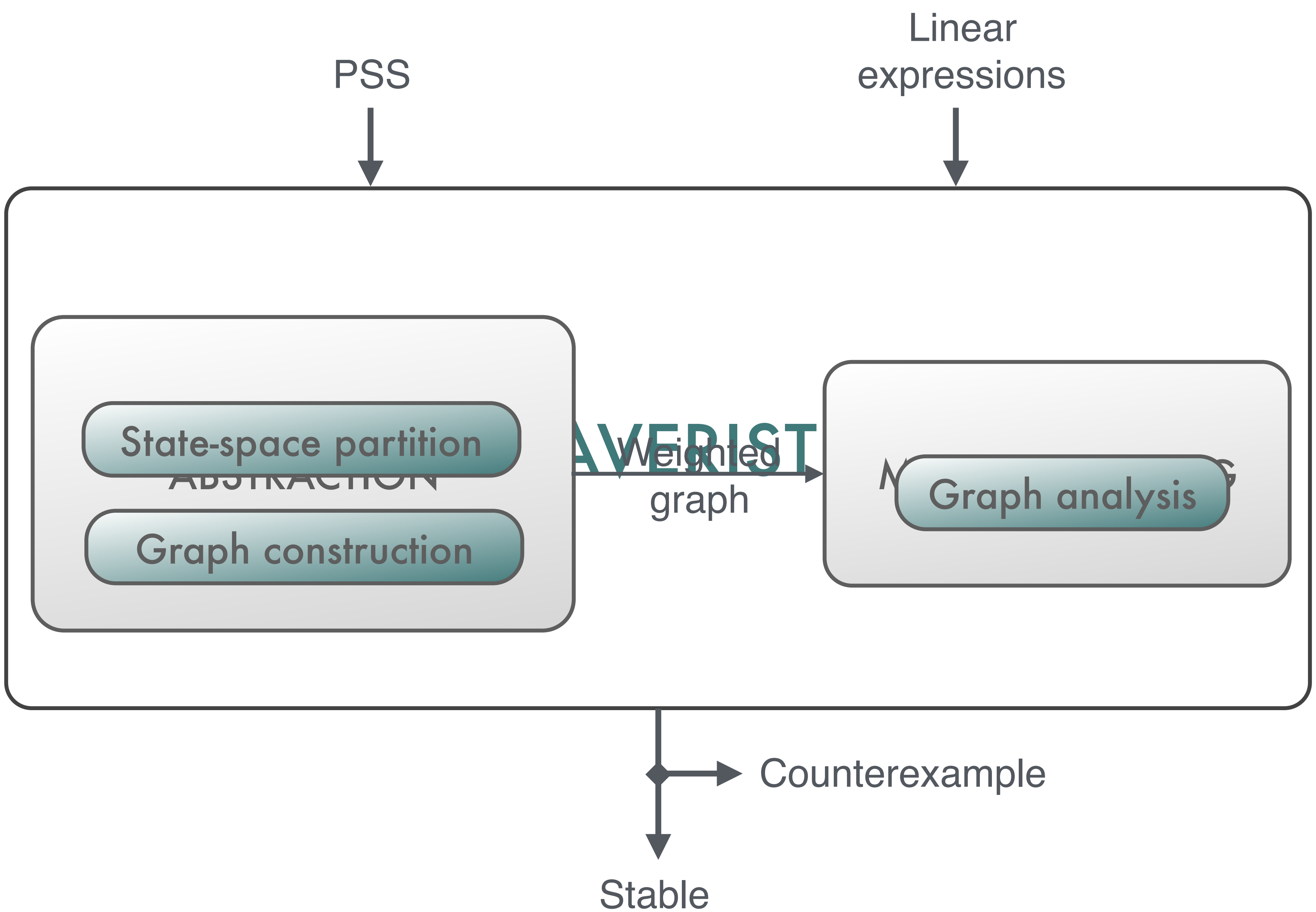
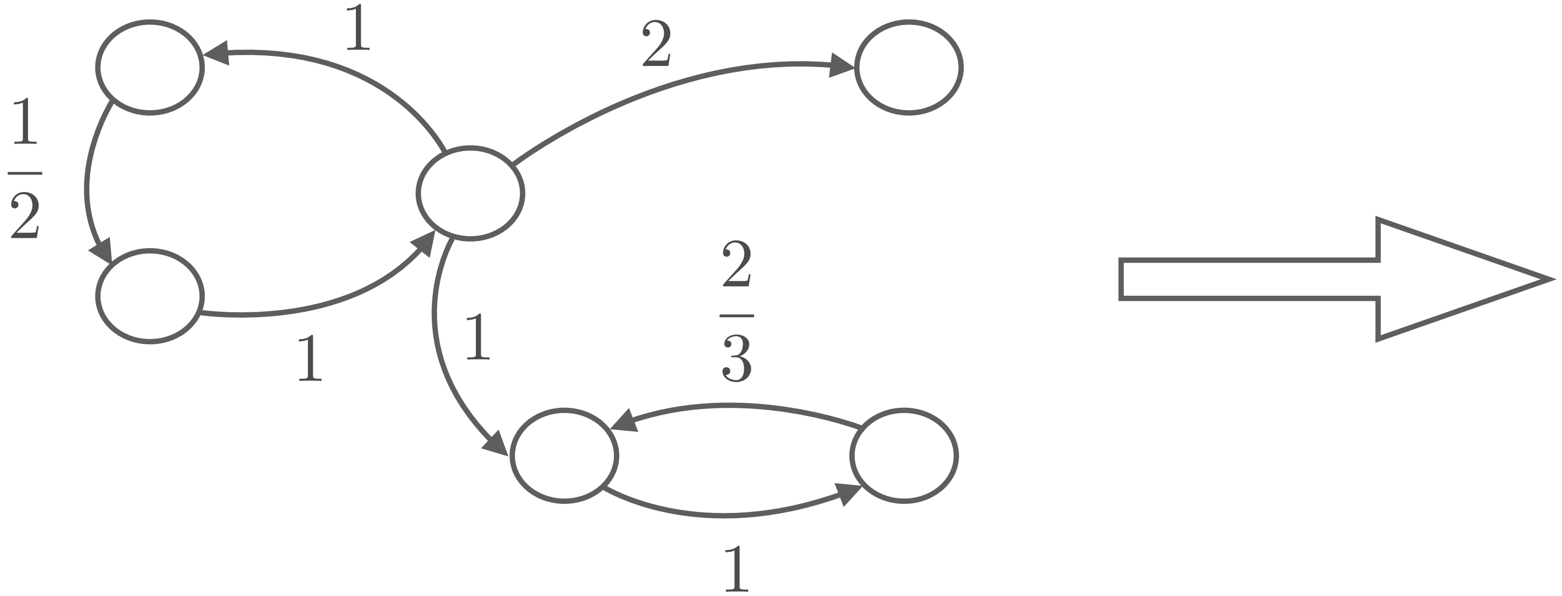
AVERIST: An Algorithmic VERifier for STability

