon(third_square, [])])))
True
>>> (unite_multisegment_with_multipolygon(
...      Multisegment([Segment(Point(0, 0), Point(4, 0)),
...                    Segment(Point(0, 0), Point(4, 4))]),
...      Multipolygon([Polygon(first_inner_square, []),
...                    Polygon(second_square, [])]))
...  == Mix(EMPTY, Multisegment([Segment(Point(0, 0), Point(4, 0)),
...                              Segment(Point(0, 0), Point(1, 1)),
...                              Segment(Point(3, 3), Point(4, 4))]),
...         Multipolygon([Polygon(first_inner_square, []),
...                       Polygon(second_square, [])])))
True
```

clipping.planar.**complete_intersect_regions**(*first: ground.hints.Contour*, *second: ground.hints.Contour*, *, *context: Optional[ground.base.Context] = None*) → Union[ground.hints.Empty, ground.hints.Mix, ground.hints.Multipoint, ground.hints.Multipolygon, ground.hints.Multisegment, ground.hints.Polygon, ground.hints.Segment]

Returns intersection of regions considering cases with regions touching each other in points/segments.

Time complexity: `O(segments_count * log segments_count)`

Memory complexity: `O(segments_count)`

where `segments_count = edges_count + intersections_count`, `edges_count = len(first.vertices) + len(second.vertices)`, `intersections_count` — number of intersections between regions edges.

Parameters

- **first** – first operand.

- **second** – second operand.
- **context** – geometric context.

Returns intersection of operands.

```
>>> from ground.base import get_context
>>> context = get_context()
>>> EMPTY = context.empty
>>> Contour = context.contour_cls
>>> Mix = context.mix_cls
>>> Multipoint = context.multipoint_cls
>>> Multipolygon = context.multipolygon_cls
>>> Multisegment = context.multisegment_cls
>>> Point = context.point_cls
>>> Polygon = context.polygon_cls
>>> Segment = context.segment_cls
>>> first_square = Contour([Point(0, 0), Point(4, 0), Point(4, 4),
...                          Point(0, 4)])
>>> second_square = Contour([Point(4, 0), Point(8, 0), Point(8, 4),
...                           Point(4, 4)])
>>> third_square = Contour([Point(4, 4), Point(8, 4), Point(8, 8),
...                          Point(4, 8)])
>>> first_inner_square = Contour([Point(1, 1), Point(3, 1), Point(3, 3),
...                                Point(1, 3)])
>>> (complete_intersect_regions(first_inner_square, second_square)
...  is EMPTY)
True
>>> (complete_intersect_regions(first_square, third_square)
...  == Multipoint([Point(4, 4)]))
True
>>> (complete_intersect_regions(first_square, second_square)
...  == Segment(Point(4, 0), Point(4, 4)))
True
>>> (complete_intersect_regions(first_square, first_square)
...  == Polygon(first_square, []))
True
```

`clipping.planar.`**`intersect_regions`**(*first: ground.hints.Contour*, *second: ground.hints.Contour*, ***, *context: Optional[ground.base.Context] = None*) → Union[ground.hints.Empty, ground.hints.Multipolygon, ground.hints.Polygon]

Returns intersection of regions.

Time complexity: `O(segments_count * log segments_count)`

Memory complexity: `O(segments_count)`

where `segments_count = edges_count + intersections_count`, `edges_count = len(first.vertices) + len(second.vertices)`, `intersections_count` — number of intersections between regions edges.

Parameters

- **first** – first operand.
- **second** – second operand.
- **context** – geometric context.

Returns intersection of operands.

```
>>> from ground.base import get_context
>>> context = get_context()
>>> EMPTY = context.empty
>>> Contour = context.contour_cls
>>> Multipolygon = context.multipolygon_cls
>>> Point = context.point_cls
>>> Polygon = context.polygon_cls
>>> first_square = Contour([Point(0, 0), Point(4, 0), Point(4, 4),
...                         Point(0, 4)])
>>> second_square = Contour([Point(4, 0), Point(8, 0), Point(8, 4),
...                          Point(4, 4)])
>>> third_square = Contour([Point(4, 4), Point(8, 4), Point(8, 8),
...                         Point(4, 8)])
>>> first_inner_square = Contour([Point(1, 1), Point(3, 1), Point(3, 3),
...                               Point(1, 3)])
>>> (intersect_regions(first_inner_square, second_square)
...  is intersect_regions(first_square, third_square)
...  is intersect_regions(first_square, second_square)
...  is EMPTY)
True
>>> (intersect_regions(first_square, first_square)
...  == Polygon(first_square, []))
True
```

`clipping.planar.`**`complete_intersect_region_with_multiregion`**(*region: ground.hints.Contour*, *multiregion: Sequence[ground.hints.Contour]*, ***, *context: Optional[ground.base.Context] = None*) → Union[ground.hints.Empty, ground.hints.Mix, ground.hints.Multipoint, ground.hints.Multipolygon, ground.hints.Multisegment, ground.hints.Polygon, ground.hints.Segment]

Returns intersection of region with multiregion considering cases with regions touching each other in points/segments.

Time complexity: `O(segments_count * log segments_count)`

Memory complexity: `O(segments_count)`

where `segments_count = edges_count + intersections_count`, `edges_count = len(region.vertices) + multiregion_edges_count`, `multiregion_edges_count = sum(len(region.vertices) for region in multiregion)`, `intersections_count` — number of intersections between regions edges.

Parameters

- **`region`** – first operand.
- **`multiregion`** – second operand.
- **`context`** – geometric context.

Returns intersection of operands.

```
>>> from ground.base import get_context
>>> context = get_context()
>>> EMPTY = context.empty
>>> Contour = context.contour_cls
>>> Mix = context.mix_cls
>>> Multipoint = context.multipoint_cls
>>> Multipolygon = context.multipolygon_cls
>>> Multisegment = context.multisegment_cls
>>> Point = context.point_cls
>>> Polygon = context.polygon_cls
>>> Segment = context.segment_cls
>>> first_square = Contour([Point(0, 0), Point(4, 0), Point(4, 4),
...                         Point(0, 4)])
>>> second_square = Contour([Point(4, 0), Point(8, 0), Point(8, 4),
...                          Point(4, 4)])
>>> third_square = Contour([Point(4, 4), Point(8, 4), Point(8, 8),
...                         Point(4, 8)])
>>> first_inner_square = Contour([Point(1, 1), Point(3, 1), Point(3, 3),
...                               Point(1, 3)])
>>> second_inner_square = Contour([Point(5, 1), Point(7, 1), Point(7, 3),
...                                Point(5, 3)])
>>> (complete_intersect_region_with_multiregion(
...      second_inner_square, [first_square, third_square])
...  is EMPTY)
True
>>> (complete_intersect_region_with_multiregion(
...      first_square, [second_inner_square, third_square])
...  == Multipoint([Point(4, 4)]))
True
>>> (complete_intersect_region_with_multiregion(
...      second_square, [first_square, third_square])
...  == Multisegment([Segment(Point(4, 0), Point(4, 4)),
...                   Segment(Point(4, 4), Point(8, 4))]))
True
>>> (complete_intersect_region_with_multiregion(
...      first_square, [first_inner_square, second_inner_square])
...  == Polygon(first_inner_square, []))
True
>>> (complete_intersect_region_with_multiregion(
...      first_square, [first_inner_square, third_square])
...  == Mix(Multipoint([Point(4, 4)]), EMPTY,
...         Polygon(first_inner_square, [])))
True
>>> (complete_intersect_region_with_multiregion(
...      first_square, [first_inner_square, second_square])
...  == Mix(EMPTY, Segment(Point(4, 0), Point(4, 4)),
...         Polygon(first_inner_square, [])))
True
```

`clipping.planar.intersect_region_with_multiregion`(*region: ground.hints.Contour*, *multiregion: Sequence[ground.hints.Contour]*, ***, *context: Optional[ground.base.Context] = None*) → Union[ground.hints.Empty, ground.hints.Multipolygon, ground.hints.Polygon]

Returns intersection of region with multiregion.

Time complexity: `O(segments_count * log segments_count)`

Memory complexity: `O(segments_count)`

where `segments_count = edges_count + intersections_count`, `edges_count = len(region.vertices) + multiregion_edges_count`, `multiregion_edges_count = sum(len(region.vertices) for region in multiregion)`, `intersections_count` — number of intersections between regions edges.

Parameters

- **region** – first operand.
- **multiregion** – second operand.
- **context** – geometric context.

Returns intersection of operands.

```
>>> from ground.base import get_context
>>> context = get_context()
>>> EMPTY = context.empty
>>> Contour = context.contour_cls
>>> Multipolygon = context.multipolygon_cls
>>> Point = context.point_cls
>>> Polygon = context.polygon_cls
>>> first_square = Contour([Point(0, 0), Point(4, 0), Point(4, 4),
...                         Point(0, 4)])
>>> second_square = Contour([Point(4, 0), Point(8, 0), Point(8, 4),
...                          Point(4, 4)])
>>> third_square = Contour([Point(4, 4), Point(8, 4), Point(8, 8),
...                         Point(4, 8)])
>>> first_inner_square = Contour([Point(1, 1), Point(3, 1), Point(3, 3),
...                               Point(1, 3)])
>>> second_inner_square = Contour([Point(5, 1), Point(7, 1), Point(7, 3),
...                                Point(5, 3)])
>>> (intersect_region_with_multiregion(
...      second_inner_square, [first_square, third_square])
...  is intersect_region_with_multiregion(
...          first_square, [second_inner_square, third_square])
...  is intersect_region_with_multiregion(
...          second_square, [first_square, third_square])
...  is EMPTY)
True
>>> (intersect_region_with_multiregion(
...      first_square, [first_inner_square, second_inner_square])
...  == intersect_region_with_multiregion(
...          first_square, [first_inner_square, third_square])
...  == intersect_region_with_multiregion(
...          first_square, [first_inner_square, second_square])
```

(continues on next page)

(continued from previous page)

```
...   == Polygon(first_inner_square, []))
True
```

clipping.planar.**complete_intersect_multiregions**(*first: Sequence[ground.hints.Contour], second: Sequence[ground.hints.Contour], *, context: Optional[ground.base.Context] = None*) → Union[ground.hints.Empty, ground.hints.Mix, ground.hints.Multipoint, ground.hints.Multipolygon, ground.hints.Multisegment, ground.hints.Polygon, ground.hints.Segment]

Returns intersection of multiregions considering cases with regions touching each other in points/segments.

Time complexity: `O(segments_count * log segments_count)`

Memory complexity: `O(segments_count)`

where `segments_count = edges_count + intersections_count`, `edges_count = first_edges_count + second_edges_count`, `first_edges_count = sum(len(region.vertices) for region in first)`, `second_edges_count = sum(len(region.vertices) for region in second)`, `intersections_count` — number of intersections between multiregions edges.

Parameters

- **first** – first operand.
- **second** – second operand.
- **context** – geometric context.

Returns intersection of operands.

```
>>> from ground.base import get_context
>>> context = get_context()
>>> EMPTY = context.empty
>>> Contour = context.contour_cls
>>> Mix = context.mix_cls
>>> Multipoint = context.multipoint_cls
>>> Multipolygon = context.multipolygon_cls
>>> Multisegment = context.multisegment_cls
>>> Point = context.point_cls
>>> Polygon = context.polygon_cls
>>> Segment = context.segment_cls
>>> first_square = Contour([Point(0, 0), Point(4, 0), Point(4, 4),
...                         Point(0, 4)])
>>> second_square = Contour([Point(4, 0), Point(8, 0), Point(8, 4),
...                          Point(4, 4)])
>>> third_square = Contour([Point(4, 4), Point(8, 4), Point(8, 8),
...                         Point(4, 8)])
>>> fourth_square = Contour([Point(0, 4), Point(4, 4), Point(4, 8),
...                          Point(0, 8)])
>>> first_inner_square = Contour([Point(1, 1), Point(3, 1), Point(3, 3),
...                               Point(1, 3)])
>>> second_inner_square = Contour([Point(5, 1), Point(7, 1), Point(7, 3),
...                                Point(5, 3)])
>>> third_inner_square = Contour([Point(5, 5), Point(7, 5), Point(7, 7),
...                               Point(5, 7)])
```

(continues on next page)

(continued from previous page)

