Chapter 1

Modeling

One significant difference between coho and other hybrid system tools is lacking of explicit dynamics functions.Ususually, the dyanamic system described by ODE equations is required at the beginning by the hybrid verification tools. However, in coho, the dynamic of circuit is described by a table of simulation data, an explicit equation is not avaliable. We develope algorithm to approximate the dynamic as *linear differential inclusion* (LDI) with in a given region (space).

Now, we have an problem. To compute the forward space, the LDI should be computed first. However, to compute the LDI, the reachable space with time window $[0, t]$ should be provided. Therefore, an exact solution can not be computed here, over approximation techiniques must be applied.

1.1 Basic Method

The first algorithm developed by Ian is quite simple. The algorithm is shown in algorithm 1. As illustrated in figure 1.1, the red face is f to move foward. The green cube is f^s which is moved forward and the blue cube is f^m to compute the LDI model.

The correctness is easy to prove. As we know, *if the derivative is finite, the extreme trajectories are those emanating from faces of $\mathcal{S}$*. Thus, we can move forard the faces only.

Lemma 1.1.1 *Any trajectories from f^s moves at most Δd on each direction within time $[0, \Delta t]$.*

Proof The maximum derivative $maxDot$ over points of all f^m model space is used to compute the time step Δt. The largest moving distance is $\int_0^{\Delta t} \dot{x}\, dt \leq maxDot \cdot \Delta t = \Delta d$.

With lemma 1.1.1, we know the forward face $\delta_{\Delta t}(f^s)$ is contained in the model space f^m. Thus the LDI model is over approximated and valid.

Algorithm 1: The basic COHO algorithm

```
Input: Current reachable space $\mathcal{S}$ and bloat amount $\Delta d$
Output: Forward reachable space $\delta_{\Delta t}(\mathcal{S})$ and time step $\Delta t$
1  for each slice do
2      for each face do
3          $f^s$= lp_bloat(face,$\Delta d$, inward);
4          $f^m$= lp_bloat($f^s$,$\Delta d$);
5          Compute LDI $\dot{x} = Ax + b \pm u$ within the $f^m$;
6          $maxDot = max\{\ \dot{x} \mid x \in f^m\ \}$;
7          $\Delta t = min(\Delta t, \frac{\Delta d}{maxDot})$ ;
8  for each slice do
9      $[x_i, x_j] = dims(slice)$;
10     for each face do
11         $\delta_{\Delta t}(f) = face\_forward(f^s, LDI, \Delta t)$;
12         $proj(j) = lp\_project(\delta_{\Delta t}(f), x_i, x_j)$;
13     slices(i) = union(projs);
14 $\delta_{\Delta t}(\mathcal{S}) = ph\_create(slices)$;
```

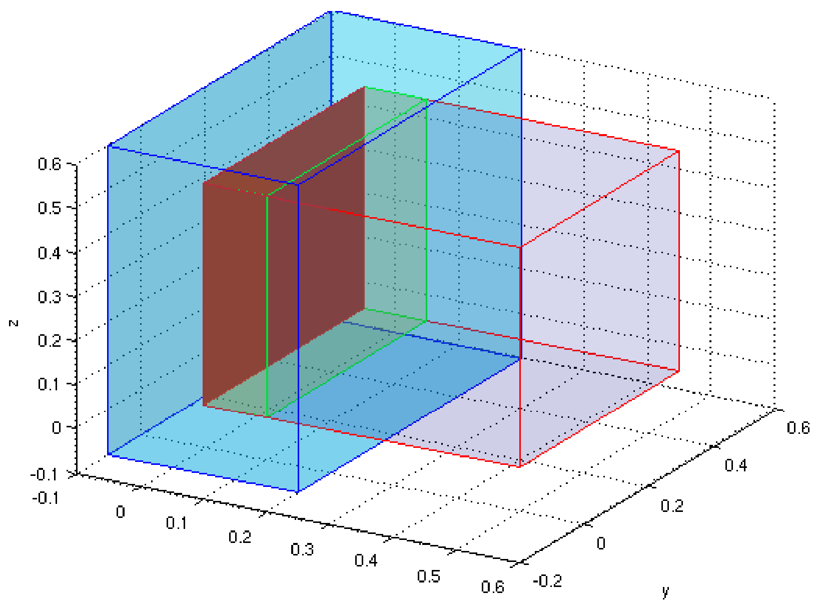


Figure 1.1: An example of the basic algorithm

However, the forward face is projected onto its corresponding slice $[x_i, x_j]$ rather than all slices in the system. The extreme value of x_i (or x_j) may eminates from faces which is on the prism of the slice, as shown in figure 1.2.

