Static Analysis of MPI programs using Abstract Interpretation

Vincent Botbol — CEA LIST/LIP6
Emmanuel Chailloux — LIP6
Tristan Le Gall — CEA LIST

June 15, 2017

GDR-GPL, Montpellier 2017

Motivations

Goals

- Static Analysis of Concurrent Programs
- Proof of (numerical) safety properties
  - Division by zero
  - Arithmetic overflow
  - ...

We want...

- Dynamic creation/destruction of process
- Communications (point-to-point, multicast, reduce, ...)

Context

- Abstract Intepretation
- Regular Model Checking

Abstract Interpretation

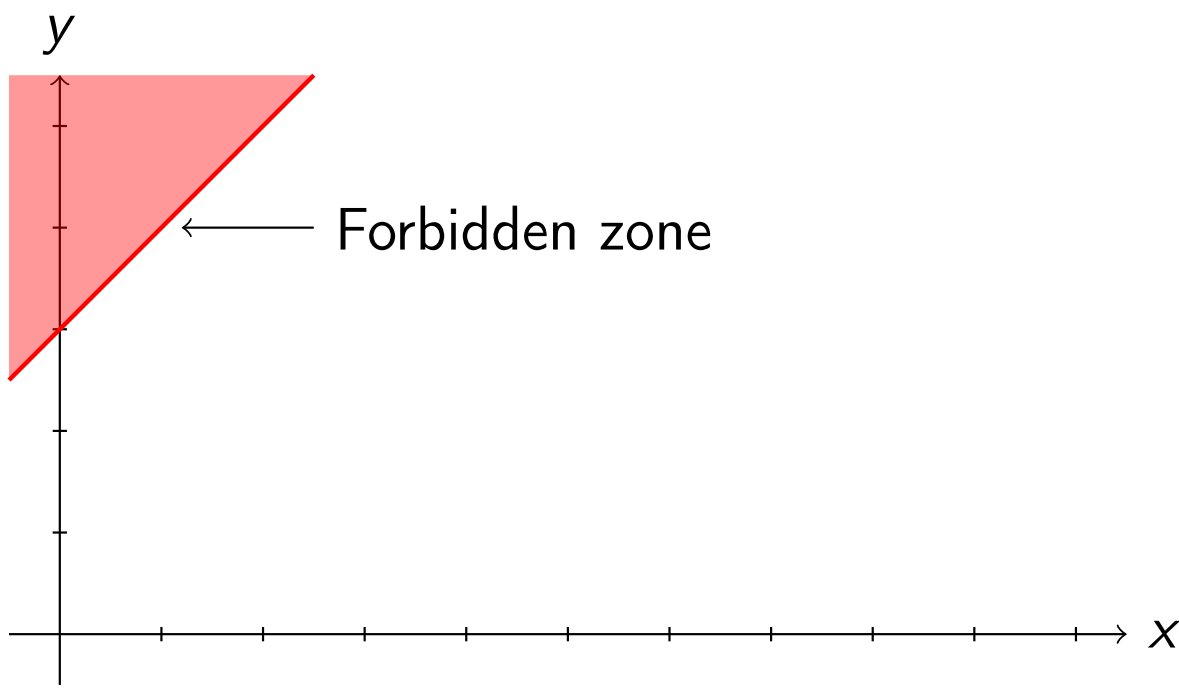
Compute the program set of states

```
1: int x, y
2: y ← 1
3: x ← random(1, 5)
4: while y < 3 and x ≤ 8
   do
5:    x ← x + y
6:    y ← 2 * y
7: x ← x - 1
8: y ← y + 1
```

Abstract Interpretation

Compute the program set of states

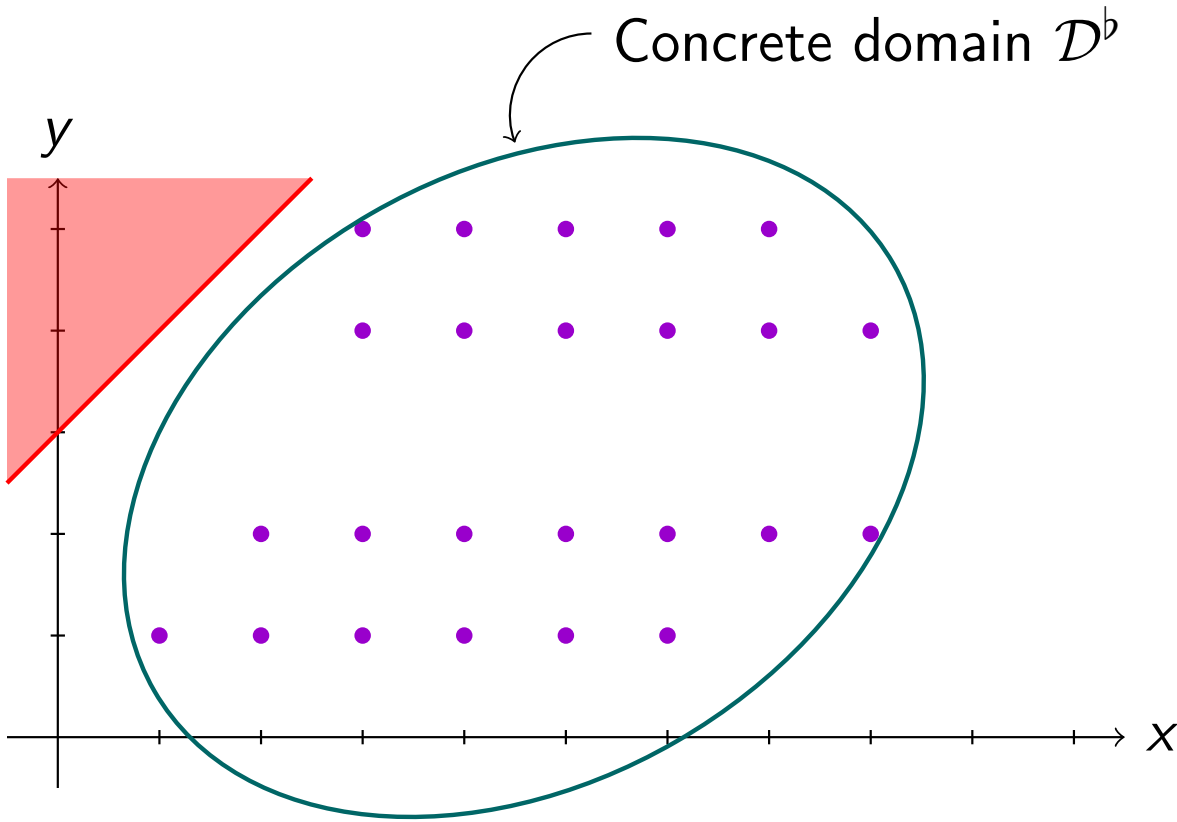
```
1: int x, y
2: y ← 1
3: x ← random(1, 5)
4: while y < 3 and x ≤ 8
   do
5:    x ← x + y
6:    y ← 2 * y
7: x ← x - 1
8: y ← y + 1
```



Abstract Interpretation

Compute the program set of states

```
1: int x, y
2: y ← 1
3: x ← random(1, 5)
4: while y < 3 and x ≤ 8
   do
5:    x ← x + y
6:    y ← 2 * y
7: x ← x - 1
8: y ← y + 1
```