```
>>> (complete_intersect_multiregions(
...      [first_inner_square, third_inner_square],
...      [second_square, fourth_square])
...  is EMPTY)
True
>>> (complete_intersect_multiregions([first_square, third_square],
...                                  [second_square, fourth_square])
...  == Multisegment([Segment(Point(0, 4), Point(4, 4)),
...                   Segment(Point(4, 0), Point(4, 4)),
...                   Segment(Point(4, 4), Point(8, 4)),
...                   Segment(Point(4, 4), Point(4, 8))]))
True
>>> (complete_intersect_multiregions([first_square, second_inner_square],
...                                  [first_inner_square, second_square])
...  == Mix(EMPTY, Segment(Point(4, 0), Point(4, 4)),
...         Multipolygon([Polygon(first_inner_square, []),
...                       Polygon(second_inner_square, [])])))
True
>>> (complete_intersect_multiregions([first_square, third_inner_square],
...                                  [first_inner_square, third_square])
...  == Mix(Multipoint([Point(4, 4)]), EMPTY,
...         Multipolygon([Polygon(first_inner_square, []),
...                       Polygon(third_inner_square, [])])))
True
>>> (complete_intersect_multiregions(
...      [first_square, third_square],
...      [first_inner_square, third_inner_square])
...  == complete_intersect_multiregions(
...         [first_inner_square, third_inner_square],
...         [first_square, third_square])
...  == complete_intersect_multiregions(
...         [first_square, third_inner_square],
...         [first_inner_square, third_inner_square])
...  == complete_intersect_multiregions(
...         [first_inner_square, third_inner_square],
...         [first_square, third_inner_square])
...  == complete_intersect_multiregions(
...         [first_inner_square, third_inner_square],
...         [first_inner_square, second_inner_square, third_inner_square])
...  == complete_intersect_multiregions(
...         [first_inner_square, second_inner_square, third_inner_square],
...         [first_inner_square, third_inner_square])
...  == Multipolygon([Polygon(first_inner_square, []),
...                   Polygon(third_inner_square, [])]))
True
>>> (complete_intersect_multiregions([first_square, third_square],
...                                  [first_square, third_square])
...  == Multipolygon([Polygon(first_square, []),
...                   Polygon(third_square, [])]))
True
```

`clipping.planar.intersect_multiregions`(*first: Sequence[ground.hints.Contour]*, *second: Sequence[ground.hints.Contour]*, ***, *context: Optional[ground.base.Context] = None*) → Union[ground.hints.Empty, ground.hints.Multipolygon, ground.hints.Polygon]

Returns intersection of multiregions.

Time complexity: `O(segments_count * log segments_count)`

Memory complexity: `O(segments_count)`

where `segments_count = edges_count + intersections_count`, `edges_count = first_edges_count + second_edges_count`, `first_edges_count = sum(len(region.vertices) for region in first)`, `second_edges_count = sum(len(region.vertices) for region in second)`, `intersections_count` — number of intersections between multiregions edges.

Parameters

- **first** – first operand.
- **second** – second operand.
- **context** – geometric context.

Returns intersection of operands.

```
>>> from ground.base import get_context
>>> context = get_context()
>>> EMPTY = context.empty
>>> Contour = context.contour_cls
>>> Multipolygon = context.multipolygon_cls
>>> Point = context.point_cls
>>> Polygon = context.polygon_cls
>>> first_square = Contour([Point(0, 0), Point(4, 0), Point(4, 4),
...                         Point(0, 4)])
>>> second_square = Contour([Point(4, 0), Point(8, 0), Point(8, 4),
...                          Point(4, 4)])
>>> third_square = Contour([Point(4, 4), Point(8, 4), Point(8, 8),
...                         Point(4, 8)])
>>> fourth_square = Contour([Point(0, 4), Point(4, 4), Point(4, 8),
...                          Point(0, 8)])
>>> first_inner_square = Contour([Point(1, 1), Point(3, 1), Point(3, 3),
...                               Point(1, 3)])
>>> second_inner_square = Contour([Point(5, 1), Point(7, 1), Point(7, 3),
...                                Point(5, 3)])
>>> third_inner_square = Contour([Point(5, 5), Point(7, 5), Point(7, 7),
...                               Point(5, 7)])
>>> (intersect_multiregions([first_inner_square, third_inner_square],
...                         [second_square, fourth_square])
...  is intersect_multiregions([first_square, third_square],
...                            [second_square, fourth_square])
...  is EMPTY)
True
>>> (intersect_multiregions([first_square, third_inner_square],
...                         [first_inner_square, third_square])
...  == intersect_multiregions([first_square, third_square],
...                            [first_inner_square, third_inner_square])
```

(continues on next page)

(continued from previous page)

```
...  == intersect_multiregions([first_inner_square, third_inner_square],
...                            [first_square, third_square])
...  == Multipolygon([Polygon(first_inner_square, []),
...                   Polygon(third_inner_square, [])]))
True
>>> (intersect_multiregions([first_square, second_inner_square],
...                         [first_inner_square, second_inner_square])
...  == intersect_multiregions([first_inner_square, second_inner_square],
...                            [first_square, second_inner_square])
...  == intersect_multiregions(
...         [first_inner_square, second_inner_square],
...         [first_inner_square, second_inner_square, third_inner_square])
...  == intersect_multiregions(
...         [first_inner_square, second_inner_square, third_inner_square],
...         [first_inner_square, second_inner_square])
...  == Multipolygon([Polygon(first_inner_square, []),
...                   Polygon(second_inner_square, [])]))
True
>>> (intersect_multiregions([first_square, third_square],
...                         [first_square, third_square])
...  == Multipolygon([Polygon(first_square, []),
...                   Polygon(third_square, [])]))
True
```

clipping.planar.**complete_intersect_polygons**(*first: ground.hints.Polygon*, *second: ground.hints.Polygon*, *, *context: Optional[ground.base.Context] = None*) → Union[ground.hints.Empty, ground.hints.Mix, ground.hints.Multipoint, ground.hints.Multisegment, ground.hints.Polygon, ground.hints.Segment]

Returns intersection of polygons considering cases with polygons touching each other in points/segments.

Time complexity: `O(segments_count * log segments_count)`

Memory complexity: `O(segments_count)`

where `segments_count = edges_count + intersections_count`, `edges_count = first_edges_count + second_edges_count`, `first_edges_count = len(first.border.vertices) + sum(len(hole.vertices) for hole in first.holes)`, `second_edges_count = len(second.border.vertices) + sum(len(hole.vertices) for hole in second.holes)`, `intersections_count` — number of intersections between polygons edges.

Parameters

- **first** – first operand.
- **second** – second operand.
- **context** – geometric context.

Returns intersection of operands.

```
>>> from ground.base import get_context
>>> context = get_context()
>>> EMPTY = context.empty
>>> Contour = context.contour_cls
>>> Mix = context.mix_cls
```

(continues on next page)

(continued from previous page)

```
>>> Multipoint = context.multipoint_cls
>>> Multipolygon = context.multipolygon_cls
>>> Multisegment = context.multisegment_cls
>>> Point = context.point_cls
>>> Polygon = context.polygon_cls
>>> Segment = context.segment_cls
>>> first_square = Contour([Point(0, 0), Point(4, 0), Point(4, 4),
...                         Point(0, 4)])
>>> second_square = Contour([Point(4, 0), Point(8, 0), Point(8, 4),
...                          Point(4, 4)])
>>> third_square = Contour([Point(4, 4), Point(8, 4), Point(8, 8),
...                         Point(4, 8)])
>>> first_inner_square = Contour([Point(1, 1), Point(3, 1), Point(3, 3),
...                               Point(1, 3)])
>>> clockwise_first_inner_square = Contour([Point(1, 1), Point(1, 3),
...                                         Point(3, 3), Point(3, 1)])
>>> complete_intersect_polygons(Polygon(first_inner_square, []),
...                             Polygon(second_square, [])) is EMPTY
True
>>> (complete_intersect_polygons(Polygon(first_square, []),
...                              Polygon(third_square, []))
...  == Multipoint([Point(4, 4)]))
True
>>> (complete_intersect_polygons(Polygon(first_square, []),
...                              Polygon(second_square, []))
... == Segment(Point(4, 0), Point(4, 4)))
True
>>> (complete_intersect_polygons(Polygon(first_inner_square, []),
...                              Polygon(first_square,
...                                      [clockwise_first_inner_square]))
...  == Multisegment([Segment(Point(1, 1), Point(3, 1)),
...                   Segment(Point(1, 1), Point(1, 3)),
...                   Segment(Point(1, 3), Point(3, 3)),
...                   Segment(Point(3, 1), Point(3, 3))]))
True
>>> (complete_intersect_polygons(Polygon(first_square, []),
...                              Polygon(first_inner_square, []))
...  == Polygon(first_inner_square, []))
True
>>> (complete_intersect_polygons(Polygon(first_square, []),
...                              Polygon(first_square, []))
...  == Polygon(first_square, []))
True
>>> (complete_intersect_polygons(Polygon(first_square,
...                                      [clockwise_first_inner_square]),
...                              Polygon(first_square,
...                                      [clockwise_first_inner_square]))
...  == Polygon(first_square, [clockwise_first_inner_square]))
True
```

clipping.planar.**intersect_polygons**(*first: ground.hints.Polygon*, *second: ground.hints.Polygon*, ***, *context: Optional[ground.base.Context] = None*) → Union[ground.hints.Empty, ground.hints.Multipolygon, ground.hints.Polygon]

Returns intersection of polygons.

Time complexity: `O(segments_count * log segments_count)`

Memory complexity: `O(segments_count)`

where `segments_count = edges_count + intersections_count`, `edges_count = first_edges_count + second_edges_count`, `first_edges_count = len(first.border.vertices) + sum(len(hole.vertices) for hole in first.holes)`, `second_edges_count = len(second.border.vertices) + sum(len(hole.vertices) for hole in second.holes)`, `intersections_count` — number of intersections between polygons edges.

Parameters

- **first** – first operand.
- **second** – second operand.
- **context** – geometric context.

Returns intersection of operands.

```
>>> from ground.base import get_context
>>> context = get_context()
>>> EMPTY = context.empty
>>> Contour = context.contour_cls
>>> Point = context.point_cls
>>> Polygon = context.polygon_cls
>>> first_square = Contour([Point(0, 0), Point(4, 0), Point(4, 4),
...                         Point(0, 4)])
>>> second_square = Contour([Point(4, 0), Point(8, 0), Point(8, 4),
...                          Point(4, 4)])
>>> first_inner_square = Contour([Point(1, 1), Point(3, 1), Point(3, 3),
...                               Point(1, 3)])
>>> clockwise_first_inner_square = Contour([Point(1, 1), Point(1, 3),
...                                         Point(3, 3), Point(3, 1)])
>>> (intersect_polygons(Polygon(first_inner_square, []),
...                     Polygon(second_square, []))
...  is intersect_polygons(Polygon(first_square, []),
...                        Polygon(second_square, []))
...  is intersect_polygons(Polygon(first_inner_square, []),
...                        Polygon(first_square,
...                                [clockwise_first_inner_square]))
...  is EMPTY)
True
>>> (intersect_polygons(Polygon(first_square, []),
...                     Polygon(first_inner_square, []))
...  == Polygon(first_inner_square, []))
True
>>> (intersect_polygons(Polygon(first_square,
...                             [clockwise_first_inner_square]),
...                     Polygon(first_square,
...                             [clockwise_first_inner_square]))
```

(continues on next page)

(continued from previous page)

```
...  == Polygon(first_square, [clockwise_first_inner_square]))
True
```

clipping.planar.**subtract_polygons**(*minuend: ground.hints.Polygon*, *subtrahend: ground.hints.Polygon*, **, *context: Optional[ground.base.Context] = None*) → Union[ground.hints.Empty, ground.hints.Multipolygon, ground.hints.Polygon]

Returns difference of polygons.

Time complexity: `O(segments_count * log segments_count)`

Memory complexity: `O(segments_count)`

where `segments_count = edges_count + intersections_count`, `edges_count = minuend_edges_count + subtrahend_edges_count`, `minuend_edges_count = len(minuend.border.vertices) + sum(len(hole.vertices) for hole in minuend.holes)`, `subtrahend_edges_count = len(subtrahend.border.vertices) + sum(len(hole.vertices) for hole in subtrahend.holes)`, `intersections_count` — number of intersections between polygons edges.

Parameters

- **minuend** – polygon to subtract from.
- **subtrahend** – polygon to subtract.
- **context** – geometric context.

Returns difference between minuend and subtrahend.

