Implementation of “Parma Polyhedron Library”-functions in MATLAB

Leonhard Asselborn

Electrical and Computer Engineering
Carnegie Mellon University

Group meeting – Oct. 21\textsuperscript{st} 2010
Overview

- Introduction
- Motivation
- MEX-Functions
- Parma Polyhedron Library
- Challenges in detail
- Benchmarking results
- Conclusion
Introduction

Reachability analysis of nonlinear and hybrid Systems

Reachable set can be approximated with polytopes.
Introduction

Polytope:
- geometric object with flat sides
- n-dimensions
- 2- polytope: polygon
- 3- polytope: polyhedron

- Regular polytopes
- Convex polytopes
- ...

Diagram showing a 3-dimensional polytope (P) with vertices and flat sides.
Introduction

Convex Polytope

- Some authors use convex polytope and convex polyhedron interchangeably

- Two representations
  - Vertex representation (V-representation)
  - Half space representation (H-representation)

\[ P = \{ x \in \mathbb{R}^n \mid Cx \leq d \}, \quad C \in \mathbb{R}^{q \times n}, \quad d \in \mathbb{R}^q \]
Motivation

Matlab:
- Easy, interactive environment
- Fast numerical algorithms
- Easy building of prototypes
- Large number of toolboxes

Parma Polyhedron Library:
- C++ Library developed at the University of Parma
- Used to create convex polytopes
- Very user friendly (you write \( x + 2*y + 5*z \leq 7 \) when you mean it)
- Efficient
- Very precise by using GMP data types (precision is limited by the available memory)

My internship
MEX-functions

- MEX stands for Matlab EXecutable
- External interface function which allow to interface with C/C++ or Fortran subroutines.
- Two main reasons:
  - Ability to call large existing C++ routines
  - Speed
- Runs like a function in Matlab: \([\text{result}] = \text{mexfunc}(A,B,C)\)
MEX-functions

- Code example:

```c
void mexFunction( int nlhs, mxArray *plhs[],
                  int nrhs, const mxArray *prhs[]){
    if ((nrhs == 2)&&(nlhs == 1)){
        double *Input0 = mxGetPr(prhs[1]);
        double *Input1 = mxGetPr(prhs[2]);

        double *Return_arg = mxGetPr(plhs[0]);
        Return_arg[0] = Input0[0] + Input1[0];
    }
}
```
Parma Polyhedron Library

- Code example:

```c
int main()
{
    Variable x(0);
    Variable y(1);
    Constraint_System CS;
    Linear_Expression LE1 = -1*x + 0*y;
    Linear_Expression LE2 = 1*x + 0*y;
    CS.insert(LE1 <= 3);
    CS.insert(LE2 <= 5);
    C_Polyhedron P(CS);
}
```
Challenges in detail

- Make PPL and GMP available in Eclipse
- Compile to .mexglx files with Eclipse
- Find a structure to be easily maintained
Challenges in detail

- Code example:

```cpp
list<Variable> VarList;
list<Variable>::iterator it;

// creates a list of cols_C "Variable" objects
for(int i=0; i< cols_C; i++)
    VarList.push_back(Variable(i));

// creates linear expression "lin_exp_temp"
for ( it=VarList.begin() ; it != VarList.end(); it++ ){
    m = it->id();
    lin_exp_temp+=C_row_temp[m]*(*it);
}
```
Benchmarking results

- Randomized zonotopes are investigated
- Zonotope is a special form of polytopes
- PPL-Library compared with MPT-Toolbox
- Procedure:
  1. $P = P_1 \cap P_2$
  2. `extremePoints(P)`
- Avg. time of five iterations is computed
# Benchmarking results

<table>
<thead>
<tr>
<th>halfspaces</th>
<th>avg. time MPT [s]</th>
<th>avg. time MPT [s]</th>
<th>avg. time PPL [s]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Solver: CCD</td>
<td>Solver: analytical</td>
<td></td>
</tr>
<tr>
<td>dimension</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>0.0047</td>
<td>0.0039</td>
<td>0.0012</td>
</tr>
<tr>
<td>n = 2</td>
<td>8</td>
<td>0.0067</td>
<td>0.0049</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>0.0073</td>
<td>0.0056</td>
</tr>
<tr>
<td>dimension</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>0.0149</td>
<td>0.1094</td>
<td>0.0155</td>
</tr>
<tr>
<td>n = 4</td>
<td>40</td>
<td>0.0287</td>
<td>0.2292</td>
</tr>
<tr>
<td></td>
<td>70</td>
<td>0.0536</td>
<td>0.8288</td>
</tr>
<tr>
<td>dimension</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>42</td>
<td>0.105</td>
<td>53.2394</td>
<td>1.151</td>
</tr>
<tr>
<td>n = 6</td>
<td>112</td>
<td>1.341</td>
<td>251.42</td>
</tr>
<tr>
<td></td>
<td>252</td>
<td>9.922</td>
<td>-</td>
</tr>
</tbody>
</table>
Benchmarking results

- Randomized parallelootope are investigated
- Procedure:
  1. $P = P_1 \cap P_2$
  2. `extremePoints(P)`
- Avg. time of five iterations is computed

<table>
<thead>
<tr>
<th>dimensions</th>
<th>2</th>
<th>4</th>
<th>6</th>
<th>8</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>avg. time MPT [s] Solver: CCD</td>
<td>0.0025</td>
<td>0.0054</td>
<td>0.017</td>
<td>0.08</td>
<td>4.08</td>
</tr>
<tr>
<td>avg. time MPT [s] Solver: analytical</td>
<td>0.0031</td>
<td>0.0331</td>
<td>0.999</td>
<td>91.555</td>
<td>-</td>
</tr>
<tr>
<td>avg. time PPL [s]</td>
<td>0.00074</td>
<td>0.0032</td>
<td>0.029</td>
<td>0.62</td>
<td>39.9</td>
</tr>
</tbody>
</table>
Benchmarking results

- Interesting effect occurred:

vertices using PPL:

1.452139681268078  5.160753400552368  -1.545303098756335  -0.967780946093587
1.452139681268077  5.160753400552365  -1.545303098756339  -0.967780946093589
1.452139681268070  5.160753400552331  -1.545303098756381  -0.967780946093606

vertex using MPT:

1.452139681265697  5.160753400550326  -1.545303098757148  -0.967780946092891
Conclusion

- You need to have PPL & GMP library installed on your system
- A way to use MEX-functions with non-standard C++ libraries
- Structure of the program allows to add new functions easily
- Interface is more precise than MPT-toolbox

Thank you for your attention!
Sources

- http://www.clemson.edu/ces/crb/students/vilas/projects/rp/images/c++Logo.png