Task 1: Convex polytopes

Basic setup of convex polytopes

In this task you will implement a new state set representation for reachability analysis of linear hybrid systems: Convex polytopes. This task is dedicated to provide a basic implementation of bounded, non-degenerate convex polyhedra.

Due date: Friday, 18^{th} December 2015.

Present your implementation in a short presentation ($\sim$ 20 minutes) along with your ideas towards the next task(s).

Tasks

1. Create the basic datastructure to be able to represent a d-dimensional convex polytope. Make sure to provide convenient constructors and all required getters, setters and print methods, as you did for the first task. Despite from the empty constructor, you should be able to construct a convex polytope at least from

   - a vector of points (`std::vector<hypro::Point<Number> >`),
   - a vector of hyperplanes (`std::vector<hypro::Hyperplane<Number> >`),
   - a matrix A and a vector b such that $Ax \leq b$ holds. Use the types `hypro::matrix_t<Number>` and `hypro::vector_t<Number>` for your parameters.
   - another convex Polytope (`hypro::Polytope<Number>`),
   - by a static function `Empty(std::size_t dimension)`, which constructs an empty convex polytope of the required dimension.

   Extend the provided test file `PTermPolytopeTest.cpp` to verify your implementation.

2. Extend your implementation by adding implementations for all operations required for reachability analysis of hybrid systems, which includes:

   - $conv(\cdot \bigcup \cdot)$ - union. As convex polytopes are not closed under union, it is required to compute the convex hull ($conv(..)$) of the union of two boxes.
   - $\cdot \bigcap \cdot$ - intersection. To verify a flowpipe segment against the invariant of the current location, it is required to compute the intersection of the invariant and the segment and test it for emptiness.
   - $empty(\cdot)$ - emptiness. As stated before, this is needed to verify the segment lies inside the invariant of the current location.

- $A(\cdot)$ - linear transformation. To create a flowpipe we can encode the linear dynamics inside a location of a linear hybrid automaton into a linear transformation. Consecutive application of this linear transformation allows us to compute a flowpipe.
- *Minkowski*$(\cdot, \cdot)$ - Minkowski sum. The Minkowski sum is the set-equivalent to a sum and is needed for the over-approximation of the initial set to cover the dynamics inside the current location.

Make sure to verify your implementation against the tests in the provided test file and extend the tests where needed.

3. Provide additional methods:
   - `std::size_t dimension() const` – returns the dimension of the current object.
   - `Number supremum() const` – returns $\sup_{u \in \mathcal{U}} ||u||_\infty$, the supremum regarding the infinity norm of the current polytope.
   - `void print() const` – a print method to `std::cout`.
   - `friend ostream& operator« (ostream& out, const PTermPolytope<Number>& _in)` – an output operator which can be used with any output-stream (e.g. `std::cout`).

Hints and additional information

- We have prepared an empty folder `src/lib/representations/PTermPolytope` where you should put your implementation.
- We will try to provide an exemplary reachability analysis algorithm which requires the implementation of the previously mentioned methods (we will announce when the algorithm is available).