Static analysis and software verification

Lecture 3 - Modeling programs

Vincenzo Arceri - University of Parma - vincenzo.arceri@unipr.it

```
i = read();
if (i != 0)
  j = 5 / i;
else
  j = 0;
return;
```

```
i = read();
if (i != 0)
  j = 5 / i;
else
  j = 0;
return;
```

Goal: *certify* that the program is safe w.r.t. division by zero

Posets in static analysis

- The idea is to compute the possible values of variable
  - For each program point
  - For all possible executions
- $\langle \wp(\mathbb{Z}), \subseteq \rangle$

```
1 i = read();
2 if (i != 0)
3   j = 5 / i;
4 else
5   j = 0;
6 return;
```

Program point	Values
1	i = {}
2	
3	
5	
6	

Posets in static analysis

- The idea is to compute the possible values of variable
  - For each program point
  - For all possible executions
- $\langle \wp(\mathbb{Z}), \subseteq \rangle$

```
1 i = read();
2 if (i != 0)
3   j = 5 / i;
4 else
5   j = 0;
6 return;
```

Program point	**Values**
1	i = {}
2	i = {…, -1, 0, 1,…}
3	
5	
6	

Posets in static analysis

- The idea is to compute the possible values of variable
  - For each program point
  - For all possible executions
- $\langle \wp(\mathbb{Z}), \subseteq \rangle$

```
1 i = read();
2 if (i != 0)
3   j = 5 / i;
4 else
5   j = 0;
6 return;
```

Program point	Values
1	i = {}
2	i = {…, -1, 0, 1,…}
3	i = {…, -1, 1,…}
5	i = {0}
6	

Posets in static analysis

- The idea is to compute the possible values of variable
  - For each program point
  - For all possible executions
- $\langle \wp(\mathbb{Z}), \subseteq \rangle$
- How to compute the possible values at program point 6?
  - output pp 3: j = {…, -1, 1,…}
  - output pp 5: j = {0}

```
1 i = read();
2 if (i != 0)
3   j = 5 / i;
4 else
5   j = 0;
6 return;
```

Program point	Values
1	i = {}
2	i = {…, -1, 0, 1,…}
3	i = {…, -1, 1,…}
5	i = {0}
6	

Posets in static analysis

- The idea is to compute the possible values of variable
  - For each program point
  - For all possible executions
- $\langle \wp(\mathbb{Z}), \subseteq, \cup, \cap \rangle$ is a lattice
- How to compute the possible values at program point 6?
  - output pp 3: j = {…, -1, 1,…}
  - output pp 5: j = {0}

```
1 i = read();
2 if (i != 0)
3   j = 5 / i;
4 else
5   j = 0;
6 return;
```

Program point	**Values**
1	i = {}
2	i = {…, -1, 0, 1,…}
3	i = {…, -1, 1,…}
5	i = {0}
6	

Posets in static analysis

- The idea is to compute the possible values of variable
  - For each program point
  - For all possible executions
- $\langle \wp(\mathbb{Z}), \subseteq, \cup, \cap \rangle$ is a lattice
- How to compute the possible values at program point 6?
  - output pp 3: $j = \{\ldots, -1, 1, \ldots\}$
  - output pp 5: $j = \{0\}$

```
1 i = read();
2 if (i != 0)
3   j = 5 / i;
4 else
5   j = 0;
6 return;
```

Program point	Values
1	$i = \{\}$
2	$i = \{\ldots, -1, 0, 1, \ldots\}$
3	$i = \{\ldots, -1, 1, \ldots\}$
5	$i = \{0\}$
6	

Posets in static analysis

- The idea is to compute the possible values of variable
  - For each program point
  - For all possible executions
- $\langle \wp(\mathbb{Z}), \subseteq, \cup, \cap \rangle$ is a lattice
- How to compute the possible values at program point 6?
  - output pp 3: j = {-5, -2, -1, 0, 1, 2, 5}
  - output pp 5: j = {0}

```
1 i = read();
2 if (i != 0)
3   j = 5 / i;
4 else
5   j = 0;
6 return;
```

Program point	Values
1	i = {}
2	i = {…, -1, 0, 1,…}
3	i = {…, -1, 1,…}
5	i = {0}
6	i = {…, -1, 0, 1, …} j = {-5, -2, -1, 0, 1, 2, 5}

Least upper bound in static analysis

```
1 i = {-1, 0, 1}
2 if (i != 0)
3   j = 5 / i;
4 else
5   j = 0;
6 return;
```