AVERIST ARCHITECTURE

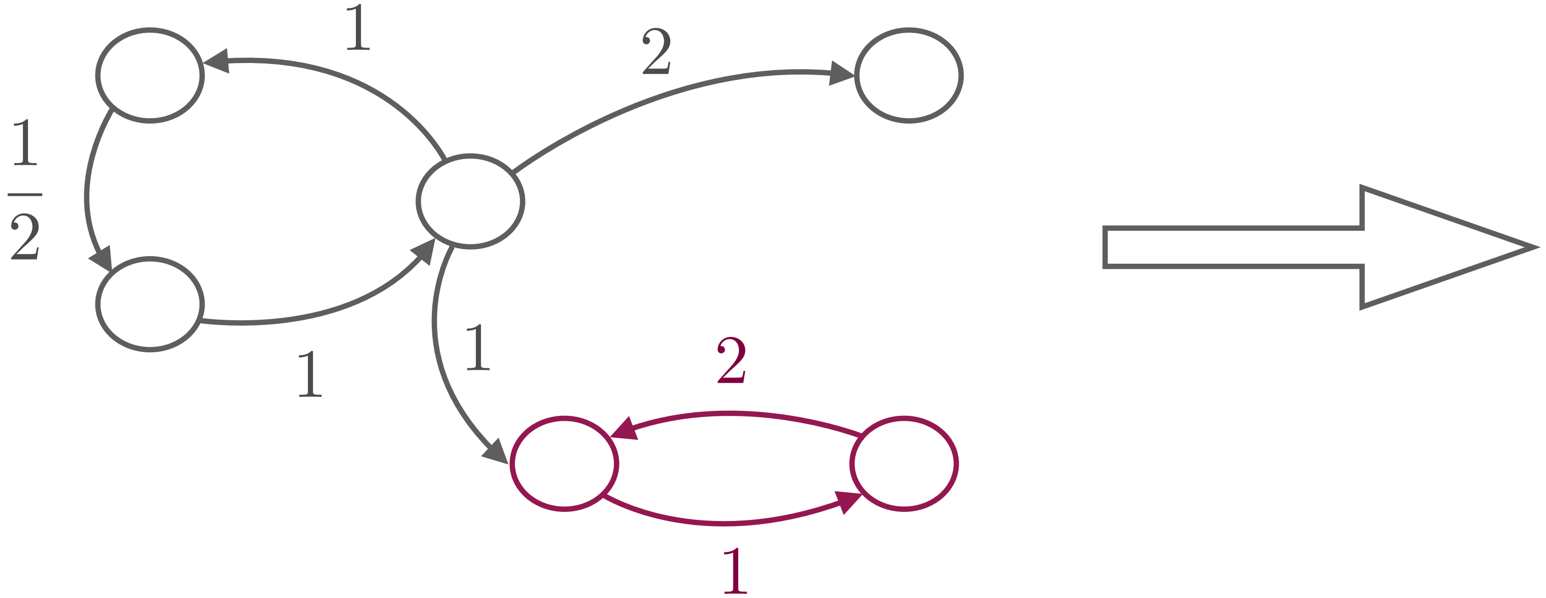
AVERIST architecture



Model-checking



$\mathcal{H}$ stable



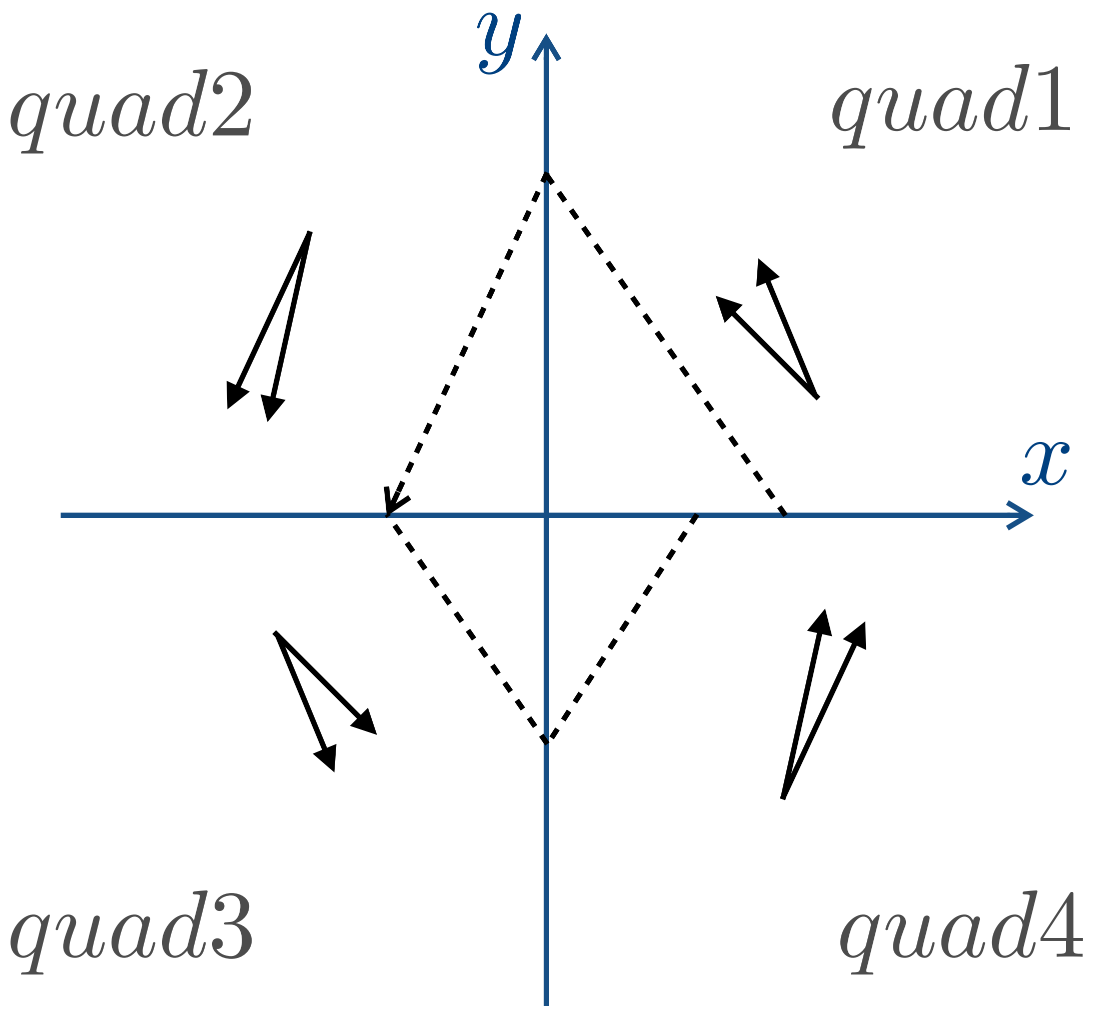
Abstract counterexample

Interactive dialog

```
sage: load('Main.py')

* Please specify the path for the folder in which the experiment data (input.dat) is stored:
/Users/mgarcia/Experiment

* Do you want the linear expressions for creating the regions to be generated automatically
(A) or do you want to add them manually (M)? Enter A/M:
A

* The linear expressions will be generated in a uniform fashion. Please specify the granular
ity -- a natural number (higher number indicates finer partition):
0

* In addition, do you want to add the linear expressions appearing in the input hybrid autom
aton? Enter Y/N:
N

STABILITY ANSWER = Stable
```

ANALYSIS RESULTS

Stable PSS



Stable PSS

Polyhedral switched system

```
var :  x,y;
location:  quad1, quad2, quad3, quad4;
loc:  quad1;
    inv:  x >= 0 AND y >= 0;
    dyn:  dx == -1 AND dy >= 1 AND dy <= 2;
    guards:
        when x == 0 goto quad2;
loc:  quad2;
    inv:  x <= 0 AND y >= 0;
    dyn:  dx >= -2 AND dx <= -1 AND dy == -4;
    guards:
        when y == 0 goto quad3;
loc:  quad3;
    inv:  x <= 0 AND y <= 0;
    dyn:  dx == 1 AND dy <= -1 AND dy >= -2;
    guards:
        when x == 0 goto quad4;
loc:  quad4;
    inv:  x >= 0 AND y <= 0;
    dyn:  dx >= 1 AND dx <= 2 AND dy == 4;
    guards:
        when y == 0 goto quad1;
```

