**Algorithm for checking if, and where two walls overlap**

Implemented in Wall::mergeWalls().

There are four possible ways that two walls can overlap, as shown below.



Here we name P1 the first point of the first wall, P2 the second point of the first wall, P3 the first point of the second wall, and P4 the second point of the second wall.

Now we sort P1 upto P4 in order of their non-equal coordinate, and call them O1, O2, O3, O4. This divides the intersection into three segments, namely [O1, O2], [O2, O3], and [O3, O4]. Notice in the picture how only the segment [O2, O3] can be present on both walls.

We now check if the second segment is on both walls by checking if both O2 and O3 are on both walls. This is implemented in the Wall::pointOnWall() method, which simply checks if the coordinate of the point is between the end coordinates of the wall. If the second segment is indeed on both walls, we know that we need to merge the segment between Pi and Pj, where i is the original index of O2, and j is the original index of O3. (This means that Pi, after sorting, ended up to be O2.)

**Algorithm for checking how to update a wall**

Implemented in Domain::mergeDomains().

There are four possible ways that a wall can be updated according to overlap, as shown below.



Here we call the four possible ways Type 1, Type 2, Type 3, and Type 4, from left to right on the picture. Note that in a wall, y0 < y1 is not guaranteed. This means that we need subtypes. Type 1a will be used for Type 1 where y0 < y1. Type 1b will be used for Type 1 where y0 > y1, and so on.