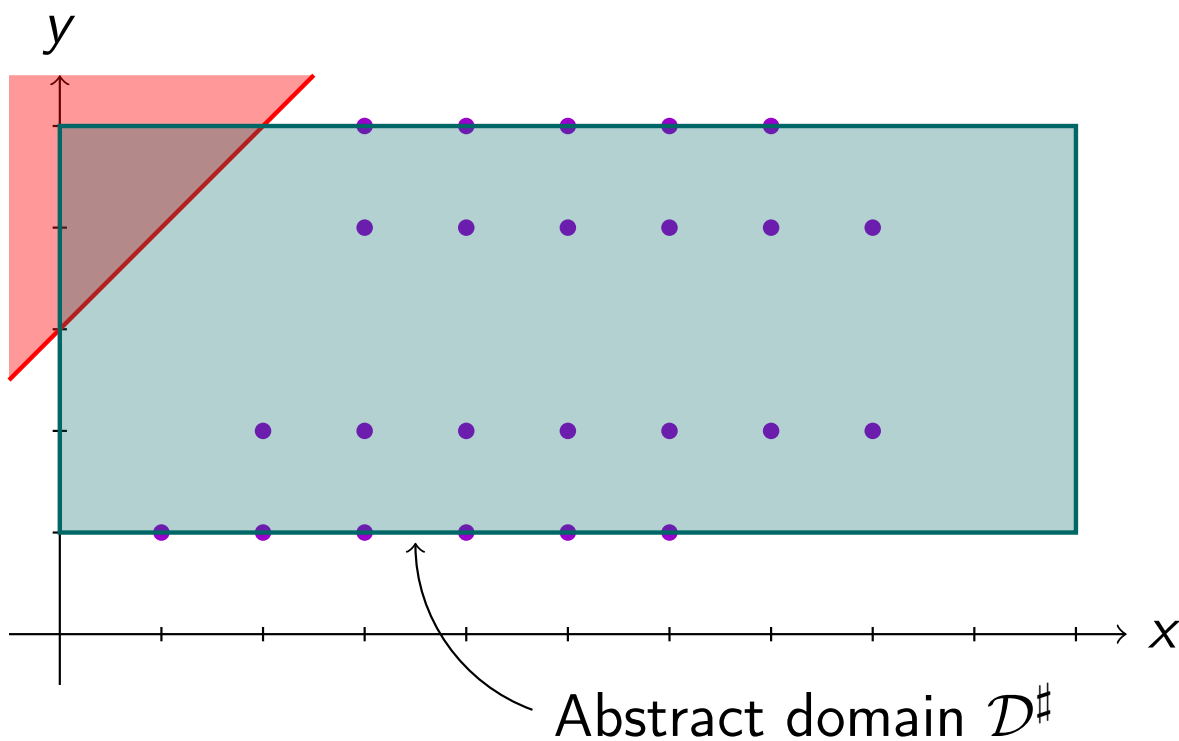
Remark

Computing the concrete domain may be undecidable (or too expensive)

Abstract Interpretation

Compute the program set of states

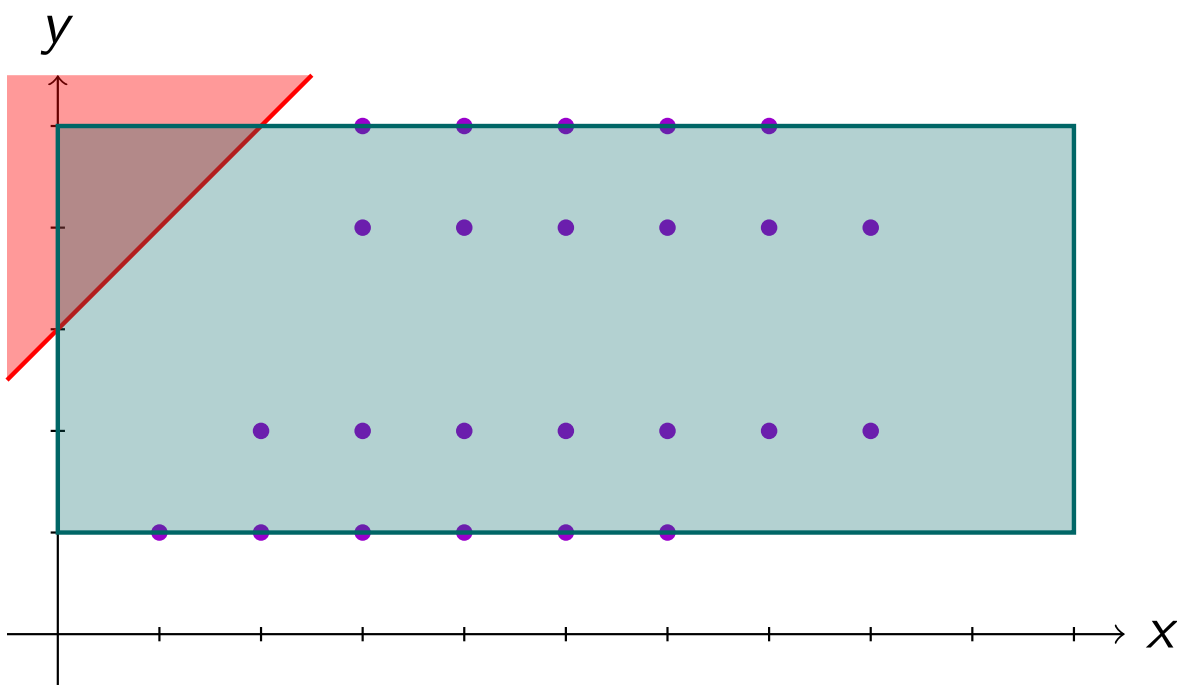
```
1: int x, y
2: y ← 1
3: x ← random(1, 5)
4: while y < 3 and x ≤ 8
   do
5:    x ← x + y
6:    y ← 2 * y
7: x ← x - 1
8: y ← y + 1
```



Abstract Interpretation

Compute the program set of states

```
1: int x, y
2: y ← 1
3: x ← random(1, 5)
4: while y < 3 and x ≤ 8
   do
5:    x ← x + y
6:    y ← 2 * y
7: x ← x - 1
8: y ← y + 1
```

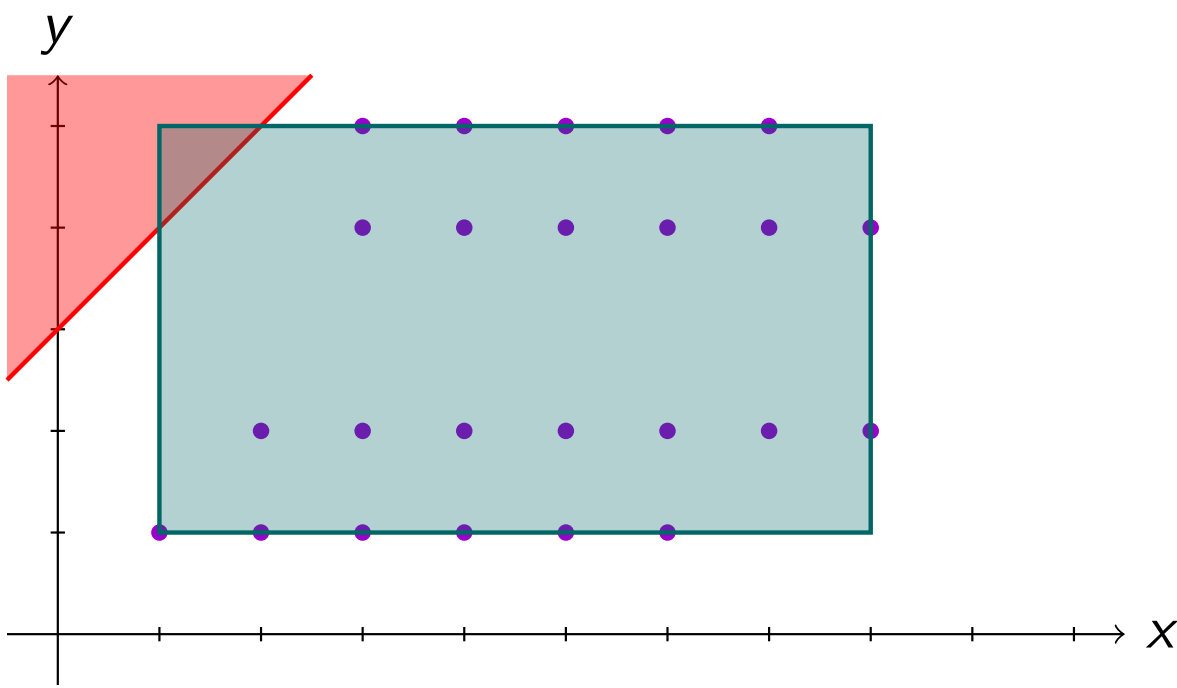


False alarm

Abstract Interpretation

Compute the program set of states

```
1: int x, y
2: y ← 1
3: x ← random(1, 5)
4: while y < 3 and x ≤ 8
   do
5:    x ← x + y
6:    y ← 2 * y
7: x ← x - 1
8: y ← y + 1
```

