

```

/*
 * covers.h
 *
 * SnapPea constructs an n-sheeted cover of a given manifold as follows.
 * First it creates n copies of a fundamental domain. For convenience it
 * uses the fundamental domain defined in choose_generators.c, which is
 * a union of the triangulation's tetrahedra. It assigns to each generator
 * (defined in choose_generators.c) a permutation of the n sheets, and
 * glues the sheets accordingly. The product of the permutations
 * surrounding an edge class must, of course, be the identity.
 * Algebraically this is equivalent to finding transitive representations
 * of the fundamental group into  $S(n)$ , the symmetric group on n letters.
 * ("Transitive" means that the corresponding covering space is connected.)
 *
 * The algorithm for computing all connected n-sheeted covers of a
 * given manifold consists of two parts:
 *
 * (1) Find all transitive representations of the manifold's fundamental
 *     group into  $S(n)$ . [cf. representations.c]
 *
 * (2) For each representation, construct the corresponding cover.
 *     [cover.c]
 *
 * This naive algorithm can, of course, be simplified. For example,
 * representations which are conjugate by an element of  $S(n)$  yield
 * equivalent covering spaces. The file representations.c describes
 * these optimizations.
 */

/*
 * This file (covers.h) is intended solely for inclusion in SnapPea.h.
 */

/*
 * A covering is "regular" iff for any two lifts of a point in the base
 * manifold, there is a covering transformation taking one to the other.
 * (An alternative definition is that the cover's fundamental group
 * projects down to a normal subgroup of the base manifold's fundamental
 * group. For a proof that the two definitions are equivalent, please
 * see page 362 of Rolfsen's Knots and Links.)
 *
 * All cyclic coverings are regular.
 */
typedef int CoveringType;
enum
{
    unknown_cover,
    irregular_cover,
    regular_cover,
    cyclic_cover
};

typedef struct RepresentationIntoSn RepresentationIntoSn;

typedef struct
{
    /*
     * How many face pairs does the fundamental domain
     * (defined in choose_generators.c) have?
     */
    int num_generators;

    /*
     * How many sheets does the covering have?
     */
    int num_sheets;

    /*
     * How many cusps (filled or unfilled) does the manifold have?
     * (For use with primitive_Dehn_image below.)
     */
    int num_cusps;

```

```

/*
 * The representations themselves are kept on a NULL-terminated
 * singly linked list.
 */
RepresentationIntoSn    *list;

} RepresentationList;

struct RepresentationIntoSn
{
/*
 * The permutation corresponding to generator i takes sheet j
 * of the cover to sheet image[i][j].
 *
 * Note that the size of the image array depends on both the
 * num_generators and the num_sheets defined in the RepresentationList.
 */
int                    **image;

/*
 * The algorithm in construct_cover() in cover.c would like to know
 * the permutation assigned to each "primitive" Dehn filling curve.
 * If the Dehn filling coefficients are (a,b), the primitive Dehn
 * filling curve is defined to be (a/c, b/c), where c = gcd(a,b).
 * (When the Dehn filling coefficients are relative prime -- as is
 * always the case for a manifold -- the primitive Dehn filling curve
 * is just the Dehn filling curve itself, and the assigned permutation
 * is perforce the identity. The concept of a primitive Dehn filling
 * curve is useful only for orbifolds.)
 *
 * The permutation corresponding to the primitive Dehn filling curve
 * on cusp i takes sheet j of the cover to sheet Dehn_image[i][j].
 * (For unfilled cusps, the identity permutation is given instead.)
 */
int                    **primitive_Dehn_image;

/*
 * Is the cover defined by this representation irregular,
 * regular or cyclic?
 */
CoveringType          covering_type;

/*
 * The RepresentationList keeps RepresentationIntoSn's on
 * a NULL-terminated singly linked list.
 */
RepresentationIntoSn    *next;
};

/*
 * find_representations() takes a PermutationSubgroup parameter
 * specifying the subgroup of the symmetric group S(n) into which
 * the representations are to be found.
 */
typedef int PermutationSubgroup;
enum
{
    permutation_subgroup_Zn,    /* finds cyclic covers only */
    permutation_subgroup_Sn    /* finds all n-fold covers */
    /* eventually an option for dihedral covers could be added */
};

```