

# Static Analysis of String Values

Giulia Costantini<sup>1</sup>, Pietro Ferrara<sup>2</sup>, Agostino Cortesi<sup>1</sup>

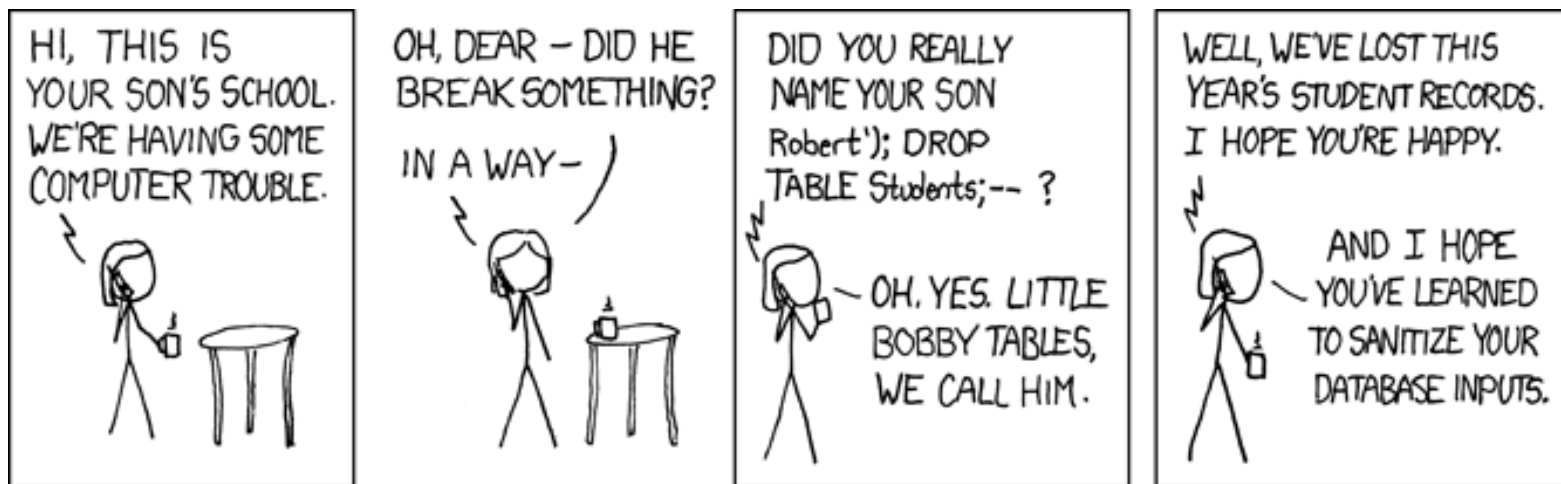
<sup>1</sup> Università Ca' Foscari  
Venice, Italy

<sup>2</sup> ETH Zurich  
Switzerland

ICFEM 2011, Durham, England

# Strings

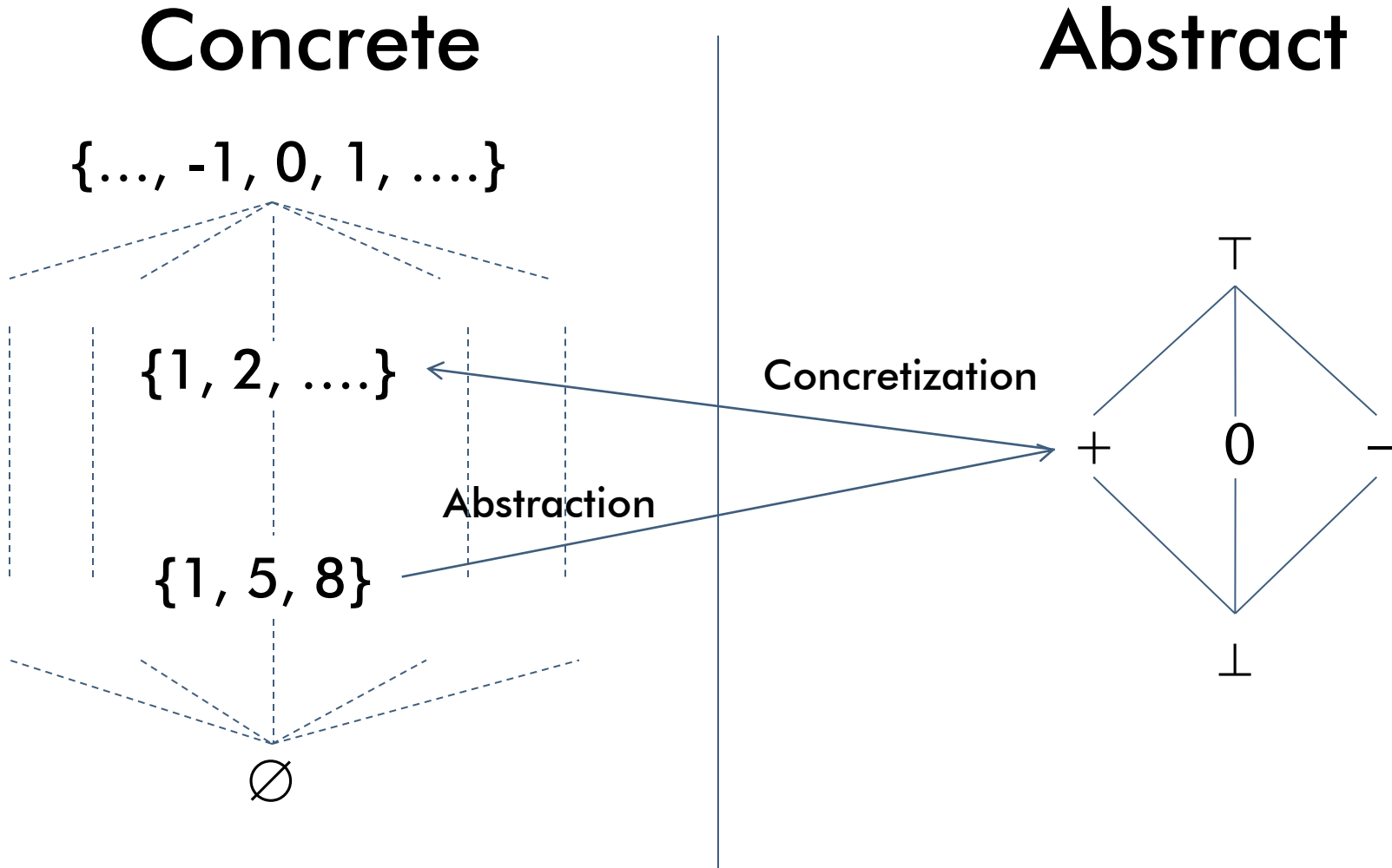
- Strings are everywhere:
  - > SQL queries
  - > Reflection
- Wrong use could have catastrophic effects