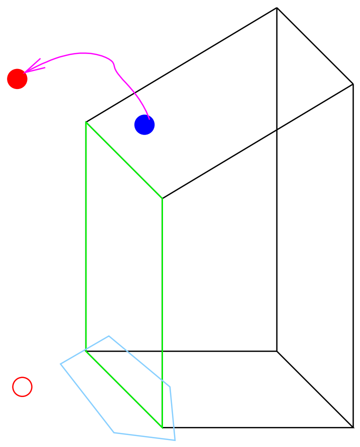


Figure 1.2: Extreme value from faces not on the prism

Thus, moving forward the face only is not sound. Ian's algorithm bloat the face inward by Δd. To prove the correctness, f^s is redefined as moving inward by $2\Delta t$ as shown in figure 1.3. Any point can move inward or outward on x_i direction by at most Δd, the thick of f^s is $2\Delta d$ thus contains all possible points that leads to extreme value of x_i.

Bloating face inward by Δd as shown in algorithm 1 may not be sound (or I do not know how to prove it). Consider the case as shown in figure 1.4, f^b with thick of Δd moves inward by Δd after time Δt, while a point pt behind f^b may move outward by Δd. Then the projection of $\delta_{\Delta t}(f^s)$ does not contains $\delta_{\Delta t}(pt)$.

However, we do not use $2\Delta d$ in the implementation. Ususally the result is still over approximated using Δd although we do not know how to prove it now. Especially there is many other over approximation, such as

- linearization error during the computation of LDI
- integration error during the computation of forward space
- projection error during projectoin forward space onto 2d subspace
- polygon operation error such as simplify error, polygon union(intersection) error

In the implementation, the over approximation must be controlled for true negative false, ususally under approximation rarely be a practical problem. Thus, it is common to use some techinques even we do not know how to prove its correctness.

