

```

/*
 * symmetry_group.c
 *
 * This file provides the following two functions
 *
 *      FuncResult  compute_symmetry_group(
 *                      Triangulation      *manifold,
 *                      SymmetryGroup      **symmetry_group_of_manifold,
 *                      SymmetryGroup      **symmetry_group_of_link,
 *                      Triangulation      **symmetric_triangulation,
 *                      Boolean            *is_full_group);
 *
 *      void free_symmetry_group(SymmetryGroup *symmetry_group);
 *
 * If *manifold is not a manifold (i.e. if any cusp is filled with
 * coefficients other than relatively prime integers), then
 * compute_symmetry_group() returns func_bad_input.
 *
 * If *manifold is a closed manifold (i.e. all cusps are
 * filled with relatively prime Dehn filling coefficients), then
 * compute_symmetry_group() attempts to compute its symmetry group.
 * If it succeeds, it sets
 *
 *      *symmetry_group_of_manifold    points to the symmetry group
 *      *symmetry_group_of_link        left as NULL
 *      *symmetric_triangulation        points to a Dehn filling
 *                                      description of the manifold
 *                                      realizing the symmetry group
 *
 *      *is_full_group                 TRUE
 *      returns                         func_OK.
 *
 * Sometimes it will compute a group which is known to be a
 * subgroup of the true symmetry group, but may or may not be
 * the full group. In this case it sets
 *
 *      *symmetry_group_of_manifold    points to the known subgroup
 *      *symmetry_group_of_link        left as NULL
 *      *symmetric_triangulation        points to a Dehn filling
 *                                      description of the manifold
 *                                      realizing the subgroup
 *
 *      *is_full_group                 FALSE
 *      returns                         func_OK.
 *
 * If the algorithm fails entirely (this should be rare) it sets
 *
 *      *symmetry_group_of_manifold    left as NULL
 *      *symmetry_group_of_link        left as NULL
 *      *symmetric_triangulation        left as NULL
 *      *is_full_group                 undefined
 *      returns                         func_failed.
 *
 * For details on the algorithm for closed manifolds,
 * see symmetry_group_closed.c.
 *
 * If *manifold is a cusped manifold (i.e. at least one cusp is unfilled,
 * and all filled cusps have relatively prime Dehn filling
 * coefficients), then compute_symmetry_group() attempts to compute
 * its symmetry group, and also the symmetry group of the
 * corresponding link (defined in symmetry_group_cusped.c).
 * If it succeeds, it sets
 *
 *      *symmetry_group_of_manifold    points to the symmetry group
 *      *symmetry_group_of_link        points to link symmetry group
 *      *symmetric_triangulation        left as NULL
 *      *is_full_group                 TRUE
 *      returns                         func_OK.
 *
 * If it fails (this should be rare), it sets
 *
 *      *symmetry_group_of_manifold    left as NULL
 *      *symmetry_group_of_link        left as NULL
 *      *symmetric_triangulation        left as NULL
 *      *is_full_group                 undefined
 *      returns                         func_failed.

```

```

*
*   The case where only a subgroup is known does not occur for
*   cusped manifolds. For details on the algorithm for cusped
*   manifolds, see symmetry_group_cusped.c.
*
*
*   compute_symmetry_group() assumes that initially
*
*       *symmetry_group_of_manifold == NULL
*       *symmetry_group_of_link     == NULL
*       *symmetric_triangulation    == NULL.
*
*   In other words, there are no leftover SymmetryGroups or
*   Triangulations to be freed.
*
*   free_symmetry_group() frees the SymmetryGroup.
*/

#include "kernel.h"

FuncResult compute_symmetry_group(
    Triangulation *manifold,
    SymmetryGroup **symmetry_group_of_manifold,
    SymmetryGroup **symmetry_group_of_link,
    Triangulation **symmetric_triangulation,
    Boolean *is_full_group)
{
    Triangulation *simplified_manifold;
    FuncResult result;

    /*
     * Make sure the variables used to pass back our results
     * are all initially empty.
     */
    if (*symmetry_group_of_manifold != NULL
        || *symmetry_group_of_link != NULL
        || *symmetric_triangulation != NULL)
        uFatalError("compute_symmetry_group", "symmetry_group");

    /*
     * If the space isn't a manifold, return func_bad_input.
     */
    if (all_Deahn_coefficients_are_relatively_prime_integers(manifold) == FALSE)
        return func_bad_input;

    /*
     * Whether the manifold is cusped or not, we want to begin
     * by getting rid of "unnecessary" cusps.
     */
    simplified_manifold = fill_reasonable_cusps(manifold);
    if (simplified_manifold == NULL)
        return func_failed;

    /*
     * Split into cases according to whether the manifold is
     * closed or cusped (i.e. whether all cusps are filled or not).
     */
    if (all_cusps_are_filled(simplified_manifold) == TRUE)
        result = compute_closed_symmetry_group(
            simplified_manifold,
            symmetry_group_of_manifold,
            symmetric_triangulation,
            is_full_group);
    else
    {
        result = compute_cusped_symmetry_group(
            simplified_manifold,
            symmetry_group_of_manifold,
            symmetry_group_of_link);

        *is_full_group = TRUE;
    }

    free_triangulation(simplified_manifold);

```

```
    return result;
}

void free_symmetry_group(
    SymmetryGroup *symmetry_group)
{
    if (symmetry_group != NULL)
    {
        int i;

        free_isometry_list(symmetry_group->symmetry_list);

        for (i = 0; i < symmetry_group->order; i++)
            my_free(symmetry_group->product[i]);
        my_free(symmetry_group->product);

        my_free(symmetry_group->order_of_element);

        my_free(symmetry_group->inverse);

        if (symmetry_group->abelian_description != NULL)
            free_abelian_group(symmetry_group->abelian_description);

        if (symmetry_group->is_direct_product == TRUE)
            for (i = 0; i < 2; i++)
                free_symmetry_group(symmetry_group->factor[i]);

        my_free(symmetry_group);
    }
}
```