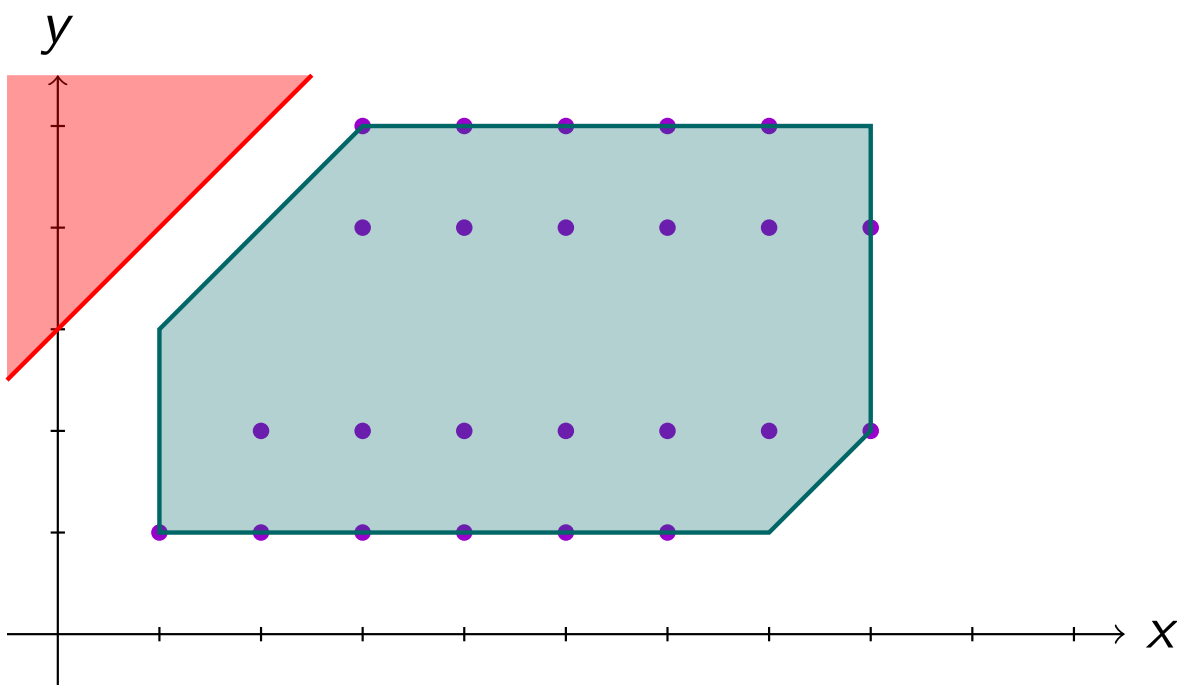


False alarm

Abstract Interpretation

Compute the program set of states

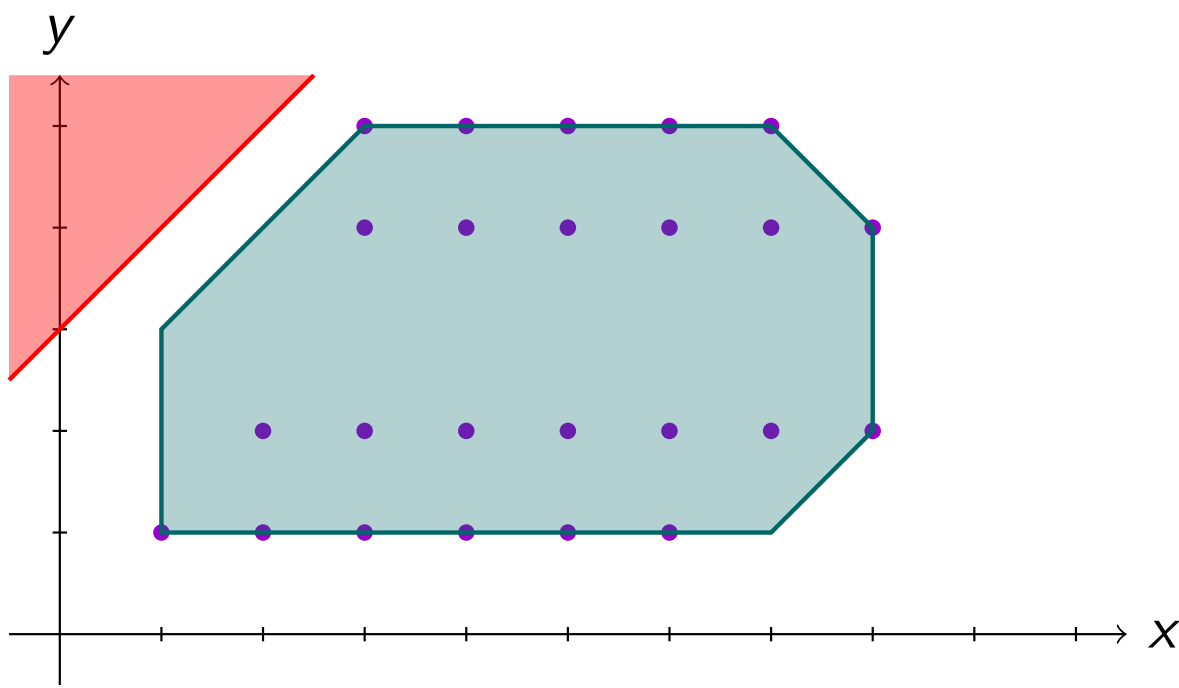
```
1: int x, y
2: y ← 1
3: x ← random(1, 5)
4: while y < 3 and x ≤ 8
   do
5:    x ← x + y
6:    y ← 2 * y
7: x ← x - 1
8: y ← y + 1
```



Abstract Interpretation

Compute the program set of states

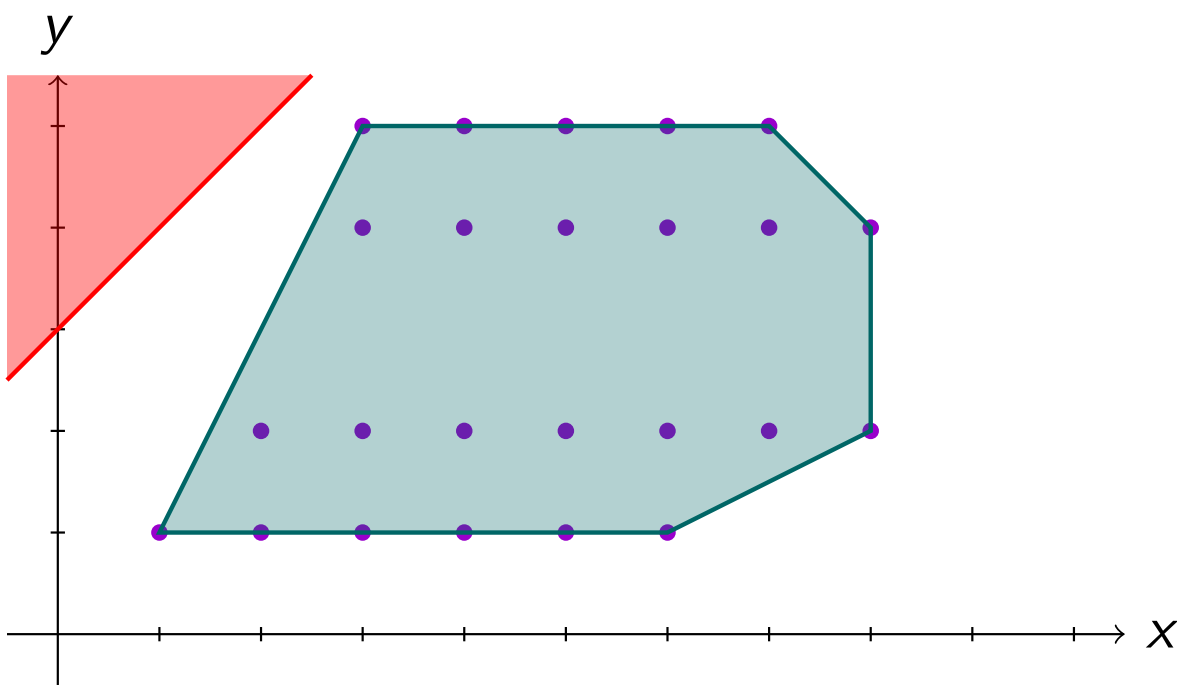
```
1: int x, y
2: y ← 1
3: x ← random(1, 5)
4: while y < 3 and x ≤ 8
   do
5:    x ← x + y
6:    y ← 2 * y
7: x ← x - 1
8: y ← y + 1
```