Program point	Values
1	i = {}
2	i = [-1, 1]
3	i = [0, 1]
5	i = [0, 0]
6	

- from 3: j = [-5, 5]
- from 5: j = [0, 0]

Least upper bound in static analysis

```
1 i = {-1, 0, 1}
2 if (i != 0)
3   j = 5 / i;
4 else
5   j = 0;
6 return;
```

Program point	Values
1	i = {}
2	i = [-1, 1]
3	i = [0, 1]
5	i = [0, 0]
6	i = [-1, -1] j = [-5, 5]

- from 3: j = [-5, 5]
- from 5: j = [0, 0]

Least upper bound in static analysis

```
1 i = {-1, 0, 1}
2 if (i != 0)
3   j = 5 / i;
4 else
5   j = 0;
6 return;
```

Program point	Values
1	i = {}
2	i = [-1, 1]
3	i = [0, 1]
5	i = [0, 0]
6	i = [-1, 1] j = [-10, 5]

- from 3: j = [-5, 5]
- from 5: j = [0, 0]

Least upper bound in static analysis

```
1 i = {-1, 0, 1}
2 if (i != 0)
3   j = 5 / i;
4 else
5   j = 0;
6 return;
```

Program point	Values
1	i = {}
2	i = [-1, 1]
3	i = [0, 1]
5	i = [0, 0]
6	i = [-1, 1] j = [-5, 5]

- from 3: j = [-5, 5]
- from 5: j = [0, 0]

The ‘abstract state’ is modeled as a function

Posets in static analysis

- The idea is to compute the possible values of variable
  - For each program point
  - For all possible executions
- $\langle \wp(\mathbb{Z}), \subseteq, \cup, \cap \rangle$ is a lattice
- How to compute the possible values at program points 3 and 5?

```
1 i = read();
2 if (i != 0)
3   j = 5 / i;
4 else
5   j = 0;
6 return;
```

Program point	Values
1	i = {}
2	i = {…, -1, 0, 1,…}
3	i = {…, -1, 1,…}
5	i = {0}
6	i = {…, -1, 0, 1, …} j = {…, -1, 0, 1, …}

Posets in static analysis

- The idea is to compute the possible values of variable
  - For each program point
  - For all possible executions
- $\langle \wp(\mathbb{Z}), \subseteq, \cup, \cap \rangle$ is a lattice
- How to compute the possible values at program points 3 and 5?
- filter(i != 0) = {…, -1, 1, …}
- 3 = 2 $\cap$ filter(x != 0) = {…, -1, 1, …}

```
1 i = read();
2 if (i != 0)
3   j = 5 / i;
4 else
5   j = 0;
6 return;
```

Program point	Values
1	i = {}
2	i = {…, -1, 0, 1,…}
3	i = {…, -1, 1,…}
5	i = {0}
6	i = {…, -1, 0, 1, …} j = {-5, -2, -1, 0, 1, 2, 5}

Fixpoints

- $\langle \wp(\mathbb{Z}), \subseteq, \cup, \cap \rangle$ is a lattice

```
1 i = 0;
2 while (?)
3   i++;
4 ...
```

1	empty set
2	{0} u [[i++]]**3**
3	**2**
4	**2**

Fixpoints

- $\langle \wp(\mathbb{Z}), \subseteq, \cup, \cap \rangle$ is a lattice

```
1 i = 0;
2 while (?)
3   i++;
4 ...
```

1	empty set
2	{0} u [[i++]]**3**
3	**2**
4	**2**

	0	**1**	**2**	**3**	**4**
1	empty set				
2	empty set				
3	empty set				
4	empty set				

Fixpoints

- $\langle \wp(\mathbb{Z}), \subseteq, \cup, \cap \rangle$ is a lattice

```
1 i = 0;
2 while (?)
3   i++;
4 ...
```

1	empty set
2	{0} u [[i++]]**3**
3	**2**
4	**2**

	0	**1**	**2**	**3**	**4**
1	empty set	empty set			
2	empty set	{0}			
3	empty set	empty set			
4	empty set	empty set			

Fixpoints

- $\langle \wp(\mathbb{Z}), \subseteq, \cup, \cap \rangle$ is a lattice

```
i = 0;
while (?)
   i++;
...
```