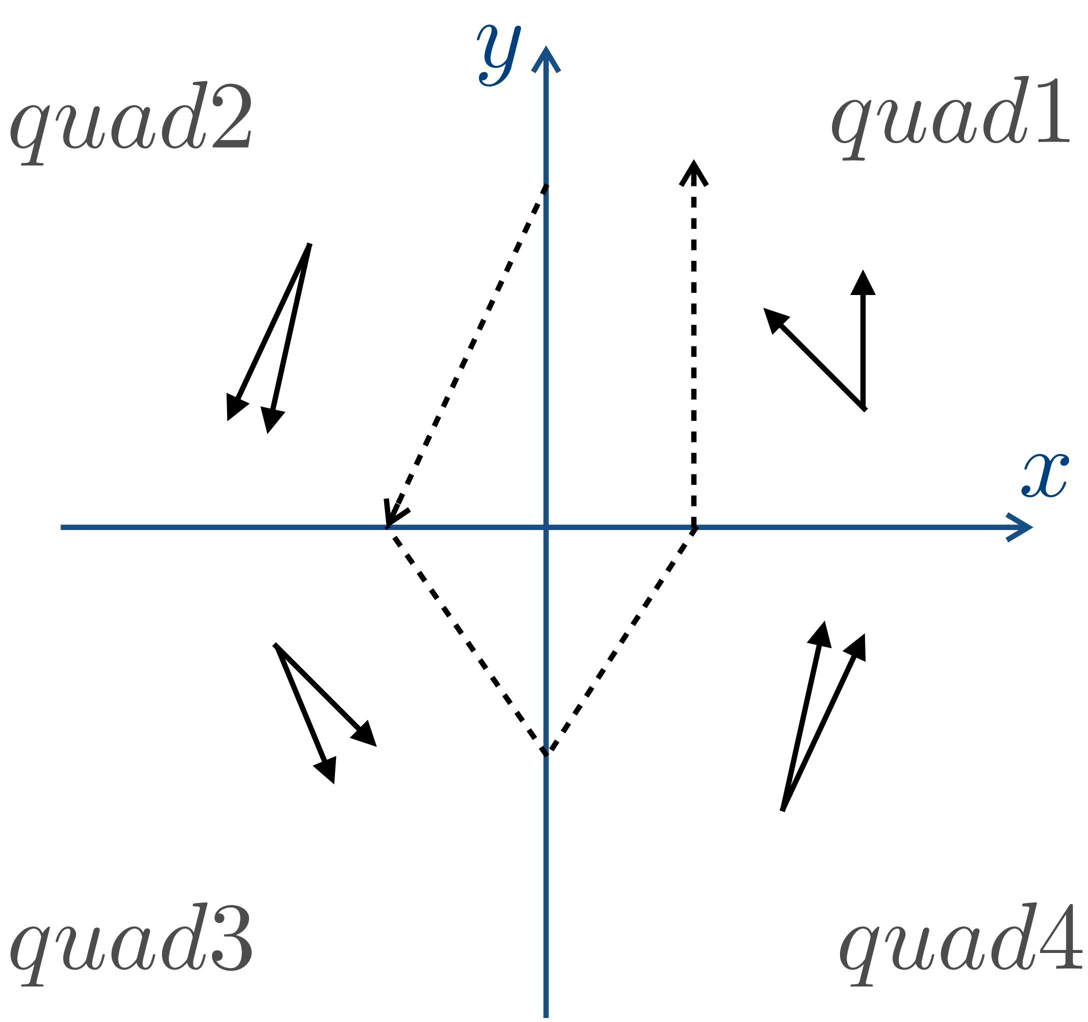
AVERIST

STABILITY ANSWER = Stable

Linear expressions

x=0, y=0

Unstable PSS - Blow-up



Unstable PSS

Polyhedral switched system

```
var :  x,y;
location:  quad1, quad2, quad3, quad4;
loc:  quad1;
    inv:  x >= 0 AND y >= 0;
    dyn:  dx >= -1 AND dx <= 0 AND dy == 1;
    guards:
        when x == 0 goto quad2;
loc:  quad2;
    inv:  x <= 0 AND y >= 0;
    dyn:  dx >= -2 AND dx <= -1 AND dy == -4;
    guards:
        when y == 0 goto quad3;
loc:  quad3;
    inv:  x <= 0 AND y <= 0;
    dyn:  dx == 1 AND dy <= -1 AND dy >= -2;
    guards:
        when x == 0 goto quad4;
loc:  quad4;
    inv:  x >= 0 AND y <= 0;
    dyn:  dx >= 1 AND dx <= 2 AND dy == 4;
    guards:
        when y == 0 goto quad1;
```