```
>>> from ground.base import get_context
>>> context = get_context()
>>> EMPTY = context.empty
>>> Contour = context.contour_cls
>>> Multipolygon = context.multipolygon_cls
>>> Point = context.point_cls
>>> Polygon = context.polygon_cls
>>> first_square = Contour([Point(0, 0), Point(4, 0), Point(4, 4),
...                         Point(0, 4)])
>>> second_square = Contour([Point(4, 0), Point(8, 0), Point(8, 4),
...                          Point(4, 4)])
>>> first_inner_square = Contour([Point(1, 1), Point(3, 1), Point(3, 3),
...                               Point(1, 3)])
>>> clockwise_first_inner_square = Contour([Point(1, 1), Point(1, 3),
...                                         Point(3, 3), Point(3, 1)])
>>> (subtract_polygons(Polygon(first_square, []),
...                    Polygon(first_square, []))
...  is subtract_polygons(Polygon(first_inner_square, []),
...                       Polygon(first_square, []))
...  is subtract_polygons(Polygon(first_square,
...                               [clockwise_first_inner_square]),
...                       Polygon(first_square,
...                               [clockwise_first_inner_square]))
...  is EMPTY)
True
>>> (subtract_polygons(Polygon(first_inner_square, []),
```

(continues on next page)

(continued from previous page)

```
...                     Polygon(second_square, []))
...  == subtract_polygons(Polygon(first_inner_square, []),
...                        Polygon(first_square,
...                                [clockwise_first_inner_square]))
...  == Polygon(first_inner_square, []))
True
>>> (subtract_polygons(Polygon(first_square, []),
...                     Polygon(first_inner_square, []))
...  == subtract_polygons(Polygon(first_square, [first_inner_square]),
...                        Polygon(second_square, []))
...  == Polygon(first_square, [clockwise_first_inner_square]))
True
```

clipping.planar.**symmetric_subtract_polygons**(*first: ground.hints.Polygon*, *second: ground.hints.Polygon*, *, *context: Optional[ground.base.Context] = None*) → Union[ground.hints.Empty, ground.hints.Multipolygon, ground.hints.Polygon]

Returns symmetric difference of polygons.

Time complexity: `O(segments_count * log segments_count)`

Memory complexity: `O(segments_count)`

where `segments_count = edges_count + intersections_count`, `edges_count = first_edges_count + second_edges_count`, `first_edges_count = len(first.border.vertices) + sum(len(hole.vertices) for hole in first.holes)`, `second_edges_count = len(second.border.vertices) + sum(len(hole.vertices) for hole in second.holes)`, `intersections_count` — number of intersections between polygons edges.

Parameters

- **first** – first operand.
- **second** – second operand.
- **context** – geometric context.

Returns symmetric difference of operands.

```
>>> from ground.base import get_context
>>> context = get_context()
>>> EMPTY = context.empty
>>> Contour = context.contour_cls
>>> Multipolygon = context.multipolygon_cls
>>> Point = context.point_cls
>>> Polygon = context.polygon_cls
>>> first_square = Contour([Point(0, 0), Point(4, 0), Point(4, 4),
...                          Point(0, 4)])
>>> second_square = Contour([Point(4, 0), Point(8, 0), Point(8, 4),
...                           Point(4, 4)])
>>> third_square = Contour([Point(4, 4), Point(8, 4), Point(8, 8),
...                          Point(4, 8)])
>>> first_inner_square = Contour([Point(1, 1), Point(3, 1), Point(3, 3),
...                                Point(1, 3)])
>>> clockwise_first_inner_square = Contour([Point(1, 1), Point(1, 3),
...                                          Point(3, 3), Point(3, 1)])
```

(continues on next page)

(continued from previous page)

```
>>> (symmetric_subtract_polygons(Polygon(first_square, []),
...                              Polygon(first_square, []))
...  is symmetric_subtract_polygons(
...         Polygon(first_square, [clockwise_first_inner_square]),
...         Polygon(first_square, [clockwise_first_inner_square]))
...  is EMPTY)
True
>>> (symmetric_subtract_polygons(Polygon(first_square,
...                                      [clockwise_first_inner_square]),
...                              Polygon(first_inner_square, []))
...  == Polygon(first_square, []))
True
>>> (symmetric_subtract_polygons(Polygon(first_square, []),
...                              Polygon(second_square, []))
...  == Polygon(Contour([Point(0, 0), Point(8, 0), Point(8, 4),
...                      Point(0, 4)]), []))
True
>>> (symmetric_subtract_polygons(
...      Polygon(first_square, [clockwise_first_inner_square]),
...      Polygon(second_square, []))
...  == Polygon(Contour([Point(0, 0), Point(8, 0), Point(8, 4),
...                      Point(0, 4)]), [clockwise_first_inner_square]))
True
>>> (symmetric_subtract_polygons(Polygon(first_square, []),
...                              Polygon(third_square, []))
...  == Multipolygon([Polygon(first_square, []),
...                   Polygon(third_square, [])]))
True
>>> (symmetric_subtract_polygons(Polygon(first_square,
...                                      [clockwise_first_inner_square]),
...                              Polygon(third_square, []))
...  == Multipolygon([Polygon(first_square,
...                           [clockwise_first_inner_square]),
...                   Polygon(third_square, [])]))
True
```

`clipping.planar.unite_polygons`(*first: ground.hints.Polygon*, *second: ground.hints.Polygon*, ***, *context: Optional[ground.base.Context] = None*) → Union[ground.hints.Multipolygon, ground.hints.Polygon]

Returns union of polygons.

Time complexity: `O(segments_count * log segments_count)`

Memory complexity: `O(segments_count)`

where `segments_count = edges_count + intersections_count`, `edges_count = first_edges_count + second_edges_count`, `first_edges_count = len(first.border.vertices) + sum(len(hole.vertices) for hole in first.holes)`, `second_edges_count = len(second.border.vertices) + sum(len(hole.vertices) for hole in second.holes)`, `intersections_count` — number of intersections between polygons edges.

Parameters

- **`first`** – first operand.

- **second** – second operand.
- **context** – geometric context.

Returns union of operands.

```
>>> from ground.base import get_context
>>> context = get_context()
>>> EMPTY = context.empty
>>> Contour = context.contour_cls
>>> Multipolygon = context.multipolygon_cls
>>> Point = context.point_cls
>>> Polygon = context.polygon_cls
>>> first_square = Contour([Point(0, 0), Point(4, 0), Point(4, 4),
...                         Point(0, 4)])
>>> second_square = Contour([Point(4, 0), Point(8, 0), Point(8, 4),
...                          Point(4, 4)])
>>> third_square = Contour([Point(4, 4), Point(8, 4), Point(8, 8),
...                         Point(4, 8)])
>>> fourth_square = Contour([Point(0, 4), Point(4, 4), Point(4, 8),
...                          Point(0, 8)])
>>> first_inner_square = Contour([Point(1, 1), Point(3, 1), Point(3, 3),
...                               Point(1, 3)])
>>> second_inner_square = Contour([Point(5, 1), Point(7, 1), Point(7, 3),
...                                Point(5, 3)])
>>> third_inner_square = Contour([Point(5, 5), Point(7, 5), Point(7, 7),
...                               Point(5, 7)])
>>> clockwise_first_inner_square = Contour([Point(1, 1), Point(1, 3),
...                                         Point(3, 3), Point(3, 1)])
>>> clockwise_second_inner_square = Contour([Point(5, 1), Point(7, 1),
...                                          Point(7, 3), Point(5, 3)])
>>> clockwise_third_inner_square = Contour([Point(5, 5), Point(5, 7),
...                                         Point(7, 7), Point(7, 5)])
>>> (unite_polygons(Polygon(first_square, []),
...                 Polygon(first_inner_square, []))
...  == unite_polygons(Polygon(first_square,
...                            [clockwise_first_inner_square]),
...                    Polygon(first_inner_square, []))
...  == Polygon(first_square, []))
True
>>> (unite_polygons(Polygon(first_square, []),
...                 Polygon(second_square, []))
...  == Polygon(Contour([Point(0, 0), Point(8, 0), Point(8, 4),
...                      Point(0, 4)]), []))
True
>>> (unite_polygons(Polygon(first_square, [clockwise_first_inner_square]),
...                 Polygon(second_square, []))
...  == Polygon(Contour([Point(0, 0), Point(8, 0), Point(8, 4),
...                      Point(0, 4)]), [clockwise_first_inner_square]))
True
>>> (unite_polygons(Polygon(first_square, []),
...                 Polygon(third_square, []))
...  == Multipolygon([Polygon(first_square, []),
...                   Polygon(third_square, [])]))
```

(continues on next page)

(continued from previous page)

```
True
>>> (unite_polygons(Polygon(first_square, [clockwise_first_inner_square]),
...                 Polygon(third_square, []))
...  == Multipolygon([Polygon(first_square,
...                           [clockwise_first_inner_square]),
...                   Polygon(third_square, [])]))
True
```

clipping.planar.**complete_intersect_polygon_with_multipolygon**(*polygon: ground.hints.Polygon, multipolygon: ground.hints.Multipolygon*, *, *context: Optional[ground.base.Context] = None*) → Union[ground.hints.Empty, ground.hints.Mix, ground.hints.Multipoint, ground.hints.Multipolygon, ground.hints.Multisegment, ground.hints.Polygon, ground.hints.Segment]

Returns intersection of polygon with multipolygon considering cases with polygons touching each other in points/segments.

Time complexity: `O(segments_count * log segments_count)`

Memory complexity: `O(segments_count)`

where `segments_count = edges_count + intersections_count`, `edges_count = polygon_edges_count + multipolygon_edges_count`, `polygon_edges_count = len(polygon.border.vertices) + sum(len(hole.vertices) for hole in polygon.holes)`, `multipolygon_edges_count = sum(len(polygon.border.vertices) + sum(len(hole.vertices) for hole in polygon.holes) for polygon in multipolygon.polygons)`, `intersections_count` — number of intersections between polygons edges.

Parameters

- **polygon** – first operand.
- **multipolygon** – second operand.
- **context** – geometric context.

Returns intersection of operands.

```
>>> from ground.base import get_context
>>> context = get_context()
>>> EMPTY = context.empty
>>> Contour = context.contour_cls
>>> Mix = context.mix_cls
>>> Multipoint = context.multipoint_cls
>>> Multipolygon = context.multipolygon_cls
>>> Multisegment = context.multisegment_cls
>>> Point = context.point_cls
>>> Polygon = context.polygon_cls
>>> Segment = context.segment_cls
```

(continues on next page)

(continued from previous page)

```
>>> first_square = Contour([Point(0, 0), Point(4, 0), Point(4, 4),
...                         Point(0, 4)])
>>> second_square = Contour([Point(4, 0), Point(8, 0), Point(8, 4),
...                          Point(4, 4)])
>>> third_square = Contour([Point(4, 4), Point(8, 4), Point(8, 8),
...                         Point(4, 8)])
>>> fourth_square = Contour([Point(0, 4), Point(4, 4), Point(4, 8),
...                          Point(0, 8)])
>>> first_inner_square = Contour([Point(1, 1), Point(3, 1), Point(3, 3),
...                               Point(1, 3)])
>>> second_inner_square = Contour([Point(5, 1), Point(7, 1), Point(7, 3),
...                                Point(5, 3)])
>>> third_inner_square = Contour([Point(5, 5), Point(7, 5), Point(7, 7),
...                               Point(5, 7)])
>>> clockwise_first_inner_square = Contour([Point(1, 1), Point(1, 3),
...                                         Point(3, 3), Point(3, 1)])
>>> clockwise_second_inner_square = Contour([Point(5, 1), Point(5, 3),
...                                          Point(7, 3), Point(7, 1)])
>>> clockwise_third_inner_square = Contour([Point(5, 5), Point(5, 7),
...                                         Point(7, 7), Point(7, 5)])
>>> (complete_intersect_polygon_with_multipolygon(
...      Polygon(first_inner_square, []),
...      Multipolygon([Polygon(second_square, []),
...                    Polygon(fourth_square, [])]))
...  is EMPTY)
True
>>> (complete_intersect_polygon_with_multipolygon(
...      Polygon(first_square, []),
...      Multipolygon([Polygon(second_inner_square, []),
...                    Polygon(third_square, [])]))
...  == Multipoint([Point(4, 4)]))
True
>>> (complete_intersect_polygon_with_multipolygon(
...      Polygon(first_square, []),
...      Multipolygon([Polygon(second_square, []),
...                    Polygon(fourth_square, [])]))
... == Multisegment([Segment(Point(0, 4), Point(4, 4)),
...                  Segment(Point(4, 0), Point(4, 4))]))
True
>>> (complete_intersect_polygon_with_multipolygon(
...      Polygon(first_inner_square, []),
...      Multipolygon([Polygon(first_square,
...                            [clockwise_first_inner_square]),
...                    Polygon(third_square,
...                            [clockwise_third_inner_square])]))
...  == Multisegment([Segment(Point(1, 1), Point(3, 1)),
...                   Segment(Point(1, 1), Point(1, 3)),
...                   Segment(Point(1, 3), Point(3, 3)),
...                   Segment(Point(3, 1), Point(3, 3))]))
True
>>> (complete_intersect_polygon_with_multipolygon(
...      Polygon(first_square, []),
```

(continues on next page)

(continued from previous page)