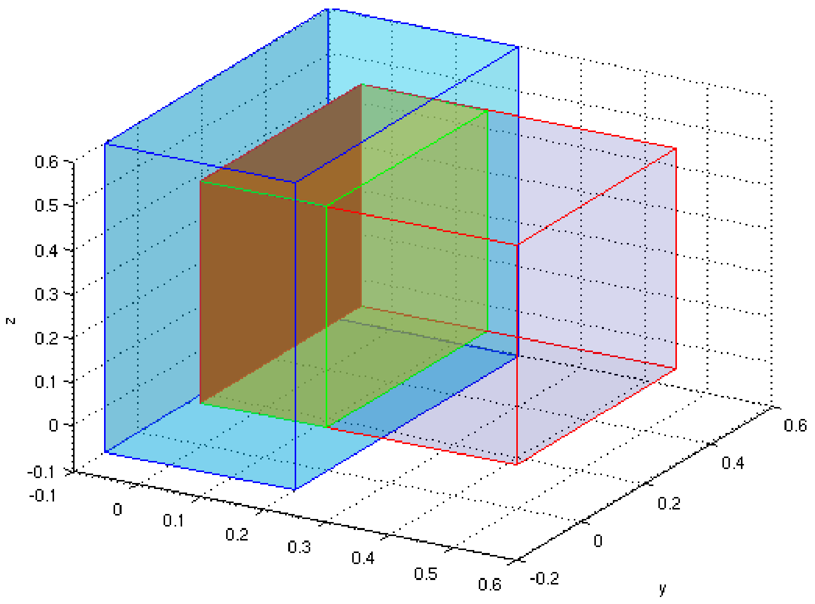


Figure 1.3: Face to forward to prove soundness

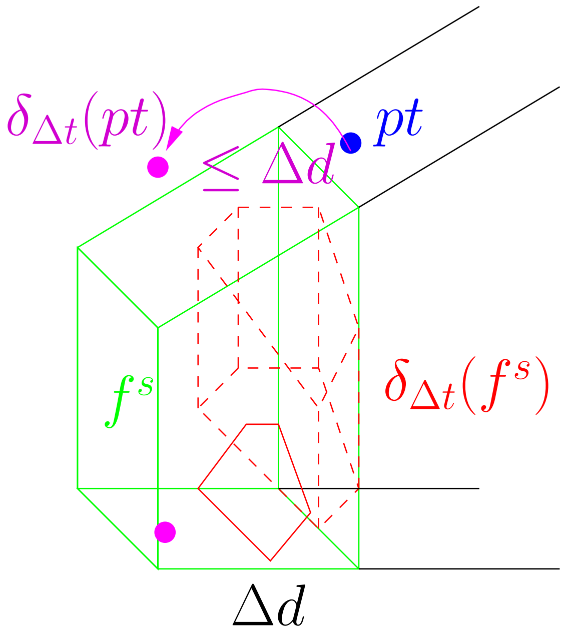


Figure 1.4: Extreme value from point behind f^s

1.2 Guess-Verify

The approximation error of the basic algorithm is usually much larger then necessary. Firt, the extreme derivative is used to compute the time step, which in fact uses the LDI as $\dot{x} = 0 \pm max(\{\dot{x}|x \in f^b\})$.However, the derivative might be much smaller on the trajctories. Even worse, the minimum time step over all faces is applied to all faces because they must use the same time step (or the extreme derivative over all bloated faces is used.), then the time step is too small for most of faces. As a result, the forward face moves much smaller than Δd.

1.2.1 Algorithm

In fact, the pair of space (or bloat amount) `modelLP` to compute LDI and time step t is valid as long as *all trajectories from the face in the time window* $[0, t]$ *is contained within the space use to compute* ***LDI***. Thus, we can **guess** an pair of `modelLP` and t, **compute** the forward face and finally **verify** all trajectories never exceed `modelLP`.

Given a face f and its LDI, the forward face can be computed at any specified time t as $\delta_t f$. However, only finite number of states can be computeds, approximation method must be use to find all reachable space within a time window $[0, t]$. An straightfoward approximation is $\delta_{[0,t]} f \in bloat(convex(f, \delta_t f), \delta d)$ as shown in figure 1.5. Usually δd is insignificant when t is small, thus, it is ignored in COHO now.

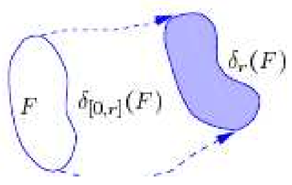


Figure 1.5: The over approximation of reachable space in a time interval

Another problem is how to verify the reachable space is contained in `modelLP`. Because both `modelLP` and $\delta_{[0,t]} f$ are high dimentional convex polyhedrons, the computation is expensive. In COHO, only the projected polygons onto its corresponding slice is checked as $proj(\delta_{[0,t]} f) \in proj(modelLP)$ (In fact, δd is ignored and only $proj(\delta_t f) \in proj(modelLP)$ is checked).

Thus, the verification algorithm is shown in algorithm 2.

1.2.2 Proof of Correctness

The algorithm is sound. First, we will prove that the verify procedure is correct although the trajectories is only check on the slice variables. Then, we will prove

Algorithm 2: The verify algorithm

```
Input: Current reachable space $\mathcal{S}$
Output: Forward reachable space $\delta_t(\mathcal{S})$ and a valid pair of bloat amount
        $\Delta d$ and time step $t$
1  while true do
2      $[t, \Delta d] = guess\_update(t, \Delta d)$;
3      for each slice do
4          $[x_i, x_j] = dims(slice)$;
5          for each face do
6              Compute bloated face $f^b$ by moving all points by $\Delta d$ ;
7              Compute LDI $\dot{x} = Ax + b \pm u$ within the $f^b$;
8              Compute extended face $f^e$ by strech the face by $\Delta d$ on all
               directions except $x_i, x_j$;
9              Forward the extented face $\delta_t(f) = face\_forward(f^e, LDI, \Delta t)$;
10     g = true;
11     for each slice do
12         for each face do
13             proj = lp_project($f^b, x\_i, x\_j$);
14             poly = lp_relax(face,$\Delta d$);
15             if $proj \notin poly$ then
16                 g = false;;
17     if g then
18         break;
19 $\delta_t(\mathcal{S}) = ph\_create(slices)$;
```

that the foward projectagon contains all possible space alougth the algorithm computes on the face only.