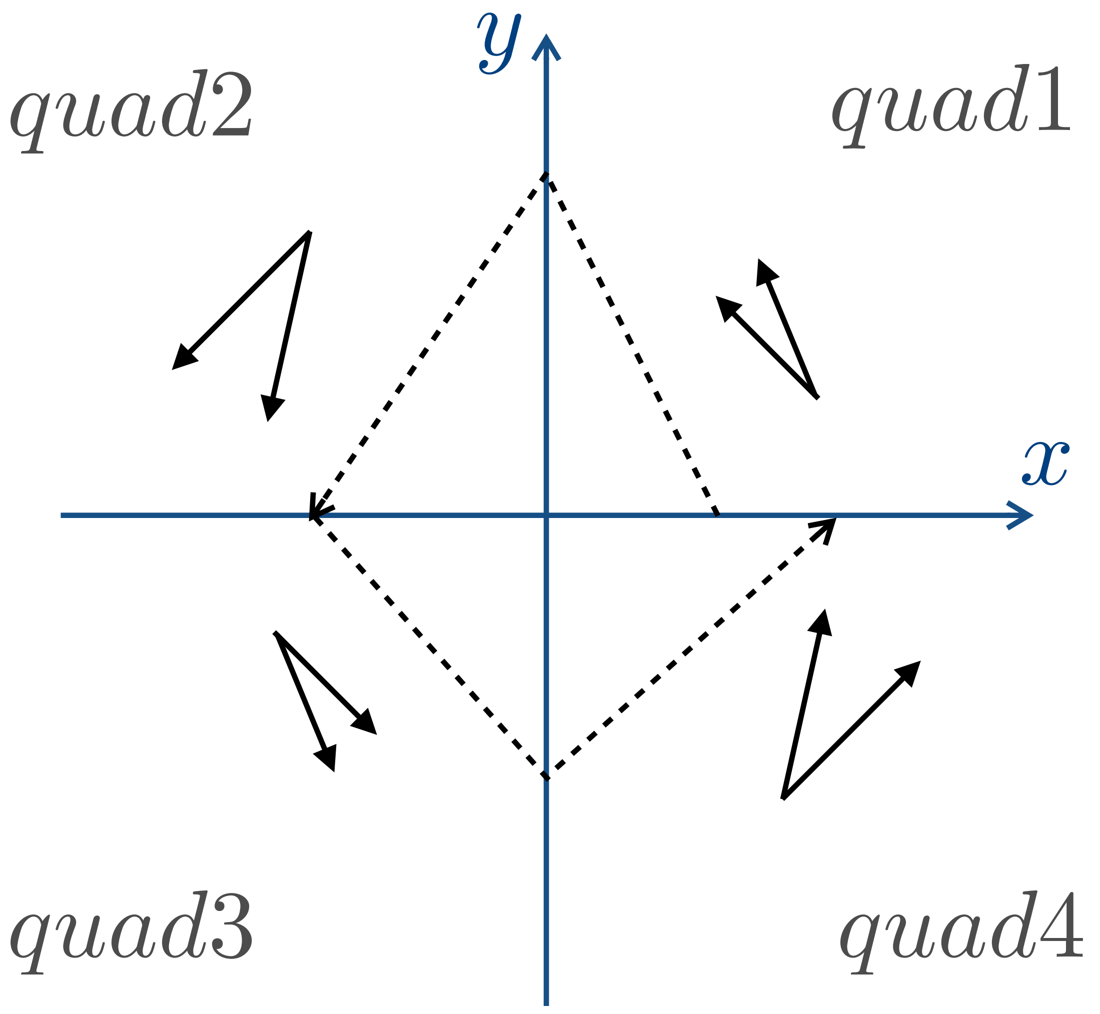
AVERIST

STABILITY ANSWER = Unstable (blow-up)

Linear expressions

x=0, y=0

Unstable PSS - Counterexample



Unstable PSS - Counterexample

Polyhedral switched system

```
var :  x,y;
location:  quad1, quad2, quad3, quad4;
loc:  quad1;
    inv:  x >= 0 AND y >= 0;
    dyn:  dx == -2 AND dy >= 1 AND dy <= 2;
    guards:
        when x == 0 goto quad2;
loc:  quad2;
    inv:  x <= 0 AND y >= 0;
    dyn:  dx >= -2 AND dx <= -1 AND dy == -2;
    guards:
        when y == 0 goto quad3;
loc:  quad3;
    inv:  x <= 0 AND y <= 0;
    dyn:  dx == 1 AND dy <= -1 AND dy >= -2;
    guards:
        when x == 0 goto quad4;
loc:  quad4;
    inv:  x >= 0 AND y <= 0;
    dyn:  dx >= 1 AND dx <= 2 AND dy == 2;
    guards:
        when y == 0 goto quad1;
```

Linear expressions

x=0, y=0

AVERIST

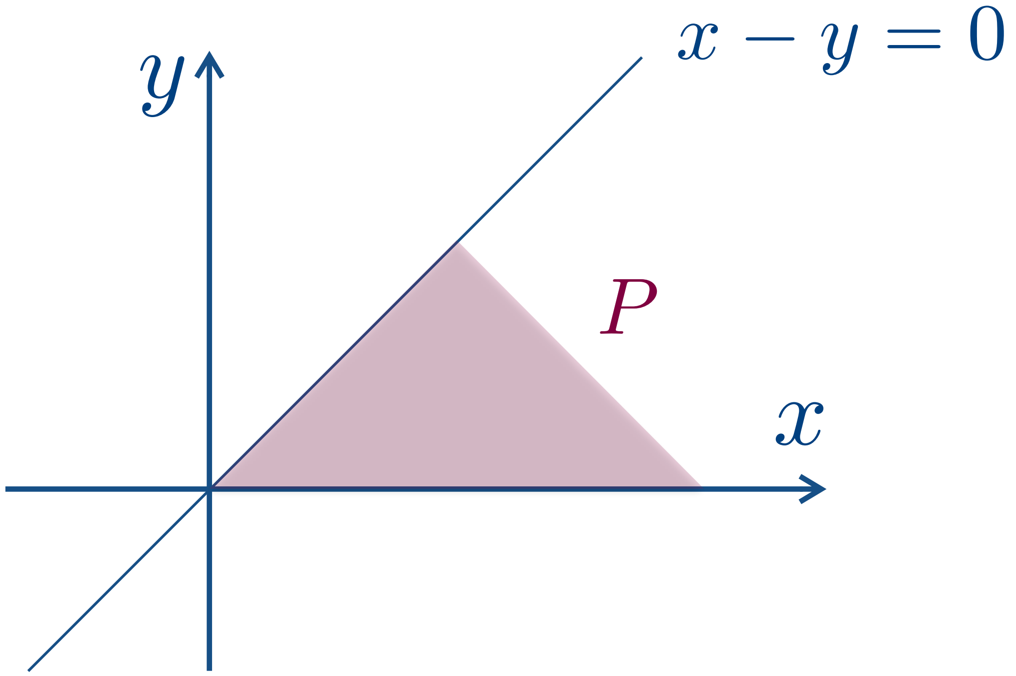
STABILITY ANSWER = Abstract counterexample

```
[('quad2', 'Constraint_System {x1==0, -x0>0}'),
 ('quad3', 'Constraint_System {x0==0, -x1>0}'),
 ('quad1', 'Constraint_System {x1==0, x0>0}'),
 ('quad1', 'Constraint_System {x0==0, x1>0}'),

 ('quad2', 'Constraint_System {x1==0, -x0>0}')]
```