Abstract Interpretation

Compute the program set of states

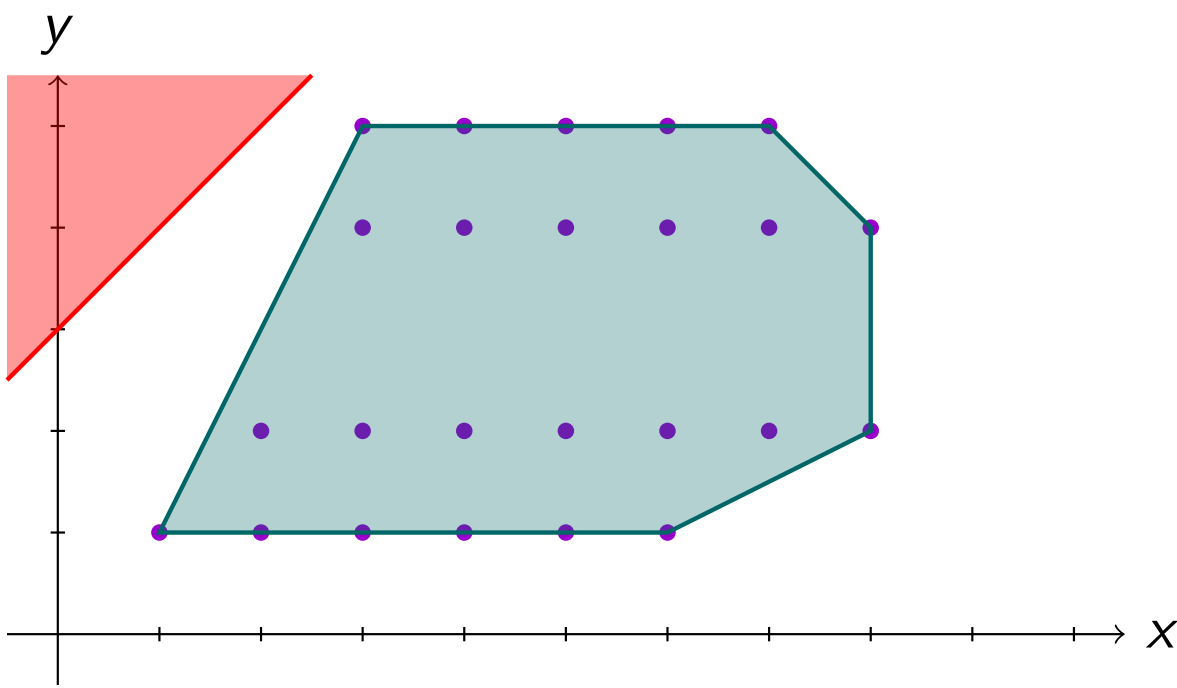
```
1: int x, y
2: y ← 1
3: x ← random(1, 5)
4: while y < 3 and x ≤ 8
   do
5:    x ← x + y
6:    y ← 2 * y
7: x ← x - 1
8: y ← y + 1
```



Abstract Interpretation

Compute the program set of states

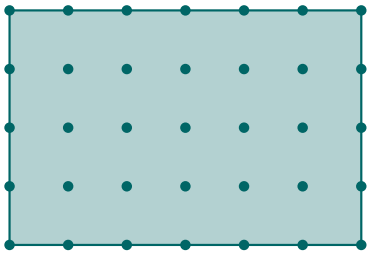
```
1: int x, y
2: y ← 1
3: x ← random(1, 5)
4: while y < 3 and x ≤ 8
   do
5:    x ← x + y
6:    y ← 2 * y
7: x ← x - 1
8: y ← y + 1
```

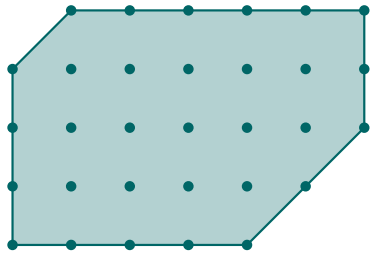


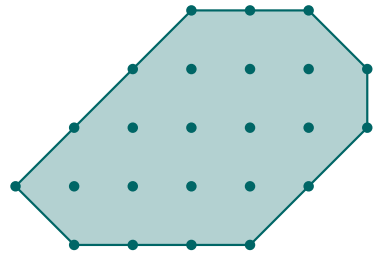
Remark

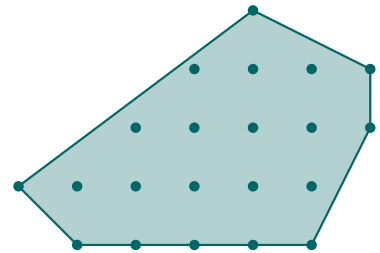
- Approximation with abstract domains
- Precision / Cost

Abstract Interpretation (2)


Intervals


Zones


Octagons


Polyhedra

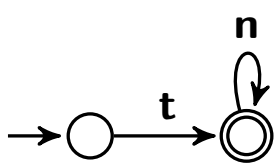
Abstract domains :

- transfer functions $\rho^\sharp$ (affectations, tests, ...)
  - $[\![stmt]\!]^\sharp : \mathbb{D} \to \mathbb{D}$
    e.g. $[\![x \leftarrow random(1, 100)]\!]^\sharp(\sigma^\sharp) = \sigma^\sharp[x \leftarrow [1; 100]]$
- intersection $\sqcap^\sharp$, union $\sqcup^\sharp$ and inclusion $\sqsubseteq^\sharp$
- widening $\triangledown^\sharp : \mathbb{D} \to \mathbb{D} \to \mathbb{D}$

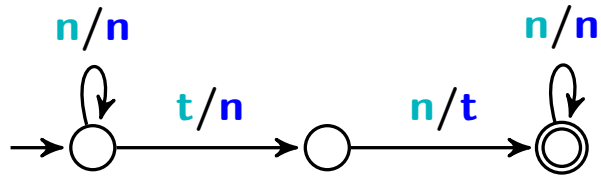
Regular Model Checking

Verification method for infinite-state systems

- System state representation $\Rightarrow$ regular language
- Transition function $\Rightarrow$ rewritting of the language

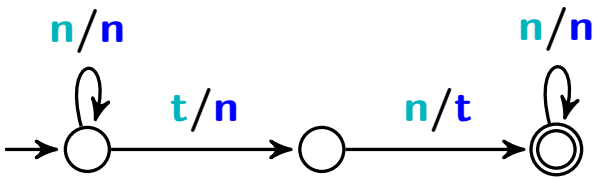


$\mathcal{I}$: System state

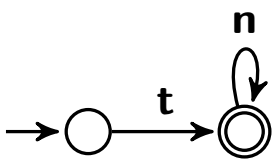


$\mathcal{T}$: Transition fonction

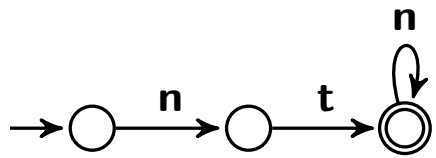
Regular Model Checking (2)



$\mathcal{T}$: Token-Passing Transducer

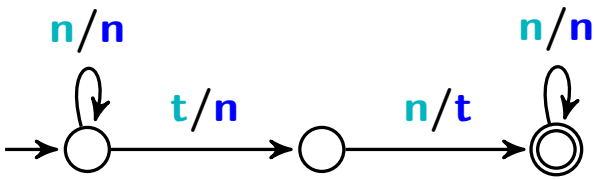


$\mathcal{I}$: Initial Configuration

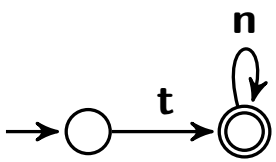


$\mathcal{T}(\mathcal{I})$

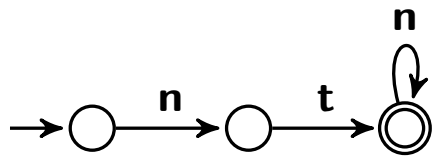
Regular Model Checking (2)