```
...     Multipolygon([Polygon(first_inner_square, []),
...                   Polygon(third_inner_square, [])]))
...  == complete_intersect_polygon_with_multipolygon(
...         Polygon(first_inner_square, []),
...         Multipolygon([Polygon(first_square, []),
...                       Polygon(third_square, [])]))
...  == complete_intersect_polygon_with_multipolygon(
...         Polygon(first_square, []),
...         Multipolygon([Polygon(first_inner_square, []),
...                       Polygon(third_inner_square, [])]))
...  == complete_intersect_polygon_with_multipolygon(
...         Polygon(first_inner_square, []),
...         Multipolygon([Polygon(first_square, []),
...                       Polygon(third_inner_square, [])]))
...  == complete_intersect_polygon_with_multipolygon(
...         Polygon(first_inner_square, []),
...         Multipolygon([Polygon(first_inner_square, []),
...                       Polygon(third_inner_square, [])]))
...  == complete_intersect_polygon_with_multipolygon(
...         Polygon(first_inner_square, []),
...         Multipolygon([Polygon(first_inner_square, []),
...                       Polygon(second_inner_square, []),
...                       Polygon(third_inner_square, [])]))
...  == Polygon(first_inner_square, []))
True
>>> (complete_intersect_polygon_with_multipolygon(
...     Polygon(first_square, []),
...     Multipolygon([Polygon(first_square, []),
...                   Polygon(third_square, [])]))
...  == Polygon(first_square, []))
True
>>> (complete_intersect_polygon_with_multipolygon(
...     Polygon(first_square, []),
...     Multipolygon([Polygon(first_inner_square, []),
...                   Polygon(third_square, [])]))
...  == Mix(Multipoint([Point(4, 4)]), EMPTY,
...         Polygon(first_inner_square, [])))
True
>>> (complete_intersect_polygon_with_multipolygon(
...     Polygon(first_square, []),
...     Multipolygon([Polygon(first_inner_square, []),
...                   Polygon(second_square, [])]))
...  == Mix(EMPTY, Segment(Point(4, 0), Point(4, 4)),
...         Polygon(first_inner_square, [])))
True
```

`clipping.planar.`**`intersect_polygon_with_multipolygon`**(*polygon: ground.hints.Polygon*, *multipolygon: ground.hints.Multipolygon*, ***, *context: Optional[ground.base.Context] = None*) → Union[ground.hints.Empty, ground.hints.Multipolygon, ground.hints.Polygon]

Returns intersection of multipolygons.

Time complexity: `O(segments_count * log segments_count)`

Memory complexity: `O(segments_count)`

where `segments_count = edges_count + intersections_count`, `edges_count = polygon_edges_count + multipolygon_edges_count`, `polygon_edges_count = len(polygon.border.vertices) + sum(len(hole.vertices) for hole in polygon.holes)`, `multipolygon_edges_count = sum(len(polygon.border.vertices) + sum(len(hole.vertices) for hole in polygon.holes) for polygon in multipolygon.polygons)`, `intersections_count` — number of intersections between polygons edges.

Parameters

- **polygon** – first operand.
- **multipolygon** – second operand.
- **context** – geometric context.

Returns intersection of operands.

```
>>> from ground.base import get_context
>>> context = get_context()
>>> EMPTY = context.empty
>>> Contour = context.contour_cls
>>> Multipolygon = context.multipolygon_cls
>>> Point = context.point_cls
>>> Polygon = context.polygon_cls
>>> first_square = Contour([Point(0, 0), Point(4, 0), Point(4, 4),
...                         Point(0, 4)])
>>> second_square = Contour([Point(4, 0), Point(8, 0), Point(8, 4),
...                          Point(4, 4)])
>>> third_square = Contour([Point(4, 4), Point(8, 4), Point(8, 8),
...                         Point(4, 8)])
>>> fourth_square = Contour([Point(0, 4), Point(4, 4), Point(4, 8),
...                          Point(0, 8)])
>>> first_inner_square = Contour([Point(1, 1), Point(3, 1), Point(3, 3),
...                               Point(1, 3)])
>>> second_inner_square = Contour([Point(5, 1), Point(7, 1), Point(7, 3),
...                                Point(5, 3)])
>>> third_inner_square = Contour([Point(5, 5), Point(7, 5), Point(7, 7),
...                               Point(5, 7)])
>>> clockwise_first_inner_square = Contour([Point(1, 1), Point(1, 3),
...                                         Point(3, 3), Point(3, 1)])
>>> clockwise_second_inner_square = Contour([Point(5, 1), Point(5, 3),
...                                          Point(7, 3), Point(7, 1)])
>>> clockwise_third_inner_square = Contour([Point(5, 5), Point(5, 7),
...                                         Point(7, 7), Point(7, 5)])
>>> (intersect_polygon_with_multipolygon(
...      Polygon(first_inner_square, []),
...      Multipolygon([Polygon(second_square, []),
...                    Polygon(fourth_square, [])]))
...  is intersect_polygon_with_multipolygon(
...          Polygon(first_square, []),
...          Multipolygon([Polygon(second_square, []),
```

(continues on next page)

(continued from previous page)

```
...                          Polygon(fourth_square, [])]))
...  is intersect_polygon_with_multipolygon(
...          Polygon(first_inner_square, []),
...          Multipolygon([Polygon(first_square,
...                                [clockwise_first_inner_square]),
...                        Polygon(third_square,
...                                [clockwise_third_inner_square])]))
...  is EMPTY)
True
>>> (intersect_polygon_with_multipolygon(
...      Polygon(first_square, []),
...      Multipolygon([Polygon(first_inner_square, []),
...                    Polygon(second_square, [])]))
...  == Polygon(first_inner_square, []))
True
>>> (intersect_polygon_with_multipolygon(
...      Polygon(first_square, []),
...      Multipolygon([Polygon(first_inner_square, []),
...                    Polygon(third_square, [])]))
...  == intersect_polygon_with_multipolygon(
...          Polygon(first_square, []),
...          Multipolygon([Polygon(first_inner_square, []),
...                        Polygon(third_inner_square, [])]))
...  == intersect_polygon_with_multipolygon(
...          Polygon(first_inner_square, []),
...          Multipolygon([Polygon(first_square, []),
...                        Polygon(third_square, [])]))
...  == intersect_polygon_with_multipolygon(
...          Polygon(first_square, []),
...          Multipolygon([Polygon(first_inner_square, []),
...                        Polygon(third_inner_square, [])]))
...  == intersect_polygon_with_multipolygon(
...          Polygon(first_inner_square, []),
...          Multipolygon([Polygon(first_square, []),
...                        Polygon(third_inner_square, [])]))
...  == intersect_polygon_with_multipolygon(
...          Polygon(first_inner_square, []),
...          Multipolygon([Polygon(first_inner_square, []),
...                        Polygon(third_inner_square, [])]))
...  == intersect_polygon_with_multipolygon(
...          Polygon(first_inner_square, []),
...          Multipolygon([Polygon(first_inner_square, []),
...                        Polygon(second_inner_square, []),
...                        Polygon(third_inner_square, [])]))
...  == Polygon(first_inner_square, []))
True
>>> (intersect_polygon_with_multipolygon(
...      Polygon(first_square, []),
...      Multipolygon([Polygon(first_square, []),
...                    Polygon(third_square, [])]))
...  == Polygon(first_square, []))
True
```

`clipping.planar.subtract_multipolygon_from_polygon`(*minuend: ground.hints.Polygon*, *subtrahend: ground.hints.Multipolygon*, ***, *context: Optional[ground.base.Context] = None*) → Union[ground.hints.Empty, ground.hints.Multipolygon, ground.hints.Polygon]

Returns difference of polygon with multipolygon.

Time complexity: `O(segments_count * log segments_count)`

Memory complexity: `O(segments_count)`

where `segments_count = edges_count + intersections_count`, `edges_count = first_edges_count + second_edges_count`, `first_edges_count = len(minuend.border.vertices) + sum(len(hole.vertices) for hole in minuend.holes)`, `second_edges_count = sum(len(polygon.border.vertices) + sum(len(hole.vertices) for hole in polygon.holes) for polygon in subtrahend.polygons)`, `intersections_count` — number of intersections between polygons edges.

Parameters

- **minuend** – polygon to subtract from.
- **subtrahend** – multipolygon to subtract.
- **context** – geometric context.

Returns difference between minuend and subtrahend.

```
>>> from ground.base import get_context
>>> context = get_context()
>>> EMPTY = context.empty
>>> Contour = context.contour_cls
>>> Multipolygon = context.multipolygon_cls
>>> Point = context.point_cls
>>> Polygon = context.polygon_cls
>>> first_square = Contour([Point(0, 0), Point(4, 0), Point(4, 4),
...                         Point(0, 4)])
>>> second_square = Contour([Point(4, 0), Point(8, 0), Point(8, 4),
...                          Point(4, 4)])
>>> third_square = Contour([Point(4, 4), Point(8, 4), Point(8, 8),
...                         Point(4, 8)])
>>> fourth_square = Contour([Point(0, 4), Point(4, 4), Point(4, 8),
...                          Point(0, 8)])
>>> outer_square = Contour([Point(0, 0), Point(8, 0), Point(8, 8),
...                         Point(0, 8)])
>>> first_inner_square = Contour([Point(1, 1), Point(3, 1), Point(3, 3),
...                               Point(1, 3)])
>>> second_inner_square = Contour([Point(5, 1), Point(7, 1), Point(7, 3),
...                                Point(5, 3)])
>>> third_inner_square = Contour([Point(5, 5), Point(7, 5), Point(7, 7),
...                               Point(5, 7)])
>>> clockwise_first_inner_square = Contour([Point(1, 1), Point(1, 3),
...                                         Point(3, 3), Point(3, 1)])
>>> clockwise_second_inner_square = Contour([Point(5, 1), Point(5, 3),
...                                          Point(7, 3), Point(7, 1)])
>>> clockwise_third_inner_square = Contour([Point(5, 5), Point(5, 7),
...                                         Point(7, 7), Point(7, 5)])
```

(continues on next page)

(continued from previous page)

```
>>> (subtract_multipolygon_from_polygon(
...      Polygon(first_square, []),
...      Multipolygon([Polygon(first_square, []),
...                    Polygon(third_square, [])]))
...  is subtract_multipolygon_from_polygon(
...          Polygon(first_inner_square, []),
...          Multipolygon([Polygon(first_square, []),
...                        Polygon(third_square, [])]))
...  is subtract_multipolygon_from_polygon(
...          Polygon(first_inner_square, []),
...          Multipolygon([Polygon(first_inner_square, []),
...                        Polygon(second_inner_square, []),
...                        Polygon(third_inner_square, [])]))
...  is EMPTY)
True
>>> (subtract_multipolygon_from_polygon(
...      Polygon(first_inner_square, []),
...      Multipolygon([Polygon(second_inner_square, []),
...                    Polygon(third_inner_square, [])]))
... == subtract_multipolygon_from_polygon(
...          Polygon(first_inner_square, []),
...          Multipolygon([Polygon(first_square,
...                                [clockwise_first_inner_square]),
...                        Polygon(third_square,
...                                [clockwise_third_inner_square])]))
...  == Polygon(first_inner_square, []))
True
>>> (subtract_multipolygon_from_polygon(
...      Polygon(first_square, []),
...      Multipolygon([Polygon(second_square, []),
...                    Polygon(fourth_square, [])]))
...  == Polygon(first_square, []))
True
>>> (subtract_multipolygon_from_polygon(
...      Polygon(first_square, []),
...      Multipolygon([Polygon(first_inner_square, []),
...                    Polygon(second_square, [])]))
...  == Polygon(first_square, [clockwise_first_inner_square]))
True
>>> (subtract_multipolygon_from_polygon(
...      Polygon(outer_square, []),
...      Multipolygon([Polygon(first_square, []),
...                    Polygon(third_square, [])]))
...  == Multipolygon([Polygon(fourth_square, []),
...                   Polygon(second_square, [])]))
True
>>> (subtract_multipolygon_from_polygon(
...      Polygon(outer_square, []),
...      Multipolygon([Polygon(first_square,
...                            [clockwise_first_inner_square]),
...                    Polygon(third_square,
...                            [clockwise_third_inner_square])]))
```

(continues on next page)

(continued from previous page)

```
...  == Multipolygon([Polygon(fourth_square, []),
...                   Polygon(first_inner_square, []),
...                   Polygon(second_square, []),
...                   Polygon(third_inner_square, [])]))
True
>>> (subtract_multipolygon_from_polygon(
...      Polygon(outer_square, []),
...      Multipolygon([Polygon(first_inner_square, []),
...                    Polygon(second_square, []),
...                    Polygon(third_inner_square, []),
...                    Polygon(fourth_square, [])]))
...  == Multipolygon([Polygon(first_square,
...                           [clockwise_first_inner_square]),
...                   Polygon(third_square,
...                           [clockwise_third_inner_square])]))
True
```

clipping.planar.**subtract_polygon_from_multipolygon**(*minuend: ground.hints.Multipolygon*, *subtrahend: ground.hints.Polygon*, ***, *context: Optional[ground.base.Context] = None*) → Union[ground.hints.Empty, ground.hints.Multipolygon, ground.hints.Polygon]

Returns difference of multipolygon with polygon.

Time complexity: `O(segments_count * log segments_count)`

Memory complexity: `O(segments_count)`

where `segments_count = edges_count + intersections_count`, `edges_count = minuend_edges_count + subtrahend_edges_count`, `minuend_edges_count = sum(len(polygon.border.vertices) + sum(len(hole.vertices) for hole in polygon.holes) for polygon in minuend.polygons)`, `subtrahend_edges_count = len(subtrahend.border.vertices) + sum(len(hole.vertices) for hole in subtrahend.holes)`, `intersections_count` — number of intersections between polygons edges.

Parameters

- **minuend** – multipolygon to subtract from.
- **subtrahend** – polygon to subtract.
- **context** – geometric context.

Returns difference between minuend and subtrahend.