DEPENDENCIES

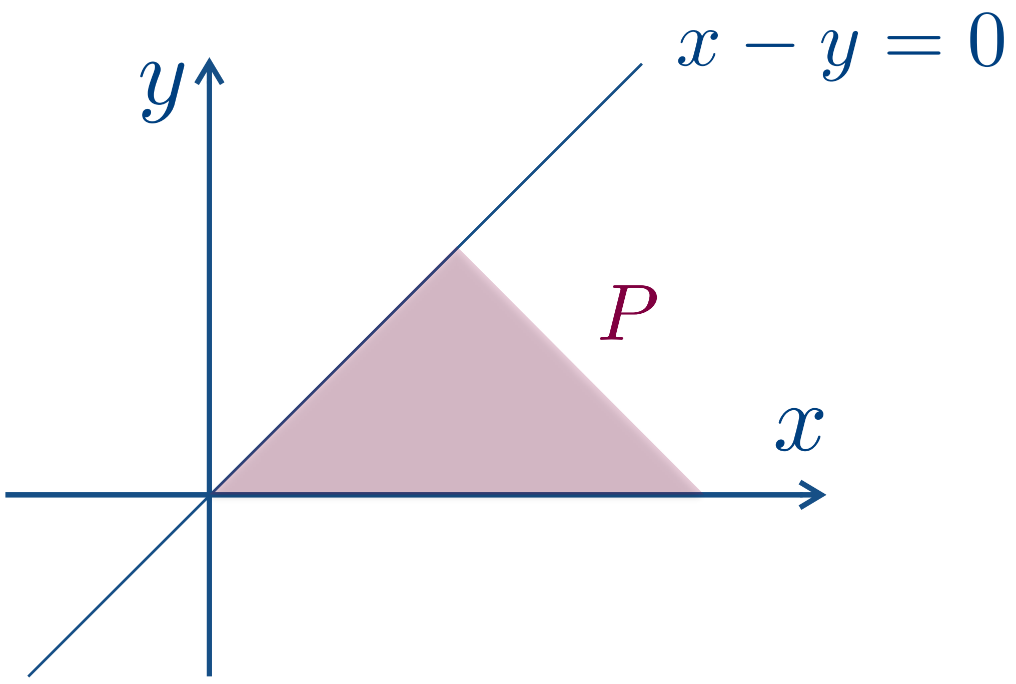
Parma Polyhedra Library - PPL

```
x = Variable(0)
y = Variable(1)
P = NNC_Polyhedron(2,'universe')
P.add_constraint(y>0)
P.add_constraint(x-y>0)
```

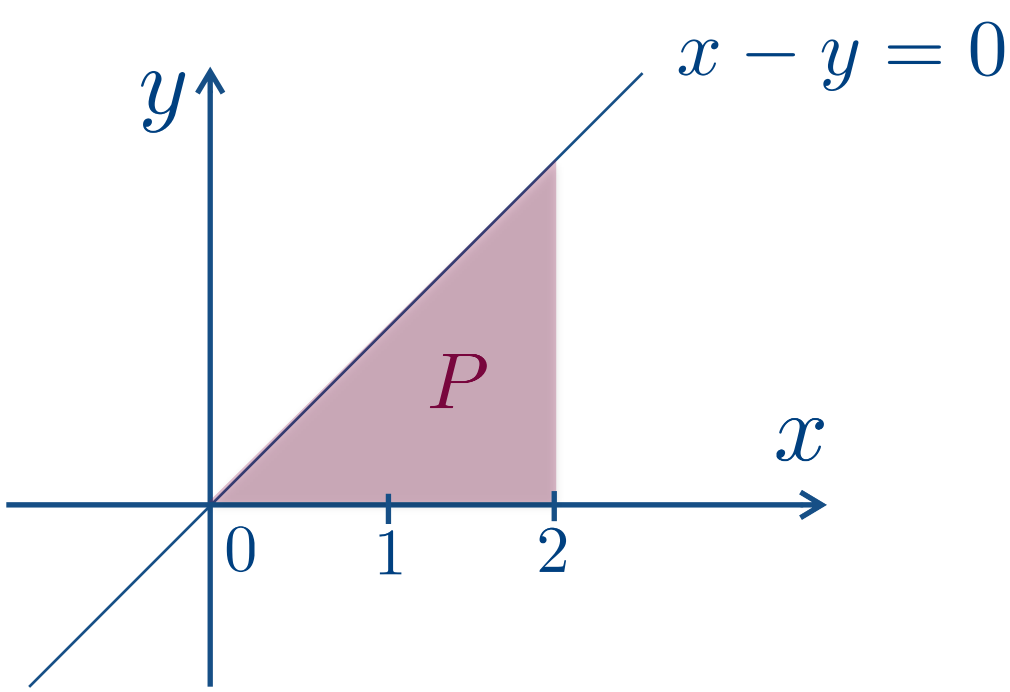


Parma Polyhedra Library - PPL

```
x = Variable(0)
y = Variable(1)
P = NNC_Polyhedron(2,'universe')
P.add_constraint(x>0)
P.add_constraint(x-y>0)
```