Verify Procedure

First, let us prove that *all trajactries are contained within* `modelLP` *($\delta_t(f) \in f^b$) iff $proj(\delta_t(f)) \in poly$.* Assume there is a face f that $\delta_t(f)$ exceeds f^b while all faces passed the verification. Without lost of genearality, let us say the f is on the $[x_1, x_2]$ slice, and $\delta_t(f)$ exceed f^b on x_3 direction. The escaped trajectories has two cases(see figure 1.6 as an example):

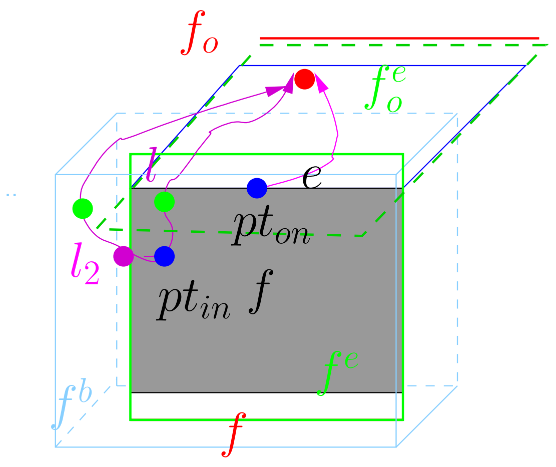


Figure 1.6: An example to illustrate the verify procedure

- The escaped trajctories are from points pt_{on} whose projection is the on the face f_o (projection polygon edge) of some slice with variable x_3 . Then when verify the face f_o, it must be found that pt_{on} moves by more than Δt on x_3 direction because LDI for both face are over approximated thus contains the trajctories.

- The escpated trajctories are from some points pt_{in} whose projection is inside the projection polygons of all slices with variable x_3. Then its coordinate is smaller/greater then the upper/lower bound v defined by some edge e while its trajectory exceeds it. Thus, there must be a intersection line/point l between the trajectory and a half plane h corresponds to e, and l must be in f^b because all trajctories only exceeds on x_3 direction but l is bounded on x_3 direction.

  Each edge is the intersection of $n-1$ half planes, e must have a half plane corresponeds to another face f_o on slice with x_3, let us say x_3, x_4 (otherwise, e is a line parrallel with x_3, v is bounded by the endpoint of e, thus we can use the other edge with the same endpoints of face f).

– If x_4 is neither x_1 nor x_2, the extended face f_o^e bloats by Δd on both x_1, x_2 directions, thus, it contains l. Trajectory from l must moves out of f_o^b within t on direction x_3, thus the verify precedure can find the problen when working on f_o.

– WOLOG, if $x_4 = x_1$, then l can bypass f_o^e by moving out of the range of x_1 of f_o. Then the trajectory must first has an intersection point/line l_2 with another extended face f_{o_2} on $[x3, x_1]$ slice which provides a lower/upper bound of x_1 variable. Then the verify procedure must find trajectory from l_2 exceeds on x_3 direction within t.

Thus, it is impossible that $\delta_t(f) \notin f^b$ if all faces pass our verify prodeure.

Working on prism faces only

Although rather than projecting all forwards face to $x_i, x_j]$ slice, we only foward faces corresponds to polygon edges of the slice and project down to compute the slice of forward projectagon. Similar with the basic algorithm, it should be proved the result is valid, or no trajectories from interior points exceeds exceeds our result.

In fact, in the basic algorithm, rather than bloating the face inward, we can increase the height of face by $2\Delta d + d$ to solve the problem. Because every point can move at most Δd on each direciton. (see old paper for prove).

Using the guess-verify algorithm, it is not gurantteed that every point can move at most Δd on each direction. What we know is that all trajectories are contained by f^d.

The face is bloated inward by Δd and increase the height to the bloated ph. (intersect(bloat(segLP,Δd, inward), bloat(ph.lp,Δd)). The forward face at most moves inward by Δd. If the trajectories from internal points do not exceed $x - \Delta d$, it is no problem. Otherwise, it must enter the region in the top region of the forward face, which is already moved forward. Thus, it can not exceed our result. Figure 1.7 shows the idea.

As shown in the figure 1.7, the blue region is the thick face to move forard, the cyan region is the modelLP to create LDI model. The dark red polygon is the projected polygon of forward face, the left most points must be in the modelLP (the right most might not be in modelLP because we are moving forard a thick face. now we have the problem how to verify it, making the right side inside the modelLP usually is impossible. Project a face and a thick face twice? Forward the face for bloat amount, forward a thick face for other faces.). The internal points must pass the green region to exceed the projected polygon. However, the green region is moved forward thus contains all possibilities.