```
>>> from ground.base import get_context
>>> context = get_context()
>>> EMPTY = context.empty
>>> Contour = context.contour_cls
>>> Multipolygon = context.multipolygon_cls
>>> Point = context.point_cls
>>> Polygon = context.polygon_cls
>>> first_square = Contour([Point(0, 0), Point(4, 0), Point(4, 4),
...                         Point(0, 4)])
>>> second_square = Contour([Point(4, 0), Point(8, 0), Point(8, 4),
...                          Point(4, 4)])
```

(continues on next page)

(continued from previous page)

```
>>> third_square = Contour([Point(4, 4), Point(8, 4), Point(8, 8),
...                         Point(4, 8)])
>>> fourth_square = Contour([Point(0, 4), Point(4, 4), Point(4, 8),
...                          Point(0, 8)])
>>> outer_square = Contour([Point(0, 0), Point(8, 0), Point(8, 8),
...                         Point(0, 8)])
>>> first_inner_square = Contour([Point(1, 1), Point(3, 1), Point(3, 3),
...                               Point(1, 3)])
>>> second_inner_square = Contour([Point(5, 1), Point(7, 1), Point(7, 3),
...                                Point(5, 3)])
>>> third_inner_square = Contour([Point(5, 5), Point(7, 5), Point(7, 7),
...                               Point(5, 7)])
>>> clockwise_first_inner_square = Contour([Point(1, 1), Point(1, 3),
...                                         Point(3, 3), Point(3, 1)])
>>> clockwise_second_inner_square = Contour([Point(5, 1), Point(5, 3),
...                                          Point(7, 3), Point(7, 1)])
>>> clockwise_third_inner_square = Contour([Point(5, 5), Point(5, 7),
...                                         Point(7, 7), Point(7, 5)])
>>> (subtract_polygon_from_multipolygon(
...      Multipolygon([Polygon(first_square, []),
...                    Polygon(third_square, [])]),
...      Polygon(outer_square, []))
...  is subtract_polygon_from_multipolygon(
...          Multipolygon([Polygon(first_inner_square, []),
...                        Polygon(second_inner_square, []),
...                        Polygon(third_inner_square, [])]),
...          Polygon(outer_square, []))
...  is EMPTY)
True
>>> (subtract_polygon_from_multipolygon(
...      Multipolygon([Polygon(first_square, []),
...                    Polygon(third_square, [])]),
...      Polygon(third_square, []))
...  == Polygon(first_square, []))
True
>>> (subtract_polygon_from_multipolygon(
...          Multipolygon([Polygon(first_inner_square, []),
...                        Polygon(second_inner_square, [])]),
...          Polygon(third_inner_square, []))
...  == subtract_polygon_from_multipolygon(
...              Multipolygon([Polygon(first_inner_square, []),
...                            Polygon(second_inner_square, []),
...                            Polygon(third_inner_square, [])]),
...              Polygon(third_inner_square, []))
...  == subtract_polygon_from_multipolygon(
...              Multipolygon([Polygon(first_square, []),
...                            Polygon(second_inner_square, [])]),
...              Polygon(first_square, [clockwise_first_inner_square]))
...  == Multipolygon([Polygon(first_inner_square, []),
...                   Polygon(second_inner_square, [])]))
True
>>> (subtract_polygon_from_multipolygon(
```

(continues on next page)

(continued from previous page)

```
...     Multipolygon([Polygon(first_square, []),
...                   Polygon(third_square, [])]),
...     Polygon(second_square, []))
...  == Multipolygon([Polygon(first_square, []),
...                   Polygon(third_square, [])]))
True
>>> (subtract_polygon_from_multipolygon(
...         Multipolygon([Polygon(first_square, []),
...                       Polygon(third_square, [])]),
...         Polygon(first_inner_square, []))
...  == subtract_polygon_from_multipolygon(
...             Multipolygon([Polygon(first_square,
...                                   [clockwise_first_inner_square]),
...                           Polygon(third_square, [])]),
...             Polygon(first_inner_square, []))
...  == Multipolygon([Polygon(first_square,
...                           [clockwise_first_inner_square]),
...                   Polygon(third_square, [])]))
True
```

`clipping.planar.`**`symmetric_subtract_multipolygon_from_polygon`**(*polygon: ground.hints.Polygon*, *multipolygon: ground.hints.Multipolygon*, ***, *context: Optional[ground.base.Context] = None*) → Union[ground.hints.Multipolygon, ground.hints.Polygon]

Returns symmetric difference of polygon with multipolygon.

Time complexity: `O(segments_count * log segments_count)`

Memory complexity: `O(segments_count)`

where `segments_count = edges_count + intersections_count`, `edges_count = polygon_edges_count + multipolygon_edges_count`, `polygon_edges_count = len(polygon.border.vertices) + sum(len(hole.vertices) for hole in polygon.holes)`, `multipolygon_edges_count = sum(len(polygon.border.vertices) + sum(len(hole.vertices) for hole in polygon.holes) for polygon in multipolygon.polygons)`, `intersections_count` — number of intersections between polygons edges.

Parameters

- **`polygon`** – first operand.
- **`multipolygon`** – second operand.
- **`context`** – geometric context.

Returns symmetric difference of operands.

```
>>> from ground.base import get_context
>>> context = get_context()
>>> EMPTY = context.empty
>>> Contour = context.contour_cls
>>> Multipolygon = context.multipolygon_cls
```

(continues on next page)

(continued from previous page)

```
>>> Point = context.point_cls
>>> Polygon = context.polygon_cls
>>> first_square = Contour([Point(0, 0), Point(4, 0), Point(4, 4),
...                         Point(0, 4)])
>>> second_square = Contour([Point(4, 0), Point(8, 0), Point(8, 4),
...                          Point(4, 4)])
>>> third_square = Contour([Point(4, 4), Point(8, 4), Point(8, 8),
...                         Point(4, 8)])
>>> fourth_square = Contour([Point(0, 4), Point(4, 4), Point(4, 8),
...                          Point(0, 8)])
>>> outer_square = Contour([Point(0, 0), Point(8, 0), Point(8, 8),
...                         Point(0, 8)])
>>> first_inner_square = Contour([Point(1, 1), Point(3, 1), Point(3, 3),
...                               Point(1, 3)])
>>> second_inner_square = Contour([Point(5, 1), Point(7, 1), Point(7, 3),
...                                Point(5, 3)])
>>> third_inner_square = Contour([Point(5, 5), Point(7, 5), Point(7, 7),
...                               Point(5, 7)])
>>> clockwise_first_inner_square = Contour([Point(1, 1), Point(1, 3),
...                                         Point(3, 3), Point(3, 1)])
>>> clockwise_second_inner_square = Contour([Point(5, 1), Point(5, 3),
...                                          Point(7, 3), Point(7, 1)])
>>> clockwise_third_inner_square = Contour([Point(5, 5), Point(5, 7),
...                                         Point(7, 7), Point(7, 5)])
>>> (symmetric_subtract_multipolygon_from_polygon(
...      Polygon(first_square, []),
...      Multipolygon([Polygon(first_square, []),
...                    Polygon(third_square, [])]))
...  == Polygon(third_square, []))
True
>>> (symmetric_subtract_multipolygon_from_polygon(
...          Polygon(first_square, []),
...          Multipolygon([Polygon(second_square, []),
...                        Polygon(fourth_square, [])]))
...  == Polygon(Contour([Point(0, 0), Point(8, 0), Point(8, 4),
...                      Point(4, 4), Point(4, 8), Point(0, 8)]), []))
True
>>> (symmetric_subtract_multipolygon_from_polygon(
...      Polygon(outer_square, []),
...      Multipolygon([Polygon(first_inner_square, []),
...                    Polygon(second_inner_square, []),
...                    Polygon(third_inner_square, [])]))
...  == Polygon(outer_square, [clockwise_first_inner_square,
...                            clockwise_second_inner_square,
...                            clockwise_third_inner_square]))
True
>>> (symmetric_subtract_multipolygon_from_polygon(
...          Polygon(first_inner_square, []),
...          Multipolygon([Polygon(first_inner_square, []),
...                        Polygon(second_inner_square, []),
...                        Polygon(third_inner_square, [])]))
...  == Multipolygon([Polygon(second_inner_square, []),
```

(continues on next page)

(continued from previous page)

```
...                     Polygon(third_inner_square, [])]))
True
>>> (symmetric_subtract_multipolygon_from_polygon(
...      Polygon(first_inner_square, []),
...      Multipolygon([Polygon(first_square,
...                            [clockwise_first_inner_square]),
...                    Polygon(third_square, [])]))
... == symmetric_subtract_multipolygon_from_polygon(
...          Polygon(outer_square, []),
...          Multipolygon([Polygon(second_square, []),
...                        Polygon(fourth_square, [])]))
... == Multipolygon([Polygon(first_square, []),
...                  Polygon(third_square, [])]))
True
>>> (symmetric_subtract_multipolygon_from_polygon(
...      Polygon(first_inner_square, []),
...      Multipolygon([Polygon(second_inner_square, []),
...                    Polygon(third_inner_square, [])]))
...  == Multipolygon([Polygon(first_inner_square, []),
...                   Polygon(second_inner_square, []),
...                   Polygon(third_inner_square, [])]))
True
>>> (symmetric_subtract_multipolygon_from_polygon(
...      Polygon(outer_square, []),
...      Multipolygon([Polygon(first_square,
...                            [clockwise_first_inner_square]),
...                    Polygon(third_square,
...                            [clockwise_third_inner_square])]))
...  == symmetric_subtract_multipolygon_from_polygon(
...          Polygon(outer_square, [clockwise_first_inner_square,
...                                 clockwise_third_inner_square]),
...          Multipolygon([Polygon(first_square, []),
...                        Polygon(third_square, [])]))
...  == Multipolygon([Polygon(fourth_square, []),
...                   Polygon(first_inner_square, []),
...                   Polygon(second_square, []),
...                   Polygon(third_inner_square, [])]))
True
>>> (symmetric_subtract_multipolygon_from_polygon(
...      Polygon(first_square, []),
...      Multipolygon([Polygon(first_inner_square, []),
...                    Polygon(third_inner_square, [])]))
...  == symmetric_subtract_multipolygon_from_polygon(
...          Polygon(first_inner_square, []),
...          Multipolygon([Polygon(first_square, []),
...                        Polygon(third_inner_square, [])]))
...  == Multipolygon([Polygon(first_square,
...                           [clockwise_first_inner_square]),
...                   Polygon(third_inner_square, [])]))
True
>>> (symmetric_subtract_multipolygon_from_polygon(
...      Polygon(outer_square, []),
```

(continues on next page)

(continued from previous page)

```
...     Multipolygon([Polygon(first_inner_square, []),
...                   Polygon(second_square, []),
...                   Polygon(third_inner_square, []),
...                   Polygon(fourth_square, [])]))
...  == Multipolygon([Polygon(first_square,
...                           [clockwise_first_inner_square]),
...                   Polygon(third_square,
...                           [clockwise_third_inner_square])]))
True
```

clipping.planar.**unite_polygon_with_multipolygon**(*polygon: ground.hints.Polygon*, *multipolygon: ground.hints.Multipolygon*, ***, *context: Optional[ground.base.Context] = None*) → Union[ground.hints.Multipolygon, ground.hints.Polygon]

Returns union of polygon with multipolygon.

Time complexity: `O(segments_count * log segments_count)`

Memory complexity: `O(segments_count)`

where `segments_count = edges_count + intersections_count`, `edges_count = polygon_edges_count + multipolygon_edges_count`, `polygon_edges_count = len(polygon.border.vertices) + sum(len(hole.vertices) for hole in polygon.holes)`, `multipolygon_edges_count = sum(len(polygon.border.vertices) + sum(len(hole.vertices) for hole in polygon.holes) for polygon in multipolygon.polygons)`, `intersections_count` — number of intersections between polygons edges.

Parameters

- **polygon** – first operand.
- **multipolygon** – second operand.
- **context** – geometric context.

Returns union of operands.