1	empty set
2	{0} u [[i++]]**3**
3	**2**
4	**2**

	0	**1**	**2**	**3**	**4**
1	empty set	empty set	empty set		
2	empty set	{0}	{0}		
3	empty set	empty set	{0}		
4	empty set	empty set	{0}		

Fixpoints

- $\langle \wp(\mathbb{Z}), \subseteq , \cup , \cap \rangle$ is a lattice

```
1 i = 0;
2 while (?)
3   i++;
4 ...
```

1	empty set
2	{0} u [[i++]]**3**
3	**2**
4	**2**

	0	**1**	**2**	**3**	**4**
1	empty set	empty set	empty set	empty set	
2	empty set	{0}	{0}	{0, 1}	
3	empty set	empty set	{0}	{0}	
4	empty set	empty set	{0}	{0}	

Fixpoints

- $\langle \wp(\mathbb{Z}), \subseteq, \cup, \cap \rangle$ is a lattice

```
1 i = 0;
2 while (?)
3   i++;
4 ...
```

1	empty set
2	{0} u [[i++]]**3**
3	**2**
4	**2**

	0	**1**	**2**	**3**	**4**
1	empty set	empty set	empty set	empty set	empty set
2	empty set	{0}	{0}	{0, 1}	{0, 1}
3	empty set	empty set	{0}	{0}	{0, 1}
4	empty set	empty set	{0}	{0}	{0, 1}

Fixpoints

- $\langle \wp(\mathbb{Z}), \subseteq, \cup, \cap \rangle$ is a lattice

```
1 i = 0;
2 while (?)
3     i++;
4 ...
```

1	empty set
2	{0} u [[i++]]**3**
3	**2**
4	**2**

	0	**1**	**2**	**3**	**4**
1	empty set	empty set	empty set	empty set	empty set
2	empty set	{0}	{0}	{0, 1}	{0, 1}
3	empty set	empty set	{0}	{0}	{0, 1}
4	empty set	empty set	{0}	{0}	{0, 1}

By Kleene theorem, the fixpoint exists, but we cannot compute it in this way…

Fixpoints

- $\langle \{0, +, -, \bot, \top\}, \sqsubseteq, \sqcup, \sqcap, \top, \bot \rangle$ is a lattice

```
1 i = 0;
2 while (?)
3   i++;
4 ...
```

1	bottom
2	0 u [[i++]]**3**
3	**2**
4	**2**

	0	**1**	**2**	**3**	**4**
1	bottom				
2	bottom				
3	bottom				
4	bottom				

Fixpoints

- $\langle \{0, +, -, \bot, \top\}, \sqsubseteq, \sqcup, \sqcap, \top, \bot \rangle$ is a lattice

```
1 i = 0;
2 while (?)
3   i++;
4 ...
```

1	bottom
2	0 u [[i++]]**3**
3	**2**
4	**2**

	0	**1**	**2**	**3**	**4**
1	bottom	bottom	bottom		
2	bottom	0	T		
3	bottom	+	T		
4	bottom	0	T		

Fixpoints

- $\langle \{0, +, -, \bot, \top\}, \sqsubseteq, \sqcup, \sqcap, \top, \bot \rangle$ is a lattice

```
1 i = 0;
2 while (?)
3   i++;
4 ...
```

1	bottom
2	0 u [[i++]]**3**
3	**2**
4	**2**

	0	**1**	**4**	**5**	**6**
1	bottom	…	**bottom**	**bottom**	
2	bottom	…	**T**	**T**	
3	bottom	…	**T**	**T**	
4	bottom	…	**T**	**T**	

Fixpoint!