In fact, if the projected polygon in on the left side, then we only need to bloat the face rather than the thick one. Becaue the face already catch all trajectories that pass over the face. In fact, the tick of face to move forward should be from x to $proj(fwd(x))$ to prove its correctness.

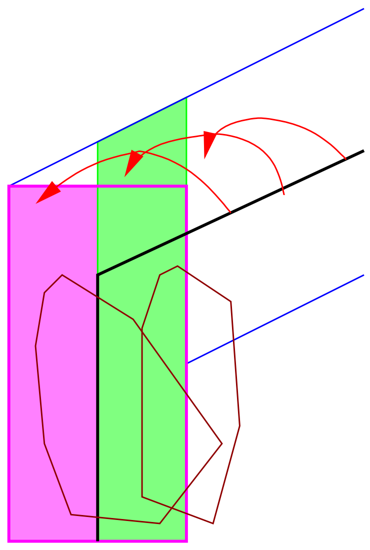


Figure 1.7: Project prism faces is sound

1.3 Felxiable Bloat

Till now, we use the same bloat amount Δd for each direction. However, it is common that some signals change much faster than others in a system. Thus, we can use different bloat amount for different variables. Futhure, a signal may raise a lot but fall a little. Thus, we also use different bloat amount for raise and fall directions. In the following Δd represents the structure

$$\begin{vmatrix} \Delta d_1^- & \Delta d_1^+ \\ \Delta d_2^- & \Delta d_2^+ \\ \vdots & \vdots \\ \Delta d_n^- & \Delta d_n^+ \end{vmatrix}$$

Give the bloat amount Δd, the possible rechable space of a point is a cube as shown in figure 1.8. Thus, given a region describe by a `LP`, the possible reachable space is another *relax* `LP`, as shown in figure 1.9.

The algorithm to compute the relax `LP` is shown in algorithm 3.

However, there as shown in the figure 1.9, the relax `LP` has more constraints than the original one, which can be viewed as the half planes by relaxing the *virtual* bounding box constraints. Thus, we add redundant bounding box constraints to reduce error.

Then, given a `LP` lp and its relatexed `LP` lp^r (or polygon), how to compute the bloat amount Δd in the verify procedure? It is a `LP` problem:

$$min \quad \Delta d \tag{1.1}$$

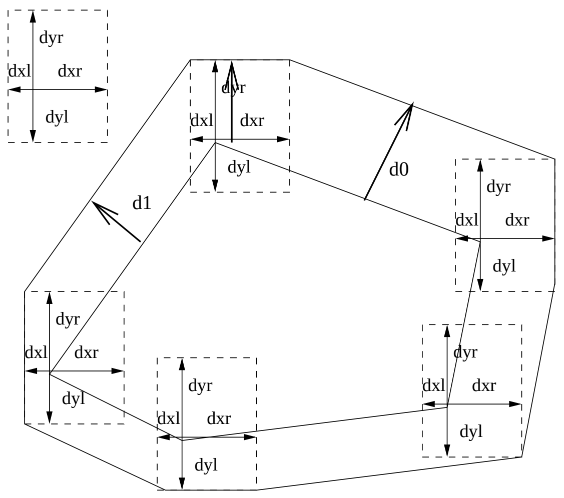
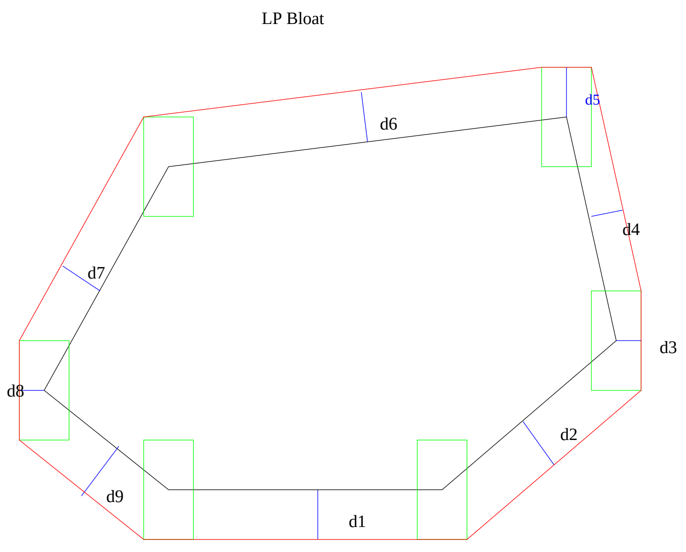


Figure 1.8: The possible reachable space of a point



Given the LP describe the reachable region and its bounding box Ax <= b;

move every point by dx/dy on each direction, the new reachable region is Ax <=b+d

where di is the bloat amount of ith constraint(edge) along its normal.