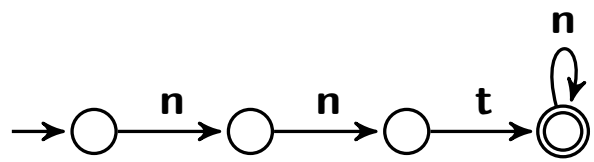
$\mathcal{T}$: Token-Passing Transducer



$\mathcal{I}$: Initial Configuration

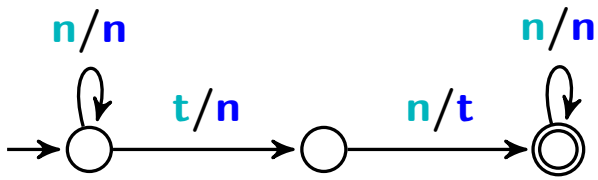


$\mathcal{T}(\mathcal{I})$

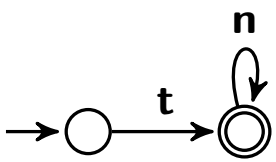


$\mathcal{T}^2(\mathcal{I})$

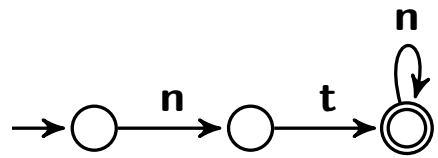
Regular Model Checking (2)



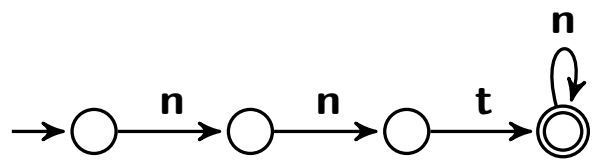
$\mathcal{T}$: Token-Passing Transducer



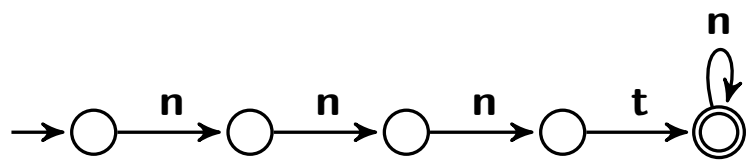
$\mathcal{I}$: Initial Configuration



$\mathcal{T}(\mathcal{I})$

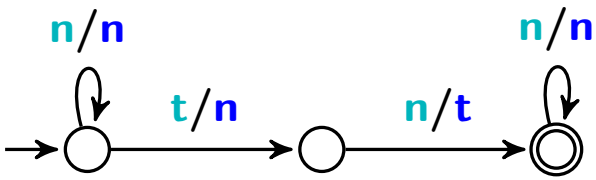


$\mathcal{T}^2(\mathcal{I})$

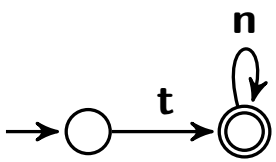


$\mathcal{T}^3(\mathcal{I})$

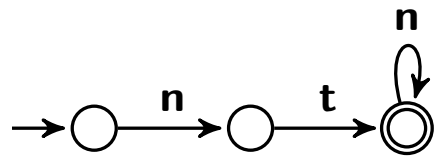
Regular Model Checking (2)



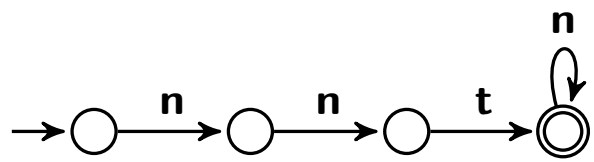
$\mathcal{T}$: Token-Passing Transducer



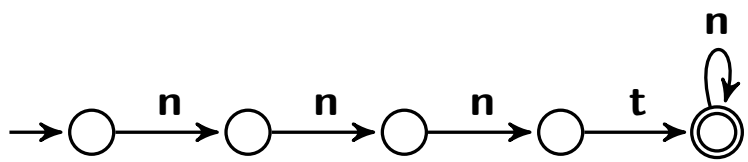
$\mathcal{I}$: Initial Configuration



$\mathcal{T}(\mathcal{I})$



$\mathcal{T}^2(\mathcal{I})$



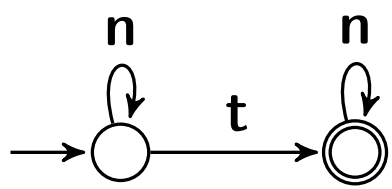
$\mathcal{T}^3(\mathcal{I})$

...

Regular Model Checking (3)

Verification goal

Compute the reachability set of the system :



$$T^* = \bigcup_{i=0}^{\infty} \mathcal{T}^i(\mathcal{I})$$

$\Rightarrow$ Undecidable !

- Semi-algorithms
- Restrict the problem
- Abstractions
- ...

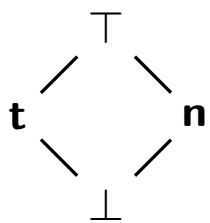
Our method

Use Abs. Int. methodology with the RMC representation

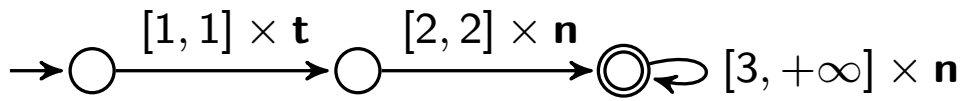
- Abstract domain $\rightarrow$ Lattice Automata
- Transfer functions $\rightarrow$ Rewritting rules

Lattice Automata

- Symbolic Automata
- Parametrized by an abstract domain ($\equiv$ alphabet)
- Transitions label : abstract element



Token lattice



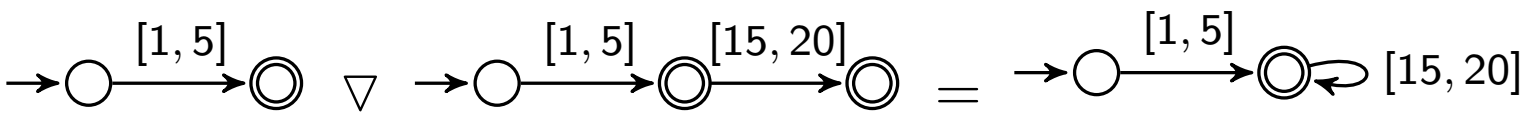
(interval $\times$ token) Lattice Automaton

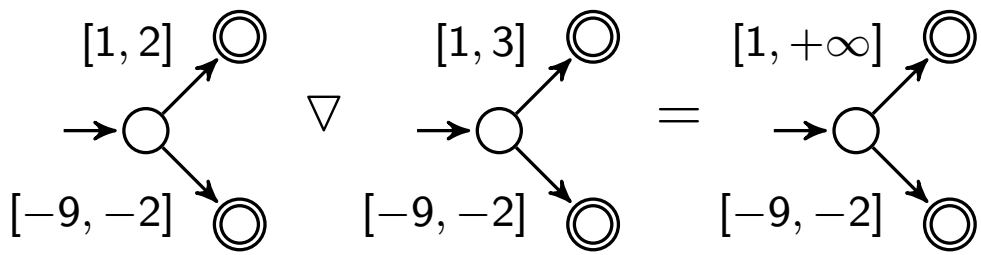
$$\mathcal{L}(A) = ([1,1] \times \mathbf{t}) \cdot ([2,2] \times \mathbf{n}) \cdot ([3,+\infty] \times \mathbf{n})^*$$

Lattice automata (2)

Opérations

- Automata operators : $\subseteq, \cap, \cup, \ldots$
- Widening operator : $\triangledown$
- $\Rightarrow$ Abstract Domain





System's transition function

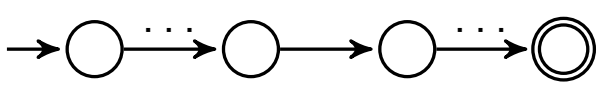
Process local transition

- Classic transfer function

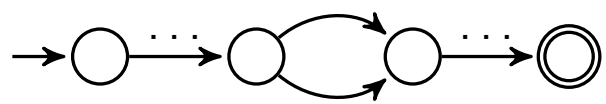
```
1       if (x > 10){
2          ...
3       } else {
4          ...
5       }
6       ...
```

1. $[\texttt{If}] \wedge \{x \in [11, +\infty]\} \ / \ \mathbf{f}(\mathbf{l}, \boldsymbol{\sigma}) \mapsto [\mathrm{L}_2], \sigma$
2. $[\texttt{If}] \wedge \{x \in [-\infty, 10]\} \ / \ \mathbf{f}(\mathbf{l}, \boldsymbol{\sigma}) \mapsto [\mathrm{L}_4], \sigma$