GNU Linear Programming Kit - GLPK



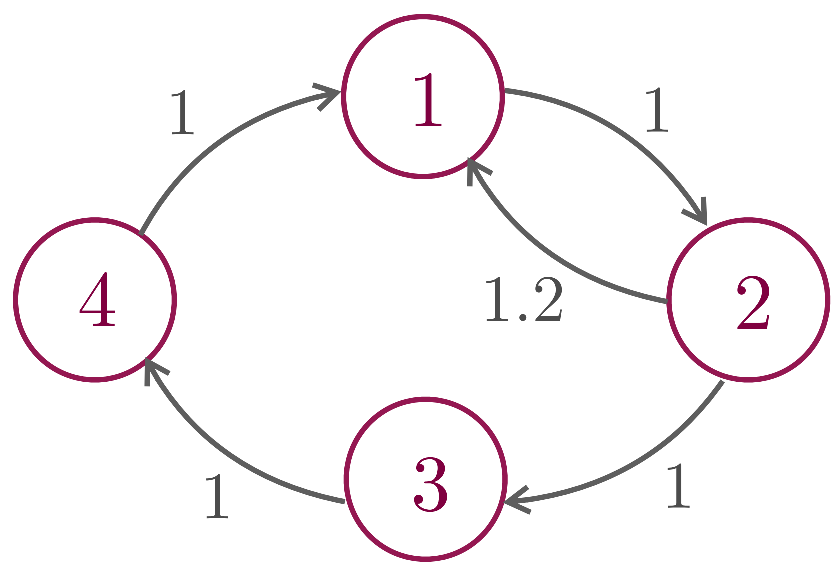

```
sage: P.maximize(1*x)
{'bounded': True,
 'generator': point(2/1, 1/1),
 'maximum': True,
 'sup_d': 1,
 'sup_n': 2}
```

```
sage: P.maximize(1*y)
{'bounded': True,
 'generator': closure_point(2/1, 2/1),
 'maximum': False,
 'sup_d': 1,
 'sup_n': 2}
```

- `'sup_n'`: Integer. The numerator of the supremum value.
- `'sup_d'`: Non-zero integer. The denominator of the supremum value.
- `'maximum'`: Boolean. `True` if and only if the supremum is also the maximum value.
- `'generator'`: a **`Generator`**. A point or closure point where expr reaches its supremum value.

NetworkX

```
import networkx as nx
G=nx.DiGraph()
G.add_nodes_from([1,2,3,4])
G.add_weighted_edges_from([(1,2,1),
(2,3,1),(3,4,1),(4,1,1),(2,1,1.2)])
negative_cycle = greater_than_one_edge_cycle(G)
```