```
>>> from ground.base import get_context
>>> context = get_context()
>>> EMPTY = context.empty
>>> Contour = context.contour_cls
>>> Multipolygon = context.multipolygon_cls
>>> Point = context.point_cls
>>> Polygon = context.polygon_cls
>>> first_square = Contour([Point(0, 0), Point(4, 0), Point(4, 4),
...                         Point(0, 4)])
>>> second_square = Contour([Point(4, 0), Point(8, 0), Point(8, 4),
...                          Point(4, 4)])
>>> third_square = Contour([Point(4, 4), Point(8, 4), Point(8, 8),
...                         Point(4, 8)])
>>> fourth_square = Contour([Point(0, 4), Point(4, 4), Point(4, 8),
...                          Point(0, 8)])
>>> outer_square = Contour([Point(0, 0), Point(8, 0), Point(8, 8),
...                         Point(0, 8)])
>>> first_inner_square = Contour([Point(1, 1), Point(3, 1), Point(3, 3),
```

(continues on next page)

(continued from previous page)

```
...                                 Point(1, 3)])
>>> second_inner_square = Contour([Point(5, 1), Point(7, 1), Point(7, 3),
...                                  Point(5, 3)])
>>> third_inner_square = Contour([Point(5, 5), Point(7, 5), Point(7, 7),
...                                 Point(5, 7)])
>>> clockwise_first_inner_square = Contour([Point(1, 1), Point(1, 3),
...                                           Point(3, 3), Point(3, 1)])
>>> clockwise_second_inner_square = Contour([Point(5, 1), Point(7, 1),
...                                            Point(7, 3), Point(5, 3)])
>>> clockwise_third_inner_square = Contour([Point(5, 5), Point(5, 7),
...                                           Point(7, 7), Point(7, 5)])
>>> (unite_polygon_with_multipolygon(
...          Polygon(first_square, []),
...          Multipolygon([Polygon(second_square, []),
...                        Polygon(fourth_square, [])]))
...  == Polygon(Contour([Point(0, 0), Point(8, 0), Point(8, 4),
...                      Point(4, 4), Point(4, 8), Point(0, 8)]), []))
True
>>> (unite_polygon_with_multipolygon(
...      Polygon(outer_square, []),
...      Multipolygon([Polygon(first_inner_square, []),
...                    Polygon(second_inner_square, []),
...                    Polygon(third_inner_square, [])]))
...  == unite_polygon_with_multipolygon(
...          Polygon(outer_square, []),
...          Multipolygon([Polygon(first_square,
...                                [clockwise_first_inner_square]),
...                        Polygon(third_square,
...                                [clockwise_third_inner_square])]))
...  == Polygon(outer_square, []))
True
>>> (unite_polygon_with_multipolygon(
...      Polygon(first_inner_square, []),
...      Multipolygon([Polygon(first_square,
...                            [clockwise_first_inner_square]),
...                    Polygon(third_square, [])]))
... == Multipolygon([Polygon(first_square, []),
...                  Polygon(third_square, [])]))
True
>>> (unite_polygon_with_multipolygon(
...      Polygon(first_inner_square, []),
...      Multipolygon([Polygon(second_inner_square, []),
...                    Polygon(third_inner_square, [])]))
...  == unite_polygon_with_multipolygon(
...          Polygon(first_inner_square, []),
...          Multipolygon([Polygon(first_inner_square, []),
...                        Polygon(second_inner_square, []),
...                        Polygon(third_inner_square, [])]))
...  == Multipolygon([Polygon(first_inner_square, []),
...                   Polygon(second_inner_square, []),
...                   Polygon(third_inner_square, [])]))
True
```

(continues on next page)

(continued from previous page)

```
>>> (unite_polygon_with_multipolygon(
...      Polygon(first_square, [clockwise_first_inner_square]),
...      Multipolygon([Polygon(first_square,
...                            [clockwise_first_inner_square]),
...                    Polygon(third_square,
...                            [clockwise_third_inner_square])]))
...  == Multipolygon([Polygon(first_square,
...                           [clockwise_first_inner_square]),
...                   Polygon(third_square,
...                           [clockwise_third_inner_square])]))
True
```

clipping.planar.**complete_intersect_multipolygons**(*first: ground.hints.Multipolygon*, *second: ground.hints.Multipolygon*, ***, *context: Optional[ground.base.Context] = None*) → Union[ground.hints.Empty, ground.hints.Mix, ground.hints.Multipoint, ground.hints.Multipolygon, ground.hints.Multisegment, ground.hints.Polygon, ground.hints.Segment]

Returns intersection of multipolygons considering cases with polygons touching each other in points/segments.

Time complexity: `O(segments_count * log segments_count)`

Memory complexity: `O(segments_count)`

where `segments_count = edges_count + intersections_count`, `edges_count = first_edges_count + second_edges_count`, `first_edges_count = sum(len(polygon.border.vertices) + sum(len(hole.vertices) for hole in polygon.holes) for polygon in first.polygons)`, `second_edges_count = sum(len(polygon.border.vertices) + sum(len(hole.vertices) for hole in polygon.holes) for polygon in second.polygons)`, `intersections_count` — number of intersections between multipolygons edges.

Parameters

- **first** – first operand.
- **second** – second operand.
- **context** – geometric context.

Returns intersection of operands.

```
>>> from ground.base import get_context
>>> context = get_context()
>>> EMPTY = context.empty
>>> Contour = context.contour_cls
>>> Mix = context.mix_cls
>>> Multipoint = context.multipoint_cls
>>> Multipolygon = context.multipolygon_cls
>>> Multisegment = context.multisegment_cls
>>> Point = context.point_cls
>>> Polygon = context.polygon_cls
>>> Segment = context.segment_cls
>>> first_square = Contour([Point(0, 0), Point(4, 0), Point(4, 4),
...                         Point(0, 4)])
>>> second_square = Contour([Point(4, 0), Point(8, 0), Point(8, 4),
```

(continues on next page)

(continued from previous page)

```
...                            Point(4, 4)])
>>> third_square = Contour([Point(4, 4), Point(8, 4), Point(8, 8),
...                           Point(4, 8)])
>>> fourth_square = Contour([Point(0, 4), Point(4, 4), Point(4, 8),
...                            Point(0, 8)])
>>> first_inner_square = Contour([Point(1, 1), Point(3, 1), Point(3, 3),
...                                 Point(1, 3)])
>>> second_inner_square = Contour([Point(5, 1), Point(7, 1), Point(7, 3),
...                                  Point(5, 3)])
>>> third_inner_square = Contour([Point(5, 5), Point(7, 5), Point(7, 7),
...                                 Point(5, 7)])
>>> clockwise_first_inner_square = Contour([Point(1, 1), Point(1, 3),
...                                           Point(3, 3), Point(3, 1)])
>>> clockwise_second_inner_square = Contour([Point(5, 1), Point(5, 3),
...                                            Point(7, 3), Point(7, 1)])
>>> clockwise_third_inner_square = Contour([Point(5, 5), Point(5, 7),
...                                           Point(7, 7), Point(7, 5)])
>>> (complete_intersect_multipolygons(
...      Multipolygon([Polygon(first_inner_square, []),
...                    Polygon(third_inner_square, [])]),
...      Multipolygon([Polygon(second_square, []),
...                    Polygon(fourth_square, [])]))
...  is EMPTY)
True
>>> (complete_intersect_multipolygons(
...      Multipolygon([Polygon(first_inner_square, []),
...                    Polygon(second_square, [])]),
...      Multipolygon([Polygon(third_inner_square, []),
...                    Polygon(fourth_square, [])]))
...  == Multipoint([Point(4, 4)]))
True
>>> (complete_intersect_multipolygons(
...      Multipolygon([Polygon(first_square, []),
...                    Polygon(third_square, [])]),
...      Multipolygon([Polygon(second_square, []),
...                    Polygon(fourth_square, [])]))
... == Multisegment([Segment(Point(0, 4), Point(4, 4)),
...                  Segment(Point(4, 0), Point(4, 4)),
...                  Segment(Point(4, 4), Point(8, 4)),
...                  Segment(Point(4, 4), Point(4, 8))]))
True
>>> (complete_intersect_multipolygons(
...      Multipolygon([Polygon(first_inner_square, []),
...                    Polygon(third_inner_square, [])]),
...      Multipolygon([Polygon(first_square,
...                            [clockwise_first_inner_square]),
...                    Polygon(third_square,
...                            [clockwise_third_inner_square])]))
...  == Multisegment([Segment(Point(1, 1), Point(3, 1)),
...                   Segment(Point(1, 1), Point(1, 3)),
...                   Segment(Point(1, 3), Point(3, 3)),
...                   Segment(Point(3, 1), Point(3, 3)),
```

(continues on next page)

(continued from previous page)

```
...                     Segment(Point(5, 5), Point(7, 5)),
...                     Segment(Point(5, 5), Point(5, 7)),
...                     Segment(Point(5, 7), Point(7, 7)),
...                     Segment(Point(7, 5), Point(7, 7))]))
True
>>> (complete_intersect_multipolygons(
...      Multipolygon([Polygon(first_square,
...                             [clockwise_first_inner_square]),
...                    Polygon(second_inner_square, [])]),
...      Multipolygon([Polygon(first_inner_square, []),
...                    Polygon(second_square,
...                             [clockwise_second_inner_square])]))
...  == Multisegment([Segment(Point(1, 1), Point(3, 1)),
...                     Segment(Point(1, 1), Point(1, 3)),
...                     Segment(Point(1, 3), Point(3, 3)),
...                     Segment(Point(3, 1), Point(3, 3)),
...                     Segment(Point(4, 0), Point(4, 4)),
...                     Segment(Point(5, 1), Point(7, 1)),
...                     Segment(Point(5, 1), Point(5, 3)),
...                     Segment(Point(5, 3), Point(7, 3)),
...                     Segment(Point(7, 1), Point(7, 3))]))
True
>>> (complete_intersect_multipolygons(
...      Multipolygon([Polygon(first_square, []),
...                    Polygon(third_square, [])]),
...      Multipolygon([Polygon(first_square, []),
...                    Polygon(third_square, [])]))
...  == Multipolygon([Polygon(first_square, []),
...                   Polygon(third_square, [])]))
True
>>> (complete_intersect_multipolygons(
...      Multipolygon([Polygon(first_square, []),
...                    Polygon(third_square, [])]),
...      Multipolygon([Polygon(first_inner_square, []),
...                    Polygon(third_inner_square, [])]))
...  == complete_intersect_multipolygons(
...          Multipolygon([Polygon(first_inner_square, []),
...                        Polygon(third_inner_square, [])]),
...          Multipolygon([Polygon(first_square, []),
...                        Polygon(third_square, [])]))
...  == complete_intersect_multipolygons(
...          Multipolygon([Polygon(first_square, []),
...                        Polygon(third_inner_square, [])]),
...          Multipolygon([Polygon(first_inner_square, []),
...                        Polygon(third_inner_square, [])]))
...  == complete_intersect_multipolygons(
...          Multipolygon([Polygon(first_inner_square, []),
...                        Polygon(third_inner_square, [])]),
...          Multipolygon([Polygon(first_square, []),
...                        Polygon(third_inner_square, [])]))
...  == complete_intersect_multipolygons(
...          Multipolygon([Polygon(first_inner_square, []),
```

(continues on next page)

(continued from previous page)

```
...                         Polygon(second_inner_square, []),
...                         Polygon(third_inner_square, [])]),
...           Multipolygon([Polygon(first_inner_square, []),
...                         Polygon(third_inner_square, [])]))
...  == complete_intersect_multipolygons(
...           Multipolygon([Polygon(first_inner_square, []),
...                         Polygon(third_inner_square, [])]),
...           Multipolygon([Polygon(first_inner_square, []),
...                         Polygon(second_inner_square, []),
...                         Polygon(third_inner_square, [])]))
...  == Multipolygon([Polygon(first_inner_square, []),
...                    Polygon(third_inner_square, [])]))
True
>>> (complete_intersect_multipolygons(
...       Multipolygon([Polygon(first_square, []),
...                     Polygon(second_inner_square, [])]),
...       Multipolygon([Polygon(first_inner_square, []),
...                     Polygon(second_square,
...                             [clockwise_second_inner_square])]))
...  == Mix(EMPTY, Multisegment([Segment(Point(4, 0), Point(4, 4)),
...                              Segment(Point(5, 1), Point(7, 1)),
...                              Segment(Point(5, 1), Point(5, 3)),
...                              Segment(Point(5, 3), Point(7, 3)),
...                              Segment(Point(7, 1), Point(7, 3))]),
...         Polygon(first_inner_square, [])))
True
>>> (complete_intersect_multipolygons(
...       Multipolygon([Polygon(first_square, []),
...                     Polygon(third_inner_square, [])]),
...       Multipolygon([Polygon(first_inner_square, []),
...                     Polygon(third_square, [])]))
...  == Mix(Multipoint([Point(4, 4)]), EMPTY,
...         Multipolygon([Polygon(first_inner_square, []),
...                       Polygon(third_inner_square, [])])))
True
>>> (complete_intersect_multipolygons(
...       Multipolygon([Polygon(first_square, []),
...                     Polygon(second_inner_square, [])]),
...       Multipolygon([Polygon(first_inner_square, []),
...                     Polygon(second_square, [])]))
...  == Mix(EMPTY, Segment(Point(4, 0), Point(4, 4)),
...         Multipolygon([Polygon(first_inner_square, []),
...                       Polygon(second_inner_square, [])])))
True
```

`clipping.planar.`**`intersect_multipolygons`**(*first: ground.hints.Multipolygon*, *second: ground.hints.Multipolygon*, ***, *context: Optional[ground.base.Context] = None*) → Union[ground.hints.Empty, ground.hints.Multipolygon, ground.hints.Polygon]

Returns intersection of multipolygons.

Time complexity: `O(segments_count * log segments_count)`

Memory complexity: `O(segments_count)`

where `segments_count = edges_count + intersections_count`, `edges_count = first_edges_count + second_edges_count`, `first_edges_count = sum(len(polygon.border.vertices) + sum(len(hole.vertices) for hole in polygon.holes) for polygon in first.polygons)`, `second_edges_count = sum(len(polygon.border.vertices) + sum(len(hole.vertices) for hole in polygon.holes) for polygon in second.polygons)`, `intersections_count` — number of intersections between multipolygons edges.

Parameters

- **first** – first operand.
- **second** – second operand.
- **context** – geometric context.

Returns intersection of operands.