$[If] \times \{x \in [0, 200]\}$



$[L_2] \times \{x \in [11, 200]\}$



$[L_4] \times \{x \in [0, 10]\}$

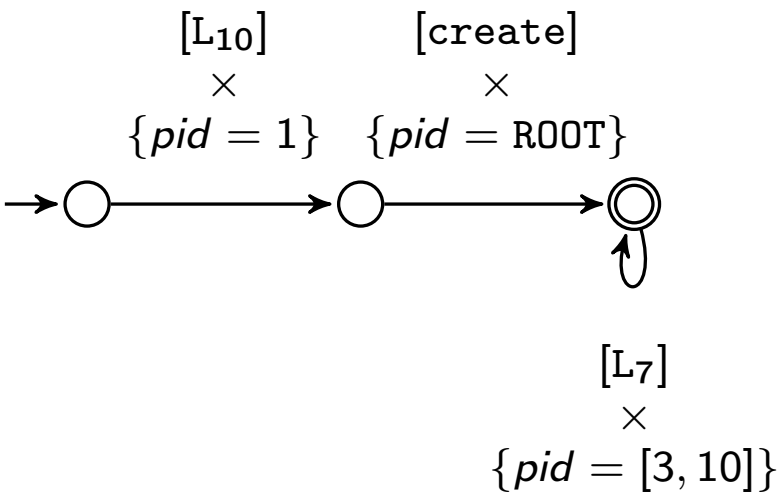
System's transition function (2)

$\Rightarrow$ Global transitions

- Guards : regular expression
- Rewritting functions : rewrites a letter **or a word**

System's transition function – Dynamic process creation

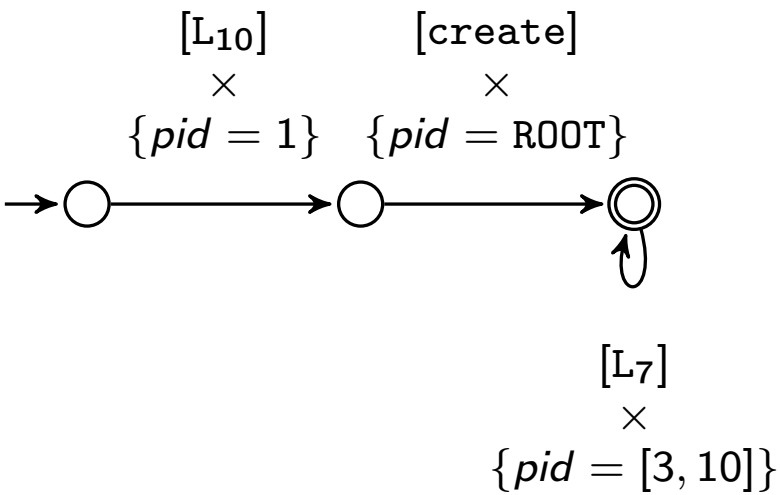
```
1  if (pid = ROOT) {
2    create();
3  }
4  ...
```



System's transition function – Dynamic process creation

```
1  if (pid = ROOT) {
2    create();
3  }
4  ...
```

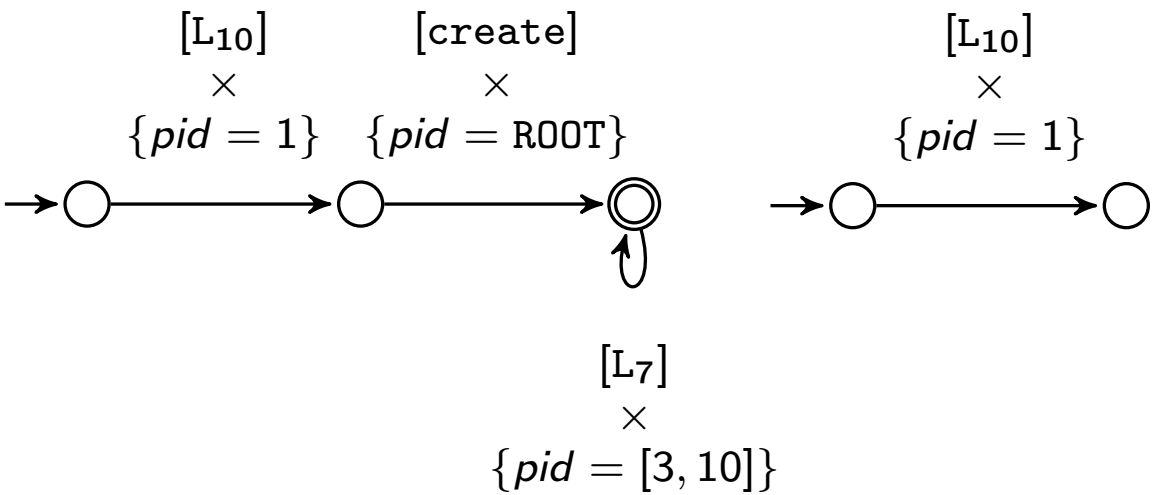
$\top^* \cdot [\texttt{create}] \cdot \top^* /$



System's transition function – Dynamic process creation

```
1  if (pid = ROOT) {
2    create();
3  }
4  ...
```

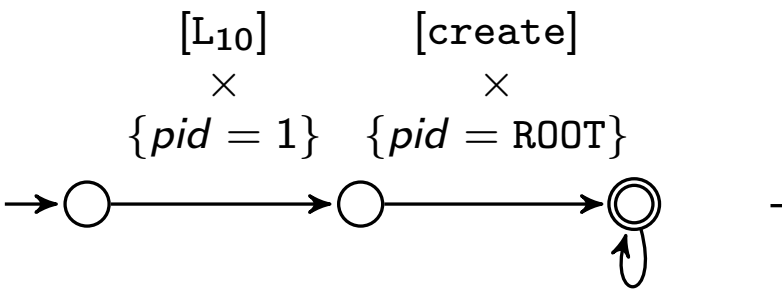
$$\top^* \cdot [\texttt{create}] \cdot \top^* \,/\, \texttt{Id}^* \cdot$$

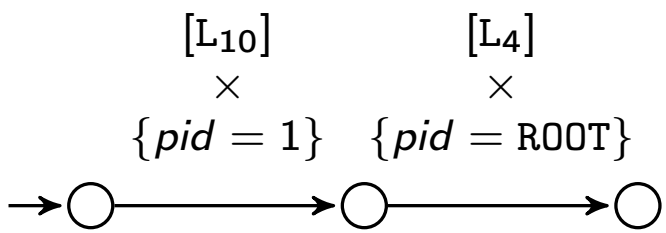


System's transition function – Dynamic process creation

```
1  if (pid = ROOT) {
2    create();
3  }
4  ...
```

$$\top^* \cdot [\texttt{create}] \cdot \top^* \;/\; \texttt{Id}^* \cdot f_1(\langle l, \sigma \rangle) \mapsto \langle [\mathrm{L}_4], \sigma \rangle \cdot$$





$$[\mathrm{L}_7] \times \{pid = [3, 10]\}$$

System's transition function – Dynamic process creation

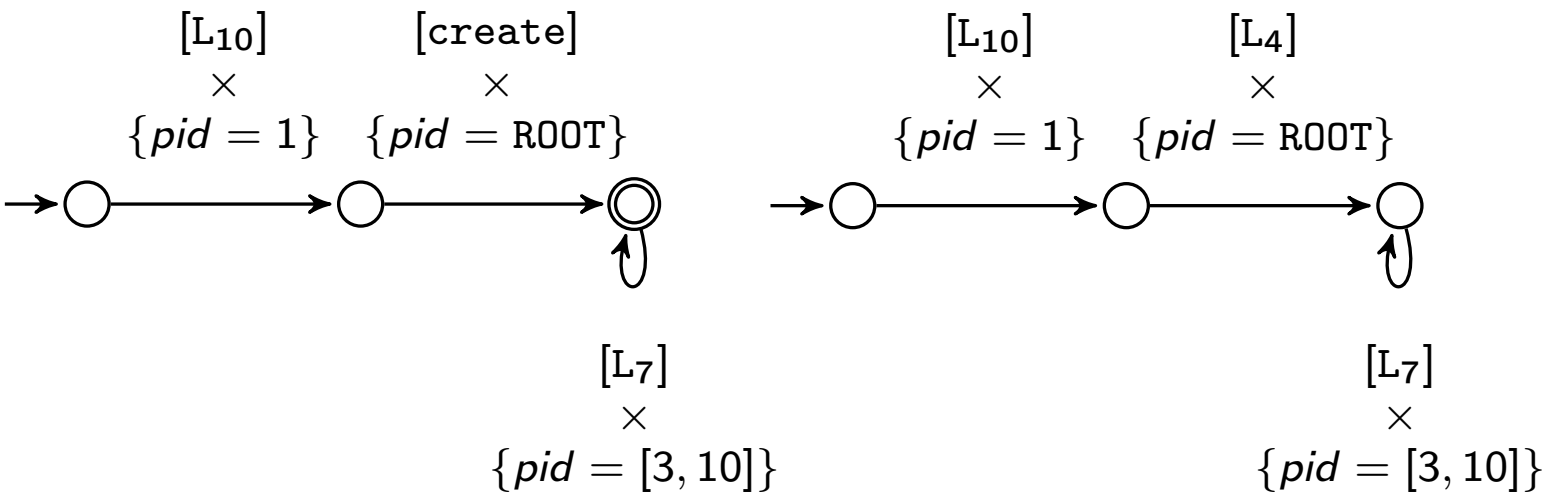
```
1  if (pid = ROOT) {
2    create();
3  }
4  ...
```