IMP

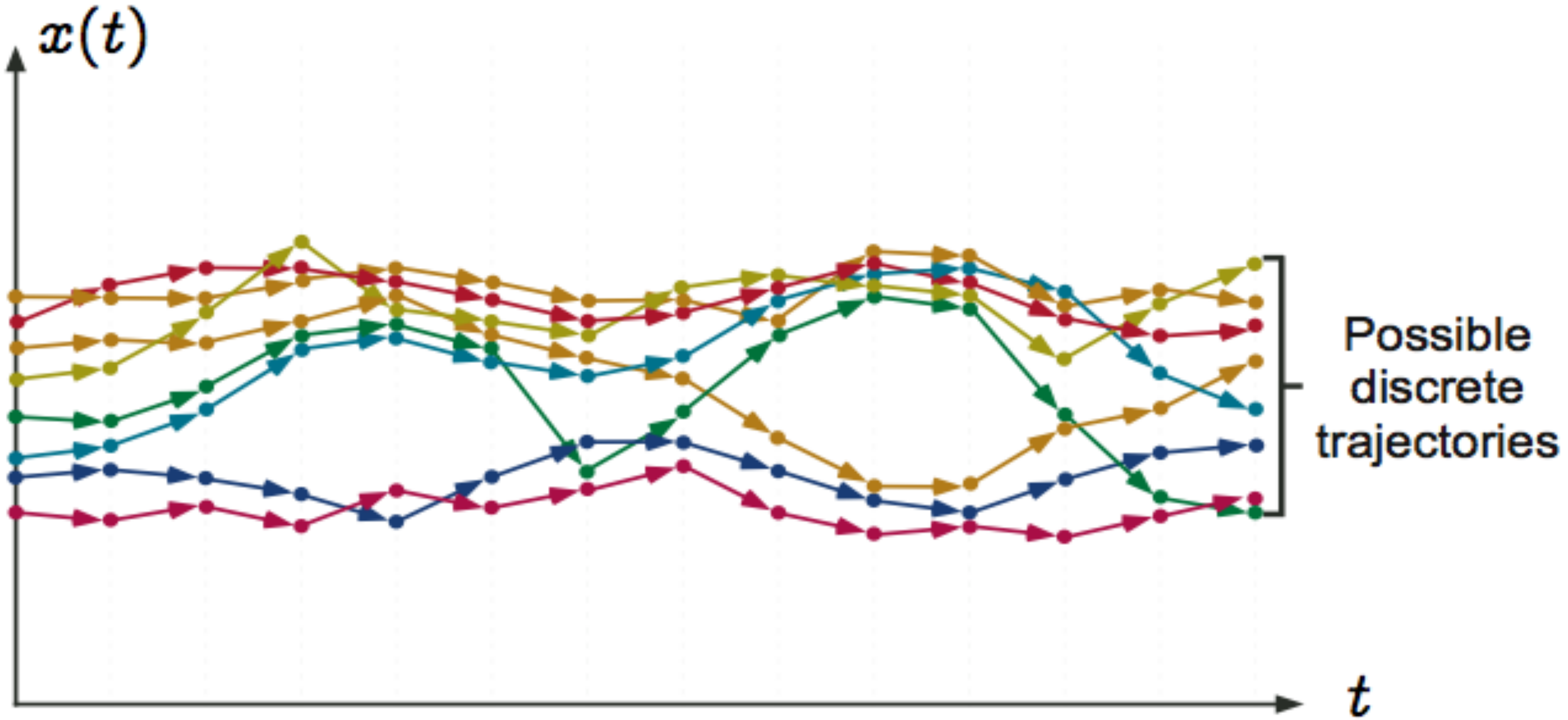
$$
\begin{aligned}
e &::= \quad x \mid n \mid e_1 \ \mathrm{op}_a \ e_2 \\
b &::= \quad \mathbf{true} \mid \mathbf{false} \mid \neg b_1 \mid b_1 \ \mathrm{op}_b \ b_2 \mid e_1 \ \mathrm{op}_c \ e_2 \\
s &::= \quad x \coloneqq e; \mid \mathbf{skip} \mid s_1 \ s_2 \mid \mathbf{if} \ b \ \mathbf{then} \ s_1 \ \mathbf{else} \ s_2 \mid \mathbf{while} \ b \ \mathbf{do} \ s_1
\end{aligned}
$$

Where:

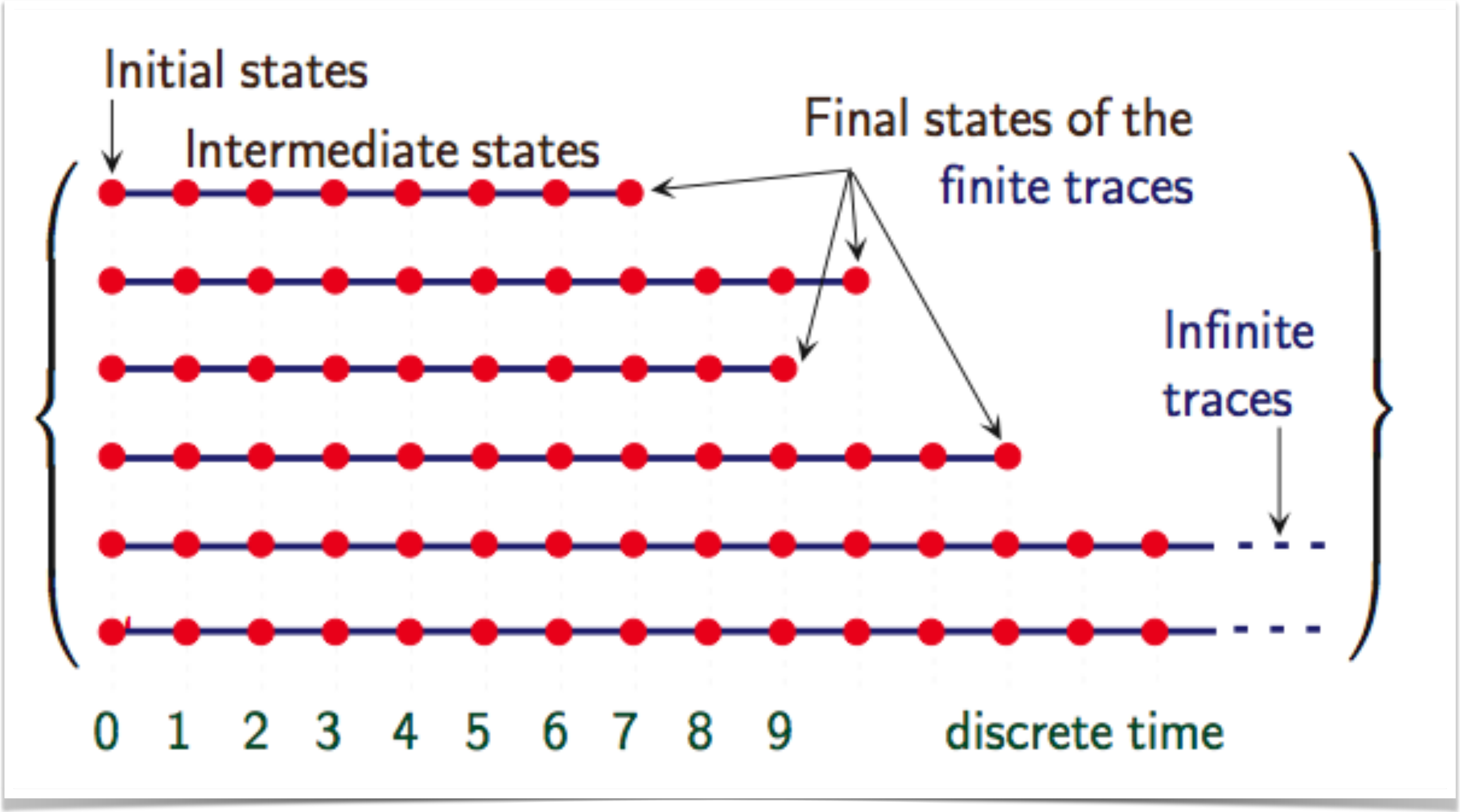
- $n \in \mathbb{Z}$
- $\mathrm{op}_a \in \{+,\ -,\ *,\ \div\}$
- $\mathrm{op}_b \in \{\&\&,\ \|\}$
- $\mathrm{op}_c \in \{==,\ >,\ <,\ \geq,\ \leq\}$

IMP concrete semantics

Trace semantics



Trace semantics



Trace semantics

- We need a lattice structure for the trace semantics
- A single execution of a program is $\tau \in X^\infty$
- We can model the concrete semantics as the lattice $\langle \wp(X^\infty), \subseteq, \cup, \cap \rangle$
- How to define the semantics (intuition)
    - Compute a single-step of the computation, obtaining a set of *partial executions*
    - Iterate over them to compute the following steps
    - Until a fixpoint is reached

Final partial execution

- First step
  - Execution of length 1
  - Union with the traces with only the initial states
  - i-th step
    - Execution of length i
    - take the unionwith the traces long at most i-1
  - Until a fixpoint is reached

Least fixpoint computation

- Given a program P, the fixpoint semantics, starting from *below* (least fixpoint)

$$\mathsf{lfp}_{\emptyset}^{\subseteq} F_P$$

- In this way, we obtain the set of partial executions

Example

Trace semantics

- The trace semantics defined so far ‘computes’ all partial executions, including
  - blocking execution
  - infinite execution
  - partial executions
- Based on the purpose, we can abstract it
  - only blocking execution
  - values for each program point

An abstraction of the trace semantics