If the constraint(edge) is ax+by<=c where a^2+b^2=1, di should be (|a|,|b|)*(dx,dy)'

Therefore, d = |A|*(dx,dy)'. It's easy to extend to high dimensional case.

Figure 1.9: The LP and its relaxed LP

Algorithm 3: The *lp_relax* algorithm

```
Input: A CohoLP $lp$ and bloat amount $\Delta d$
Output: The relaxed lp $lp^r$
1  [A,b] = lp_norm(lp);
2  for each constraint $A_i x \le b_i$ do
3      $d_i = 0$;
4      for each variable $j$ do
5          if $A_{i,j} < 0$ then
6              $d_i = d_i - A_{i,j} \cdot \Delta d_i^-$;
7          else
8              $d_i = d_i + A_{i,j} \cdot \Delta d_i^+$;
9      $b_i = b_i + d_i$;
10 $lp^r = lp\_create(A, b)$;
```

$$s.t \qquad lp^r \in lp\_relax(lp, \Delta d)$$

The coefficient of elements of Δd can be set as equally weighted or use the guessed bloat amount. The algorithm is illustrated in figure 1.10.

1.3.1 Inward and Outward Bloat

On one hand, the model space should be large enough to contain all trajectories from current reachable space (face) within the following Δt time. On the other hand, the model space should be minimized to reduce the linearization error. The current bloatAmt is a function of variables (increase and decrease direction), rather than a face, because it is difficult to guess a bloat amount for each faces. Although we use iterative method to reduce bloat amount for each face when creating the model, it is possible the large linearization error caused by initial bloat amount makes *maxDot* large thus has little progress, and the maximum derivative might happens far behinds (in front of) the face we care about.

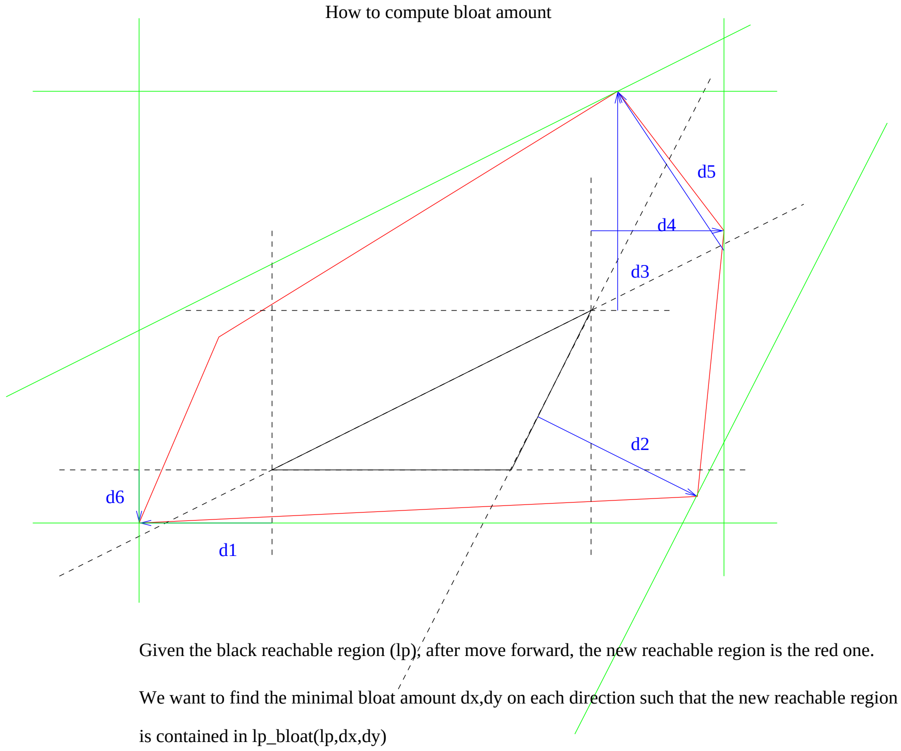
A typical case is shown in figure 1.11, all faces moves inward, however, the bloat amount is large on each direction, thus the red region is used to compute the model.