$$\top^* \cdot [\texttt{create}] \cdot \top^* \;/$$

$$\texttt{Id}^* \cdot$$

$$f_1(\langle l, \sigma \rangle) \mapsto \langle [\mathrm{L}_4], \sigma \rangle \cdot$$

$$\texttt{Id}^* \cdot$$



System's transition function – Dynamic process creation

```
1  if (pid = ROOT) {
2    create();
3  }
4  ...
```

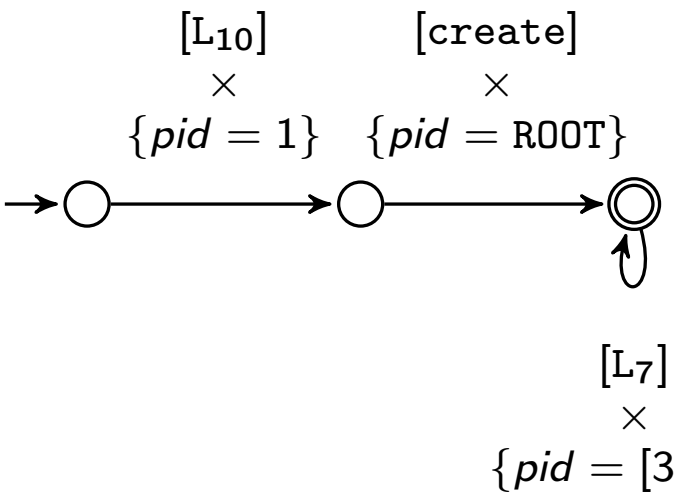
$$\top^* \cdot [\texttt{create}] \cdot \top^* \;/$$

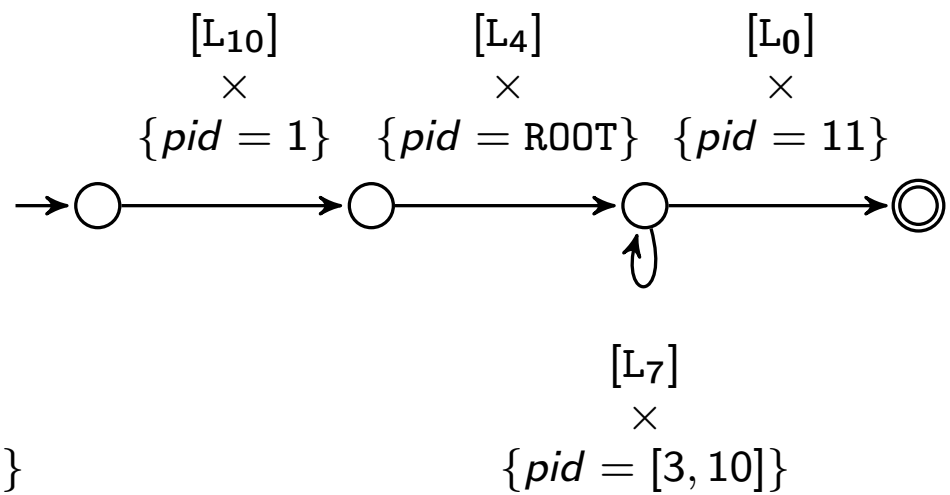
$$\texttt{Id}^* \cdot$$

$$f_1(\langle l, \sigma \rangle) \mapsto \langle [\mathrm{L}_4], \sigma \rangle \cdot$$

$$\texttt{Id}^* \cdot$$

$$f_2(\langle l, \sigma \rangle) \mapsto \langle [L_0], \sigma_\emptyset[\texttt{pid} \leftarrow \texttt{fresh\_pid}] \rangle$$





System's transition function – Point-to-point communication

```
1     if (pid = 1) {
2       x := 3;
3       send(x, 4);
4     }
5     ...
6     if (pid = 4){
7       receive(x, 1);
8     }
9     ...
```

System's transition function – Point-to-point communication

$$\top^* \cdot ([\mathrm{L}_3] \wedge \mathit{pid} = 1) \cdot \top^* \cdot ([\mathrm{L}_7] \wedge \mathit{pid} = 4) \cdot \top^* /$$

```
1    if (pid = 1) {
2      x := 3;
3      send(x, 4);
4    }
5    ...
6    if (pid = 4){
7      receive(x, 1);
8    }
9    ...
```

System's transition function – Point-to-point communication

$$\top^* \cdot ([\mathrm{L}_3] \wedge \mathit{pid} = 1) \cdot \top^* \cdot ([\mathrm{L}_7] \wedge \mathit{pid} = 4) \cdot \top^* /$$

$$\mathtt{Id}^* \cdot$$

```
1    if (pid = 1) {
2      x := 3;
3      send(x, 4);
4    }
5    ...
6    if (pid = 4){
7      receive(x, 1);
8    }
9    ...
```

System's transition function – Point-to-point communication

```
1     if (pid = 1) {
2       x := 3;
3       send(x, 4);
4     }
5     ...
6     if (pid = 4){
7       receive(x, 1);
8     }
9     ...
```

$$\top^* \cdot ([\mathrm{L}_3] \wedge \mathit{pid} = 1) \cdot \top^* \cdot ([\mathrm{L}_7] \wedge \mathit{pid} = 4) \cdot \top^* /$$

$$\mathtt{Id}^* \cdot$$

$$f(\langle l_1, \sigma_1 \rangle, \langle l_2, \sigma_2 \rangle) \mapsto \langle [\mathrm{L}_5], \sigma_1 \rangle$$

System's transition function – Point-to-point communication

```
1     if (pid = 1) {
2       x := 3;
3       send(x, 4);
4     }
5     ...
6     if (pid = 4){
7       receive(x, 1);
8     }
9     ...
```

$$\top^* \cdot ([\mathrm{L}_3] \wedge \mathit{pid} = 1) \cdot \top^* \cdot ([\mathrm{L}_7] \wedge \mathit{pid} = 4) \cdot \top^* /$$

$$\begin{array}{c} \mathtt{Id}^* \cdot \\ f(\langle l_1, \sigma_1 \rangle, \langle l_2, \sigma_2 \rangle) \mapsto \langle [\mathrm{L}_5], \sigma_1 \rangle \cdot \\ \mathtt{Id}^* \cdot \end{array}$$

System's transition function – Point-to-point communication

```
1    if (pid = 1) {
2      x := 3;
3      send(x, 4);
4    }
5    ...
6    if (pid = 4){
7      receive(x, 1);
8    }
9    ...
```

$$\top^* \cdot ([\mathrm{L}_3] \wedge pid = 1) \cdot \top^* \cdot ([\mathrm{L}_7] \wedge pid = 4) \cdot \top^* /$$

$$\mathtt{Id}^* \cdot$$

$$f(\langle l_1, \sigma_1 \rangle, \langle l_2, \sigma_2 \rangle) \mapsto \langle [\mathrm{L}_5], \sigma_1 \rangle \cdot$$

$$\mathtt{Id}^* \cdot$$

$$f(\langle l_1, \sigma_1 \rangle, \langle l_2, \sigma_2 \rangle) \mapsto \langle [\mathrm{L}_9], \sigma_2[x \leftarrow \sigma_1(x)] \rangle$$