```
>>> from ground.base import get_context
>>> context = get_context()
>>> EMPTY = context.empty
>>> Contour = context.contour_cls
>>> Multipolygon = context.multipolygon_cls
>>> Point = context.point_cls
>>> Polygon = context.polygon_cls
>>> first_square = Contour([Point(0, 0), Point(4, 0), Point(4, 4),
...                         Point(0, 4)])
>>> second_square = Contour([Point(4, 0), Point(8, 0), Point(8, 4),
...                          Point(4, 4)])
>>> third_square = Contour([Point(4, 4), Point(8, 4), Point(8, 8),
...                         Point(4, 8)])
>>> fourth_square = Contour([Point(0, 4), Point(4, 4), Point(4, 8),
...                          Point(0, 8)])
>>> first_inner_square = Contour([Point(1, 1), Point(3, 1), Point(3, 3),
...                               Point(1, 3)])
>>> second_inner_square = Contour([Point(5, 1), Point(7, 1), Point(7, 3),
...                                Point(5, 3)])
>>> third_inner_square = Contour([Point(5, 5), Point(7, 5), Point(7, 7),
...                               Point(5, 7)])
>>> clockwise_first_inner_square = Contour([Point(1, 1), Point(1, 3),
...                                         Point(3, 3), Point(3, 1)])
>>> clockwise_second_inner_square = Contour([Point(5, 1), Point(7, 1),
...                                          Point(7, 3), Point(5, 3)])
>>> clockwise_third_inner_square = Contour([Point(5, 5), Point(5, 7),
...                                         Point(7, 7), Point(7, 5)])
>>> (intersect_multipolygons(
...      Multipolygon([Polygon(first_inner_square, []),
...                    Polygon(third_inner_square, [])]),
...      Multipolygon([Polygon(second_square, []),
...                    Polygon(fourth_square, [])]))
...  is intersect_multipolygons(
...          Multipolygon([Polygon(first_square, []),
...                        Polygon(third_square, [])]),
...          Multipolygon([Polygon(second_square, []),
...                        Polygon(fourth_square, [])]))
```

(continues on next page)

(continued from previous page)

```
...   is intersect_multipolygons(
...           Multipolygon([Polygon(first_inner_square, []),
...                         Polygon(third_inner_square, [])]),
...           Multipolygon([Polygon(first_square,
...                                 [clockwise_first_inner_square]),
...                         Polygon(third_square,
...                                 [clockwise_third_inner_square])]))
...   is EMPTY)
True
>>> (intersect_multipolygons(
...       Multipolygon([Polygon(first_square, []),
...                     Polygon(second_inner_square, [])]),
...       Multipolygon([Polygon(first_inner_square, []),
...                     Polygon(second_square, [])]))
...   == Multipolygon([Polygon(first_inner_square, []),
...                    Polygon(second_inner_square, [])]))
True
>>> (intersect_multipolygons(
...       Multipolygon([Polygon(first_square, []),
...                     Polygon(third_inner_square, [])]),
...       Multipolygon([Polygon(first_inner_square, []),
...                     Polygon(third_square, [])]))
...   == intersect_multipolygons(
...           Multipolygon([Polygon(first_square, []),
...                         Polygon(third_square, [])]),
...           Multipolygon([Polygon(first_inner_square, []),
...                         Polygon(third_inner_square, [])]))
...   == intersect_multipolygons(
...           Multipolygon([Polygon(first_inner_square, []),
...                         Polygon(third_inner_square, [])]),
...           Multipolygon([Polygon(first_square, []),
...                         Polygon(third_square, [])]))
...   == intersect_multipolygons(
...           Multipolygon([Polygon(first_square, []),
...                         Polygon(third_inner_square, [])]),
...           Multipolygon([Polygon(first_inner_square, []),
...                         Polygon(third_inner_square, [])]))
...   == intersect_multipolygons(
...           Multipolygon([Polygon(first_inner_square, []),
...                         Polygon(third_inner_square, [])]),
...           Multipolygon([Polygon(first_square, []),
...                         Polygon(third_inner_square, [])]))
...   == intersect_multipolygons(
...           Multipolygon([Polygon(first_inner_square, []),
...                         Polygon(second_inner_square, []),
...                         Polygon(third_inner_square, [])]),
...           Multipolygon([Polygon(first_inner_square, []),
...                         Polygon(third_inner_square, [])]))
...   == intersect_multipolygons(
...           Multipolygon([Polygon(first_inner_square, []),
...                         Polygon(third_inner_square, [])]),
...           Multipolygon([Polygon(first_inner_square, []),
```

(continues on next page)

(continued from previous page)

```
...                         Polygon(second_inner_square, []),
...                         Polygon(third_inner_square, [])]))
...  == Multipolygon([Polygon(first_inner_square, []),
...                    Polygon(third_inner_square, [])]))
True
>>> (intersect_multipolygons(Multipolygon([Polygon(first_square, []),
...                                         Polygon(third_square, [])]),
...                          Multipolygon([Polygon(first_square, []),
...                                        Polygon(third_square, [])]))
...  == Multipolygon([Polygon(first_square, []),
...                    Polygon(third_square, [])]))
True
```

`clipping.planar.`**`subtract_multipolygons`**(*minuend: ground.hints.Multipolygon*, *subtrahend: ground.hints.Multipolygon*, ***, *context: Optional[ground.base.Context] = None*) → Union[ground.hints.Empty, ground.hints.Multipolygon, ground.hints.Polygon]

Returns difference of multipolygons.

Time complexity: `O(segments_count * log segments_count)`

Memory complexity: `O(segments_count)`

where `segments_count = edges_count + intersections_count`, `edges_count = minuend_edges_count + subtrahend_edges_count`, `minuend_edges_count = sum(len(polygon.border.vertices) + sum(len(hole.vertices) for hole in polygon.holes) for polygon in minuend.polygons)`, `subtrahend_edges_count = sum(len(polygon.border.vertices) + sum(len(hole.vertices) for hole in polygon.holes) for polygon in subtrahend.polygons)`, `intersections_count` — number of intersections between multipolygons edges.

Parameters

- **minuend** – multipolygon to subtract from.
- **subtrahend** – multipolygon to subtract.
- **context** – geometric context.

Returns difference between minuend and subtrahend.

```
>>> from ground.base import get_context
>>> context = get_context()
>>> EMPTY = context.empty
>>> Contour = context.contour_cls
>>> Multipolygon = context.multipolygon_cls
>>> Point = context.point_cls
>>> Polygon = context.polygon_cls
>>> first_square = Contour([Point(0, 0), Point(4, 0), Point(4, 4),
...                         Point(0, 4)])
>>> second_square = Contour([Point(4, 0), Point(8, 0), Point(8, 4),
...                          Point(4, 4)])
>>> third_square = Contour([Point(4, 4), Point(8, 4), Point(8, 8),
...                         Point(4, 8)])
>>> fourth_square = Contour([Point(0, 4), Point(4, 4), Point(4, 8),
...                          Point(0, 8)])
```

(continues on next page)

(continued from previous page)

```
>>> first_inner_square = Contour([Point(1, 1), Point(3, 1), Point(3, 3),
...                                  Point(1, 3)])
>>> second_inner_square = Contour([Point(5, 1), Point(7, 1), Point(7, 3),
...                                   Point(5, 3)])
>>> third_inner_square = Contour([Point(5, 5), Point(7, 5), Point(7, 7),
...                                  Point(5, 7)])
>>> clockwise_first_inner_square = Contour([Point(1, 1), Point(1, 3),
...                                            Point(3, 3), Point(3, 1)])
>>> clockwise_second_inner_square = Contour([Point(5, 1), Point(5, 3),
...                                             Point(7, 3), Point(7, 1)])
>>> clockwise_third_inner_square = Contour([Point(5, 5), Point(5, 7),
...                                            Point(7, 7), Point(7, 5)])
>>> (subtract_multipolygons(Multipolygon([Polygon(first_square, []),
...                                         Polygon(third_square, [])]),
...                            Multipolygon([Polygon(first_square, []),
...                                         Polygon(third_square, [])]))
...  is subtract_multipolygons(
...         Multipolygon([Polygon(first_inner_square, []),
...                       Polygon(third_inner_square, [])]),
...         Multipolygon([Polygon(first_square, []),
...                       Polygon(third_square, [])]))
...  is subtract_multipolygons(
...         Multipolygon([Polygon(first_inner_square, []),
...                       Polygon(third_inner_square, [])]),
...         Multipolygon([Polygon(first_square, []),
...                       Polygon(third_inner_square, [])]))
...  is subtract_multipolygons(
...         Multipolygon([Polygon(first_inner_square, []),
...                       Polygon(third_inner_square, [])]),
...         Multipolygon([Polygon(first_inner_square, []),
...                       Polygon(second_inner_square, []),
...                       Polygon(third_inner_square, [])]))
...  is EMPTY)
True
>>> (subtract_multipolygons(
...      Multipolygon([Polygon(first_inner_square, []),
...                    Polygon(second_inner_square, []),
...                    Polygon(third_inner_square, [])]),
...      Multipolygon([Polygon(first_inner_square, []),
...                    Polygon(third_inner_square, [])]))
...  == Polygon(second_inner_square, []))
True
>>> (subtract_multipolygons(
...      Multipolygon([Polygon(first_square, []),
...                    Polygon(second_inner_square, [])]),
...      Multipolygon([Polygon(first_inner_square, []),
...                    Polygon(second_square, [])]))
...  == subtract_multipolygons(
...         Multipolygon([Polygon(first_square, []),
...                       Polygon(third_inner_square, [])]),
...         Multipolygon([Polygon(first_inner_square, []),
...                       Polygon(third_square, [])]))
```

(continues on next page)

(continued from previous page)

```
...  == subtract_multipolygons(
...         Multipolygon([Polygon(first_square, []),
...                       Polygon(third_inner_square, [])]),
...         Multipolygon([Polygon(first_inner_square, []),
...                       Polygon(third_inner_square, [])]))
...  == Polygon(first_square, [clockwise_first_inner_square]))
True
>>> (subtract_multipolygons(
...      Multipolygon([Polygon(first_square, []),
...                    Polygon(third_square, [])]),
...      Multipolygon([Polygon(second_square, []),
...                    Polygon(fourth_square, [])]))
...  == Multipolygon([Polygon(first_square, []),
...                   Polygon(third_square, [])]))
True
>>> (subtract_multipolygons(
...      Multipolygon([Polygon(first_inner_square, []),
...                    Polygon(third_inner_square, [])]),
...      Multipolygon([Polygon(second_square, []),
...                    Polygon(fourth_square, [])]))
... == subtract_multipolygons(
...         Multipolygon([Polygon(first_inner_square, []),
...                       Polygon(third_inner_square, [])]),
...         Multipolygon([Polygon(first_square,
...                               [clockwise_first_inner_square]),
...                       Polygon(third_square,
...                               [clockwise_third_inner_square])]))
... == Multipolygon([Polygon(first_inner_square, []),
...                  Polygon(third_inner_square, [])]))
True
>>> (subtract_multipolygons(
...      Multipolygon([Polygon(first_square, []),
...                    Polygon(third_square, [])]),
...      Multipolygon([Polygon(first_inner_square, []),
...                    Polygon(third_inner_square, [])]))
...  == Multipolygon([Polygon(first_square,
...                           [clockwise_first_inner_square]),
...                   Polygon(third_square,
...                           [clockwise_third_inner_square])]))
True
```

`clipping.planar.`**`symmetric_subtract_multipolygons`**(*first: ground.hints.Multipolygon*, *second: ground.hints.Multipolygon*, ***, *context: Optional[ground.base.Context] = None*) → Union[ground.hints.Empty, ground.hints.Multipolygon, ground.hints.Polygon]

Returns symmetric difference of multipolygons.

Time complexity: `O(segments_count * log segments_count)`

Memory complexity: `O(segments_count)`

where `segments_count = edges_count + intersections_count`, `edges_count = first_edges_count + second_edges_count`, `first_edges_count = sum(len(polygon.`

border.vertices) + sum(len(hole.vertices) for hole in polygon.holes) for polygon in first.polygons), second_edges_count = sum(len(polygon.border.vertices) + sum(len(hole.vertices) for hole in polygon.holes) for polygon in second.polygons), intersections_count — number of intersections between multipolygons edges.

Parameters

- **first** – first operand.
- **second** – second operand.
- **context** – geometric context.

Returns symmetric difference of operands.