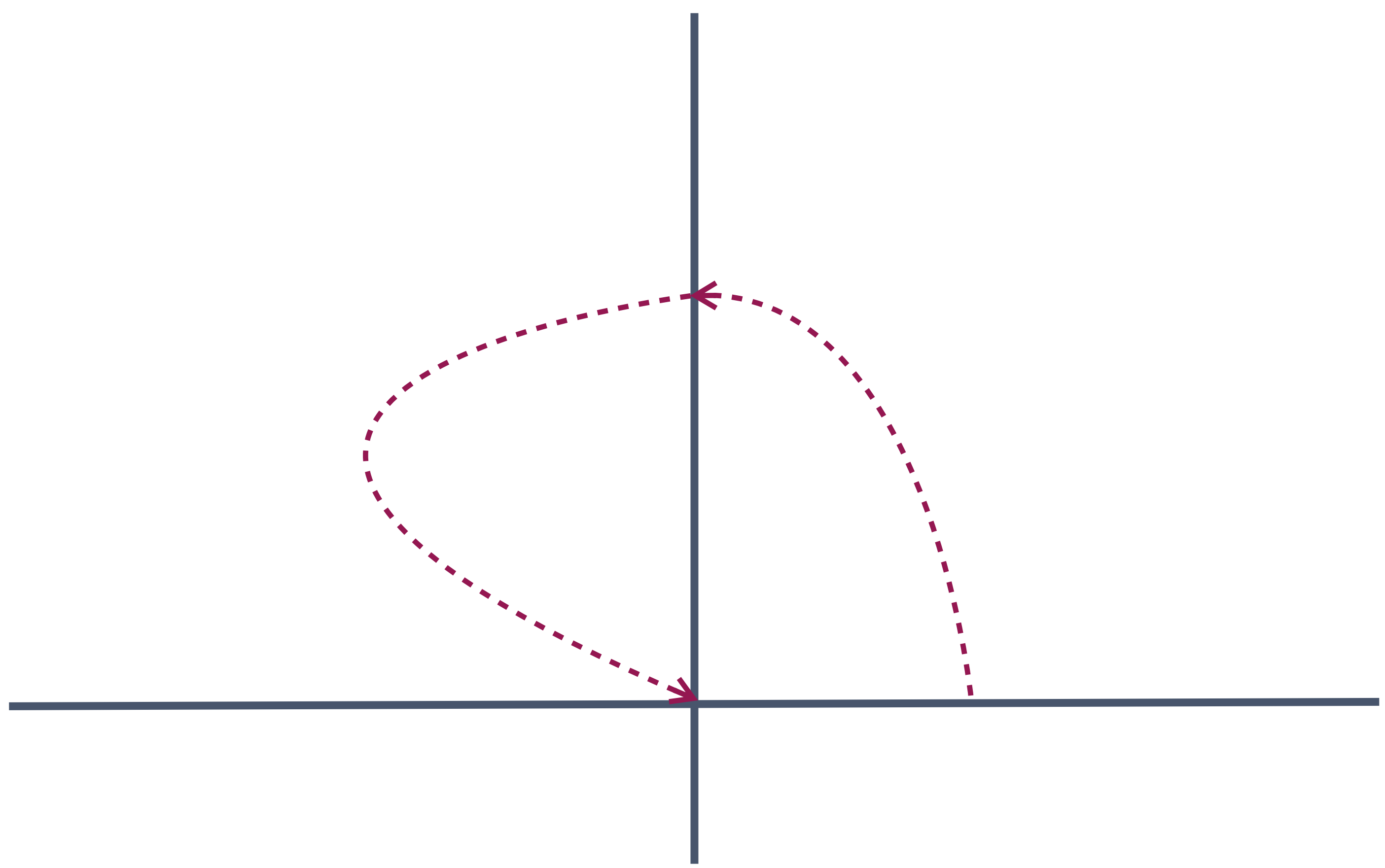


greater_than_one_edge_cycle() uses a modified Bellman-Ford algorithm in order to consider the product of weights instead the sum of them.

HYBRIDIZATION SLIDES

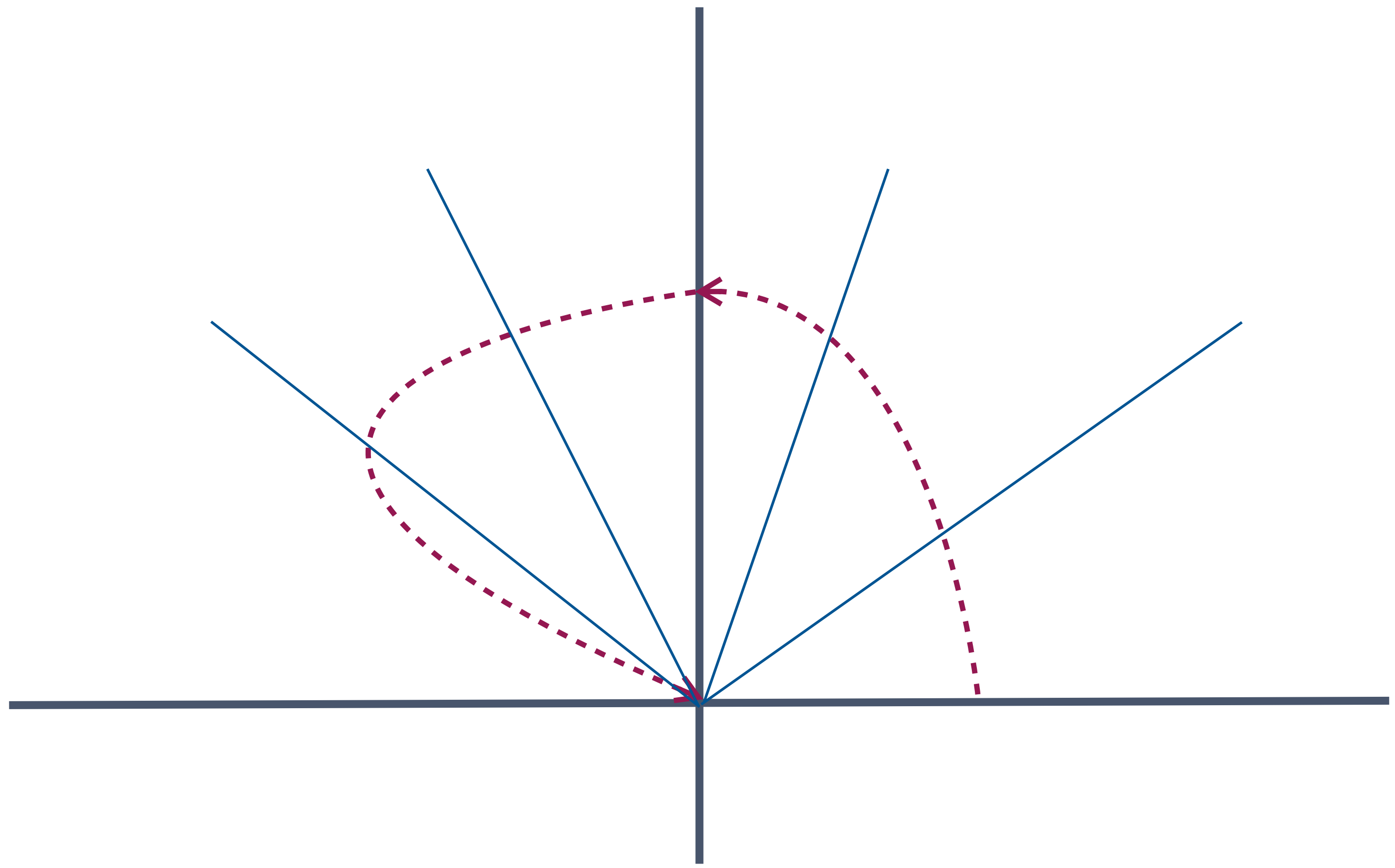
Hybridization

- Hybrid system with linear dynamics is transformed into a hybrid system with polyhedral dynamics.
- Lyapunov (asymptotic) stability is preserved.



Hybridization

- Hybrid system with linear dynamics is transformed into a hybrid system with polyhedral dynamics.
- Lyapunov (asymptotic) stability is preserved.



Hybridization

- Hybrid system with linear dynamics is transformed into a hybrid system with polyhedral dynamics.
- Lyapunov (asymptotic) stability is preserved.