System's transition function – Point-to-point communication

```
1    if (pid = 1) {
2      x := 3;
3      send(x, 4);
4    }
5    ...
6    if (pid = 4){
7      receive(x, 1);
8    }
9    ...
```

$$\top^* \cdot ([\mathrm{L}_3] \wedge \mathit{pid} = 1) \cdot \top^* \cdot ([\mathrm{L}_7] \wedge \mathit{pid} = 4) \cdot \top^* /$$

$$\mathtt{Id}^* \cdot$$

$$f(\langle l_1, \sigma_1 \rangle, \langle l_2, \sigma_2 \rangle) \mapsto \langle [\mathrm{L}_5], \sigma_1 \rangle \cdot$$

$$\mathtt{Id}^* \cdot$$

$$f(\langle l_1, \sigma_1 \rangle, \langle l_2, \sigma_2 \rangle) \mapsto \langle [\mathrm{L}_9], \sigma_2[x \leftarrow \sigma_1(x)] \rangle \cdot$$

$$\mathtt{Id}^*$$

Verification

Reachability set computation

$$\text{Let } \mathcal{T} : \mathcal{A} \to \mathcal{A}$$

$$\mathcal{T}^* \subseteq \mathcal{T}^\alpha = \begin{cases} A & \text{if } \mathcal{T}(A) \sqsubseteq A \\ A \triangledown (A \cup \mathcal{T}(A)) & \text{else} \end{cases}$$

Proving safety property

Let $\mathcal{B}$ be a bad configuration : $\mathcal{T}^\alpha \cap \mathcal{B} \stackrel{?}{=} \emptyset$

Application to MPI

MPI

- HPC library
- (A)Synchronous message-passing communications
- Single Program, Multiple Data (SPMD)

Formal verification tools

- ISP
- MUST

$\Rightarrow$ Deadlock detection

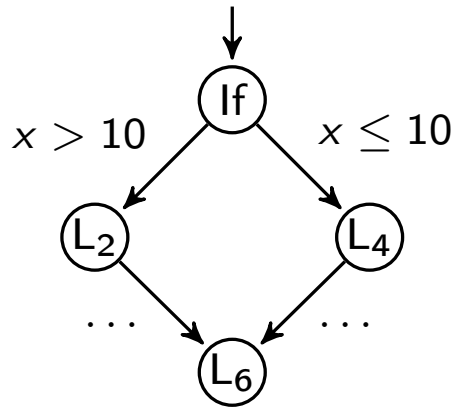
Application to MPI – Prototype overview

- Frama-C plugin
- Subset of synchronous MPI primitives
  - Point-to-point comm. : MPI_Send, MPI_Recv, ...
  - Collective comm. : MPI_Bcast, MPI_Reduce, ...
  - Dynamic creation : MPI_Create
  - ...
- Arbitrary number of processes
- Modular abstract domains (Apron library)

Application to MPI – Automatic Translation

```
1  if (x > 10){
2      ...
3  } else {
4      ...
5  }
6  ...
```

C program



CFG

$\Downarrow$

1. $[\texttt{If}] \times \{x \in [11, +\infty]\} \ / \ \mathbf{f}(\mathbf{l}, \boldsymbol{\rho}) \mapsto [\mathrm{L}_2], \rho$
2. $[\texttt{If}] \times \{x \in [-\infty, 10]\} \ / \ \mathbf{f}(\mathbf{l}, \boldsymbol{\rho}) \mapsto [\mathrm{L}_4], \rho$

Application to MPI – Automatic Translation (2)

```
1  int main(){
2     ...
3     MPI_Bcast(&x, 1, MPI_INT, root, MPI_COMM_WORLD);
4     ...
```

Application to MPI – Automatic Translation (2)

```
1  int main(){
2     ...
3     MPI_Bcast(&x, 1, MPI_INT, root, MPI_COMM_WORLD);
4     ...
```

$$[MPI\_Bcast]^* \cdot ([MPI\_Bcast] \times id \in [\![root]\!]^{\#}) \cdot [MPI\_Bcast]^* \ /$$

Application to MPI – Automatic Translation (2)

```
1  int main(){
2    ...
3    MPI_Bcast(&x, 1, MPI_INT, root, MPI_COMM_WORLD);
4    ...
```

$$[MPI\_Bcast]^* \cdot ([MPI\_Bcast] \times id \in [\![root]\!]^{\#}) \cdot [MPI\_Bcast]^* \;/$$
$$F^*(\langle l_{self}, \sigma_{self}\rangle, \langle l_{root}, \sigma_{root}\rangle) \mapsto \langle [\mathtt{L}_4], \sigma_{self}[x \leftarrow \sigma_{root}(x)]\rangle \cdot$$

Application to MPI – Automatic Translation (2)

```
1  int main(){
2     ...
3     MPI_Bcast(&x, 1, MPI_INT, root, MPI_COMM_WORLD);
4     ...
```

$$[MPI\_Bcast]^* \cdot ([MPI\_Bcast] \times id \in [\![root]\!]^\#) \cdot [MPI\_Bcast]^* \ /$$

$$F^*(\langle l_{self}, \sigma_{self}\rangle, \langle l_{root}, \sigma_{root}\rangle) \mapsto \langle [\mathrm{L}_4], \sigma_{self}[x \leftarrow \sigma_{root}(x)]\rangle \cdot$$

$$f(\langle l_{root}, \sigma_{root}\rangle) \mapsto \langle [\mathrm{L}_4], \sigma_{root}\rangle \cdot$$

Application to MPI – Automatic Translation (2)

```
1  int main(){
2     ...
3     MPI_Bcast(&x, 1, MPI_INT, root, MPI_COMM_WORLD);
4     ...
```

$$[MPI\_Bcast]^* \cdot ([MPI\_Bcast] \times id \in [\![root]\!]^\#) \cdot [MPI\_Bcast]^* \;/$$

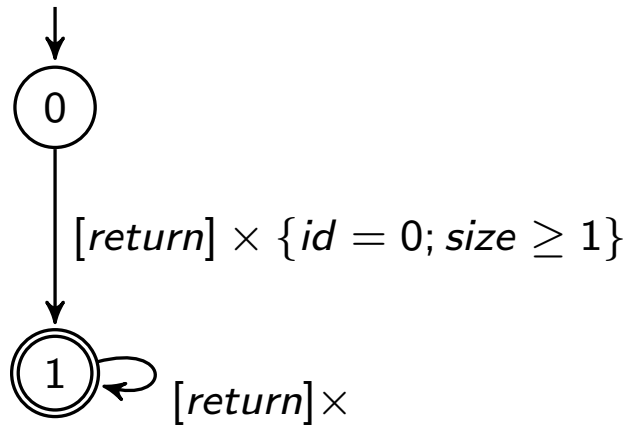
$$F^*(\langle l_{self}, \sigma_{self}\rangle, \langle l_{root}, \sigma_{root}\rangle) \mapsto \langle [\mathrm{L}_4], \sigma_{self}[x \leftarrow \sigma_{root}(x)]\rangle \cdot$$

$$f(\langle l_{root}, \sigma_{root}\rangle) \mapsto \langle [\mathrm{L}_4], \sigma_{root}\rangle \cdot$$

$$F^*(\langle l_{self}, \sigma_{self}\rangle, \langle l_{root}, \sigma_{root}\rangle) \mapsto \langle [\mathrm{L}_4], \sigma_{self}[x \leftarrow \sigma_{root}(x)]\rangle$$

Relational numerical invariants

```
 1  #include <mpi.h>
 2
 3  int main(int argc, char **argv) {
 4
 5    MPI_Init(&argc, &argv);
 6
 7    int id, i, size, x;
 8
 9    MPI_Comm_rank(MPI_COMM_WORLD, &id);
10    MPI_Comm_size(MPI_COMM_WORLD, &size);
11
12    if (id < 1){
13      for (i = 1; i < size; i++){
14        x = i * 4;
15        MPI_Send(&x, 1, MPI_INT,
16                 i, 0, MPI_COMM_WORLD);
17      }
18    } else {
19      MPI_Recv(&x, 1, MPI_INT,
20               0, 0, MPI_COMM_WORLD, NULL);
21    }
22
23    MPI_Finalize();
24    return 0;
25  }
```



Resulting invariant

Results

Working prototype

- Output : Regular numerical invariant
- Supports dynamic process creation/destruction
- Fully automated analysis

Perspectives

- Integration of EVA plugin
- Include more primitives
- Application algorithm optimisations
- Handle asynchronous communications
- Explore different concurrency models (BSP, POSIX threads, . . .)

Perspectives

- Integration of EVA plugin
- Include more primitives
- Application algorithm optimisations
- Handle asynchronous communications
- Explore different concurrency models (BSP, POSIX threads, . . .)

Thanks