```
>>> from ground.base import get_context
>>> context = get_context()
>>> EMPTY = context.empty
>>> Contour = context.contour_cls
>>> Multipolygon = context.multipolygon_cls
>>> Point = context.point_cls
>>> Polygon = context.polygon_cls
>>> first_square = Contour([Point(0, 0), Point(4, 0), Point(4, 4),
...                         Point(0, 4)])
>>> second_square = Contour([Point(4, 0), Point(8, 0), Point(8, 4),
...                          Point(4, 4)])
>>> third_square = Contour([Point(4, 4), Point(8, 4), Point(8, 8),
...                         Point(4, 8)])
>>> fourth_square = Contour([Point(0, 4), Point(4, 4), Point(4, 8),
...                          Point(0, 8)])
>>> first_inner_square = Contour([Point(1, 1), Point(3, 1), Point(3, 3),
...                               Point(1, 3)])
>>> second_inner_square = Contour([Point(5, 1), Point(7, 1), Point(7, 3),
...                                Point(5, 3)])
>>> third_inner_square = Contour([Point(5, 5), Point(7, 5), Point(7, 7),
...                               Point(5, 7)])
>>> clockwise_first_inner_square = Contour([Point(1, 1), Point(1, 3),
...                                         Point(3, 3), Point(3, 1)])
>>> clockwise_second_inner_square = Contour([Point(5, 1), Point(5, 3),
...                                          Point(7, 3), Point(7, 1)])
>>> clockwise_third_inner_square = Contour([Point(5, 5), Point(5, 7),
...                                         Point(7, 7), Point(7, 5)])
>>> (symmetric_subtract_multipolygons(
...      Multipolygon([Polygon(first_square, []),
...                    Polygon(third_square, [])]),
...      Multipolygon([Polygon(first_square, []),
...                    Polygon(third_square, [])]))
...  is EMPTY)
True
>>> (symmetric_subtract_multipolygons(
...          Multipolygon([Polygon(first_inner_square, []),
...                        Polygon(third_inner_square, [])]),
...          Multipolygon([Polygon(first_inner_square, []),
...                        Polygon(second_inner_square, []),
...                        Polygon(third_inner_square, [])]))
...  == symmetric_subtract_multipolygons(
```

(continues on next page)

(continued from previous page)

```
...         Multipolygon([Polygon(first_inner_square, []),
...                       Polygon(second_inner_square, []),
...                       Polygon(third_inner_square, [])]),
...         Multipolygon([Polygon(first_inner_square, []),
...                       Polygon(third_inner_square, [])]))
...  == Polygon(second_inner_square, []))
True
>>> (symmetric_subtract_multipolygons(
...         Multipolygon([Polygon(first_square, []),
...                       Polygon(third_square, [])]),
...         Multipolygon([Polygon(second_square, []),
...                       Polygon(fourth_square, [])]))
...  == Polygon(Contour([Point(0, 0), Point(8, 0), Point(8, 8),
...                     Point(0, 8)]), []))
True
>>> (symmetric_subtract_multipolygons(
...     Multipolygon([Polygon(first_square, []),
...                   Polygon(third_inner_square, [])]),
...     Multipolygon([Polygon(first_inner_square, []),
...                   Polygon(third_inner_square, [])]))
...  == symmetric_subtract_multipolygons(
...         Multipolygon([Polygon(first_inner_square, []),
...                       Polygon(third_inner_square, [])]),
...         Multipolygon([Polygon(first_square, []),
...                       Polygon(third_inner_square, [])]))
...  == Polygon(first_square, [clockwise_first_inner_square]))
True
>>> (symmetric_subtract_multipolygons(
...     Multipolygon([Polygon(first_square, []),
...                   Polygon(second_inner_square, [])]),
...     Multipolygon([Polygon(first_inner_square, []),
...                   Polygon(second_square, [])]))
...  == Polygon(Contour([Point(0, 0), Point(8, 0), Point(8, 4),
...                     Point(0, 4)]),
...           [clockwise_first_inner_square,
...            clockwise_second_inner_square]))
True
>>> (symmetric_subtract_multipolygons(
...         Multipolygon([Polygon(first_inner_square, []),
...                       Polygon(third_inner_square, [])]),
...         Multipolygon([Polygon(first_square,
...                               [clockwise_first_inner_square]),
...                       Polygon(third_square,
...                               [clockwise_third_inner_square])]))
... == Multipolygon([Polygon(first_square, []),
...                 Polygon(third_square, [])]))
True
>>> (symmetric_subtract_multipolygons(
...     Multipolygon([Polygon(first_inner_square, []),
...                   Polygon(third_inner_square, [])]),
...     Multipolygon([Polygon(second_square, []),
...                   Polygon(fourth_square, [])]))
```

(continues on next page)

(continued from previous page)

```
...   == Multipolygon([Polygon(fourth_square, []),
...                    Polygon(first_inner_square, []),
...                    Polygon(second_square, []),
...                    Polygon(third_inner_square, [])]))
True
>>> (symmetric_subtract_multipolygons(
...      Multipolygon([Polygon(first_inner_square, []),
...                    Polygon(third_inner_square, [])]),
...      Multipolygon([Polygon(first_square, []),
...                    Polygon(third_square, [])]))
...   == symmetric_subtract_multipolygons(
...          Multipolygon([Polygon(first_square, []),
...                        Polygon(third_inner_square, [])]),
...          Multipolygon([Polygon(first_inner_square, []),
...                        Polygon(third_square, [])]))
...   == symmetric_subtract_multipolygons(
...          Multipolygon([Polygon(first_square, []),
...                        Polygon(third_square, [])]),
...          Multipolygon([Polygon(first_inner_square, []),
...                        Polygon(third_inner_square, [])]))
...   == Multipolygon([Polygon(first_square,
...                            [clockwise_first_inner_square]),
...                    Polygon(third_square,
...                            [clockwise_third_inner_square])]))
True
```

`clipping.planar.`**`unite_multipolygons`**(*first: ground.hints.Multipolygon*, *second: ground.hints.Multipolygon*, ***, *context: Optional[ground.base.Context] = None*) → Union[ground.hints.Multipolygon, ground.hints.Polygon]

Returns union of multipolygons.

Time complexity: `O(segments_count * log segments_count)`

Memory complexity: `O(segments_count)`

where `segments_count = edges_count + intersections_count`, `edges_count = first_edges_count + second_edges_count`, `first_edges_count = sum(len(polygon.border.vertices) + sum(len(hole.vertices) for hole in polygon.holes) for polygon in first.polygons)`, `second_edges_count = sum(len(polygon.border.vertices) + sum(len(hole.vertices) for hole in polygon.holes) for polygon in second.polygons)`, `intersections_count` — number of intersections between multipolygons edges.

Parameters

- **first** – first operand.
- **second** – second operand.
- **context** – geometric context.

Returns union of operands.

```
>>> from ground.base import get_context
>>> context = get_context()
>>> EMPTY = context.empty
>>> Contour = context.contour_cls
```

(continues on next page)

(continued from previous page)

```
>>> Multipolygon = context.multipolygon_cls
>>> Point = context.point_cls
>>> Polygon = context.polygon_cls
>>> first_square = Contour([Point(0, 0), Point(4, 0), Point(4, 4),
...                         Point(0, 4)])
>>> second_square = Contour([Point(4, 0), Point(8, 0), Point(8, 4),
...                          Point(4, 4)])
>>> third_square = Contour([Point(4, 4), Point(8, 4), Point(8, 8),
...                         Point(4, 8)])
>>> fourth_square = Contour([Point(0, 4), Point(4, 4), Point(4, 8),
...                          Point(0, 8)])
>>> first_inner_square = Contour([Point(1, 1), Point(3, 1), Point(3, 3),
...                               Point(1, 3)])
>>> second_inner_square = Contour([Point(5, 1), Point(7, 1), Point(7, 3),
...                                Point(5, 3)])
>>> third_inner_square = Contour([Point(5, 5), Point(7, 5), Point(7, 7),
...                               Point(5, 7)])
>>> clockwise_first_inner_square = Contour([Point(1, 1), Point(1, 3),
...                                         Point(3, 3), Point(3, 1)])
>>> clockwise_second_inner_square = Contour([Point(5, 1), Point(7, 1),
...                                          Point(7, 3), Point(5, 3)])
>>> clockwise_third_inner_square = Contour([Point(5, 5), Point(5, 7),
...                                         Point(7, 7), Point(7, 5)])
>>> (unite_multipolygons(Multipolygon([Polygon(first_square, []),
...                                    Polygon(second_inner_square, [])]),
...                      Multipolygon([Polygon(first_inner_square, []),
...                                    Polygon(second_square, [])]))
...  == Polygon(Contour([Point(0, 0), Point(8, 0), Point(8, 4),
...                      Point(0, 4)]), []))
True
>>> (unite_multipolygons(Multipolygon([Polygon(first_square, []),
...                                    Polygon(third_square, [])]),
...                      Multipolygon([Polygon(second_square, []),
...                                    Polygon(fourth_square, [])]))
...  == Polygon(Contour([Point(0, 0), Point(8, 0), Point(8, 8),
...                      Point(0, 8)]), []))
True
>>> (unite_multipolygons(Multipolygon([Polygon(first_square, []),
...                                    Polygon(third_inner_square, [])]),
...                      Multipolygon([Polygon(first_inner_square, []),
...                                    Polygon(third_inner_square, [])]))
...  == unite_multipolygons(
...         Multipolygon([Polygon(first_inner_square, []),
...                       Polygon(third_inner_square, [])]),
...         Multipolygon([Polygon(first_square, []),
...                       Polygon(third_inner_square, [])]))
...  == Multipolygon([Polygon(first_square, []),
...                   Polygon(third_inner_square, [])]))
True
>>> (unite_multipolygons(Multipolygon([Polygon(first_square, []),
...                                    Polygon(third_square, [])]),
...                      Multipolygon([Polygon(first_square, []),
```

(continues on next page)

(continued from previous page)

```
...                                       Polygon(third_square, [])]))
...     == unite_multipolygons(
...             Multipolygon([Polygon(first_inner_square, []),
...                           Polygon(third_inner_square, [])]),
...             Multipolygon([Polygon(first_square,
...                                   [clockwise_first_inner_square]),
...                           Polygon(third_square,
...                                   [clockwise_third_inner_square])]))
...    == unite_multipolygons(Multipolygon([Polygon(first_inner_square, []),
...                                         Polygon(third_inner_square, [])]),
...                           Multipolygon([Polygon(first_square, []),
...                                         Polygon(third_square, [])]))
...    == unite_multipolygons(Multipolygon([Polygon(first_square, []),
...                                         Polygon(third_inner_square, [])]),
...                           Multipolygon([Polygon(first_inner_square, []),
...                                         Polygon(third_square, [])]))
...     == unite_multipolygons(
...             Multipolygon([Polygon(first_square, []),
...                           Polygon(third_square, [])]),
...             Multipolygon([Polygon(first_inner_square, []),
...                           Polygon(third_inner_square, [])]))
...    == Multipolygon([Polygon(first_square, []),
...                     Polygon(third_square, [])]))
True
>>> (unite_multipolygons(Multipolygon([Polygon(first_inner_square, []),
...                                       Polygon(third_inner_square, [])]),
...                         Multipolygon([Polygon(first_inner_square, []),
...                                       Polygon(second_inner_square, []),
...                                       Polygon(third_inner_square, [])]))
...     == unite_multipolygons(
...             Multipolygon([Polygon(first_inner_square, []),
...                           Polygon(second_inner_square, []),
...                           Polygon(third_inner_square, [])]),
...             Multipolygon([Polygon(first_inner_square, []),
...                           Polygon(third_inner_square, [])]))
...     == Multipolygon([Polygon(first_inner_square, []),
...                      Polygon(second_inner_square, []),
...                      Polygon(third_inner_square, [])]))
True
>>> (unite_multipolygons(
...         Multipolygon([Polygon(first_inner_square, []),
...                       Polygon(third_inner_square, [])]),
...         Multipolygon([Polygon(second_square, []),
...                       Polygon(fourth_square, [])]))
...     == Multipolygon([Polygon(fourth_square, []),
...                      Polygon(first_inner_square, []),
...                      Polygon(second_square, []),
...                      Polygon(third_inner_square, [])]))
True
```

PYTHON MODULE INDEX

C

INDEX

U