Thus, we can use different bloat amount for inward and outward direction. Because inward and outward is specified to a face, thus, we focus a face rather than a general LP in the following. The Δd is

$$\begin{vmatrix} \overset{\rightarrow\leftarrow}{\Delta d_1^-} & \overset{\rightarrow\leftarrow}{\Delta d_1^+} & \overset{\leftarrow\rightarrow}{\Delta d_1^-} & \overset{\leftarrow\rightarrow}{\Delta d_1^+} \\ \overset{\rightarrow\leftarrow}{\Delta d_2^-} & \overset{\rightarrow\leftarrow}{\Delta d_2^+} & \overset{\leftarrow\rightarrow}{\Delta d_2^-} & \overset{\leftarrow\rightarrow}{\Delta d_2^+} \\ \vdots & \vdots & \vdots & \vdots \\ \overset{\rightarrow\leftarrow}{\Delta d_n^-} & \overset{\rightarrow\leftarrow}{\Delta d_n^+} & \overset{\leftarrow\rightarrow}{\Delta d_n^-} & \overset{\leftarrow\rightarrow}{\Delta d_n^+} \end{vmatrix}$$

Then, given an face f, the relaxed face f^r is shown in figure 1.12. For the

How to compute bloat amount



Given the black reachable region (lp), after move forward, the new reachable region is the red one.

We want to find the minimal bloat amount dx,dy on each direction such that the new reachable region is contained in lp_bloat(lp,dx,dy)

First, we compute the bloat amount of each edge(constraint) along its normal, d by solving a set of lp

then we know |A|*(dx,dy)' >= d, we can solve the minimal dx,dy using lp solver

Figure 1.10: How to compute the bloat amount

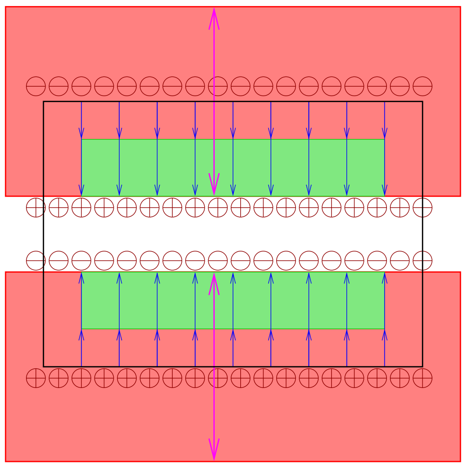


Figure 1.11: The inward and outward bloat amount

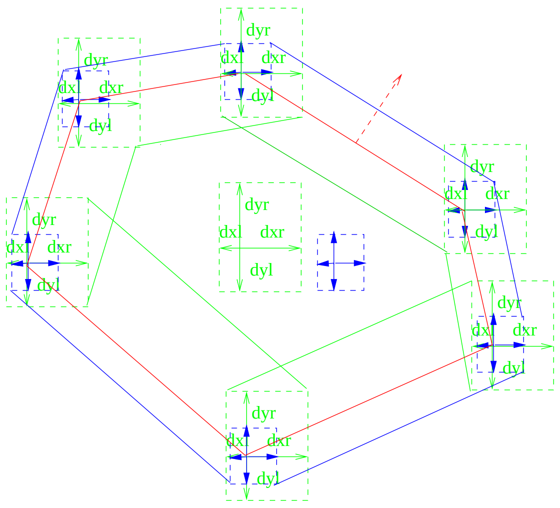


Figure 1.12: Relax `LP` using inward and outward bloat amount

endpoints, we can use the max of inward and outward bloat amount.

Given and face and its projection polygon, the bloat amount is computed as shown in figure 1.13.

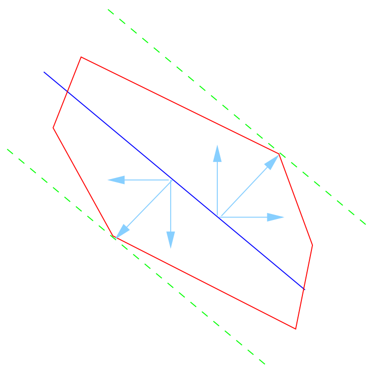


Figure 1.13: How to compute the inward and outward bloat amount