# Sound static analysis

- **Prove properties**
  - > at compile time (static)
  - > respected by all executions (sound)
- **Abstract interpretation**
  - > Cousot&Cousot 77/79
  - > Mathematical framework to
    - Define the semantics
    - Soundly approximate it
  - > Ideal goal: fast and precise abstraction

# Bases of abstract interpretation



# Semantics

## Concrete

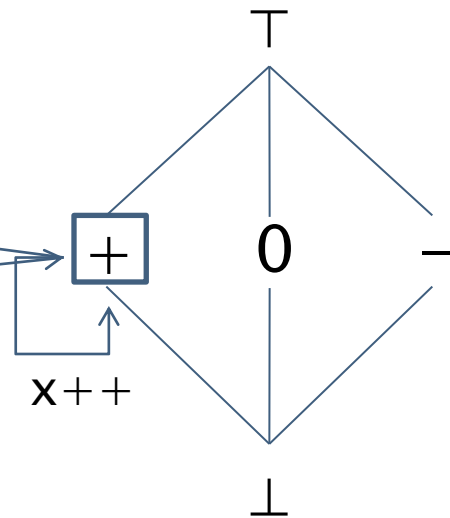
$\{\dots, -1, 0, 1, \dots\}$

$\{1, 2, \dots\}$

$\{1, 5, 8\} \xrightarrow{x++} \{2, 6, 9\}$

$\emptyset$

## Abstract



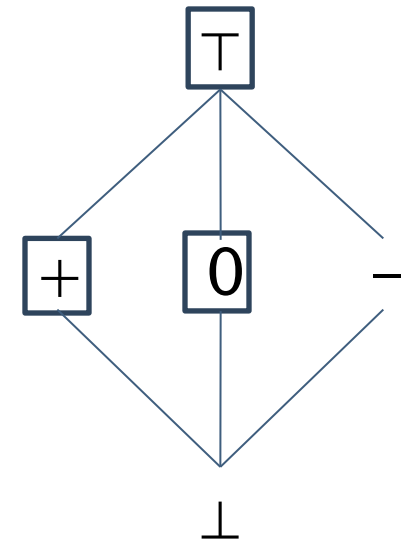
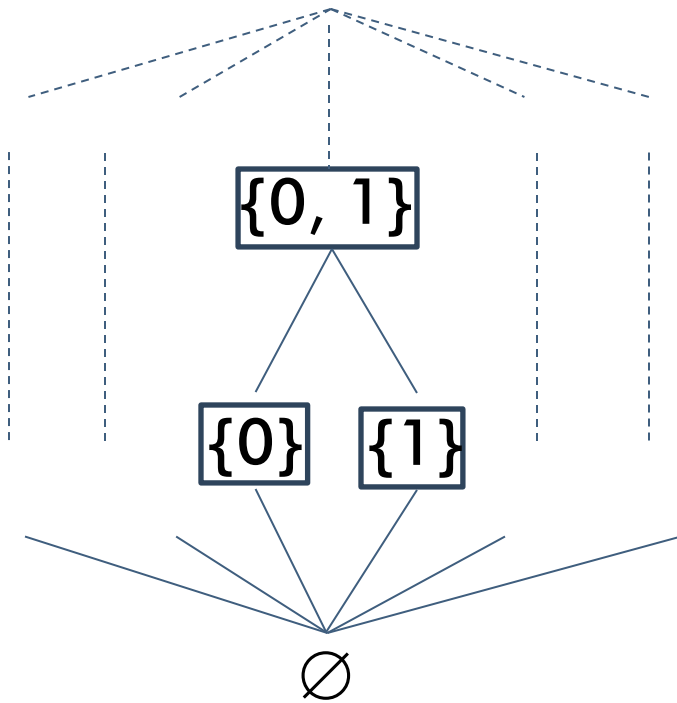
# Upper bound

Concrete

$\{\dots, -1, 0, 1, \dots\}$

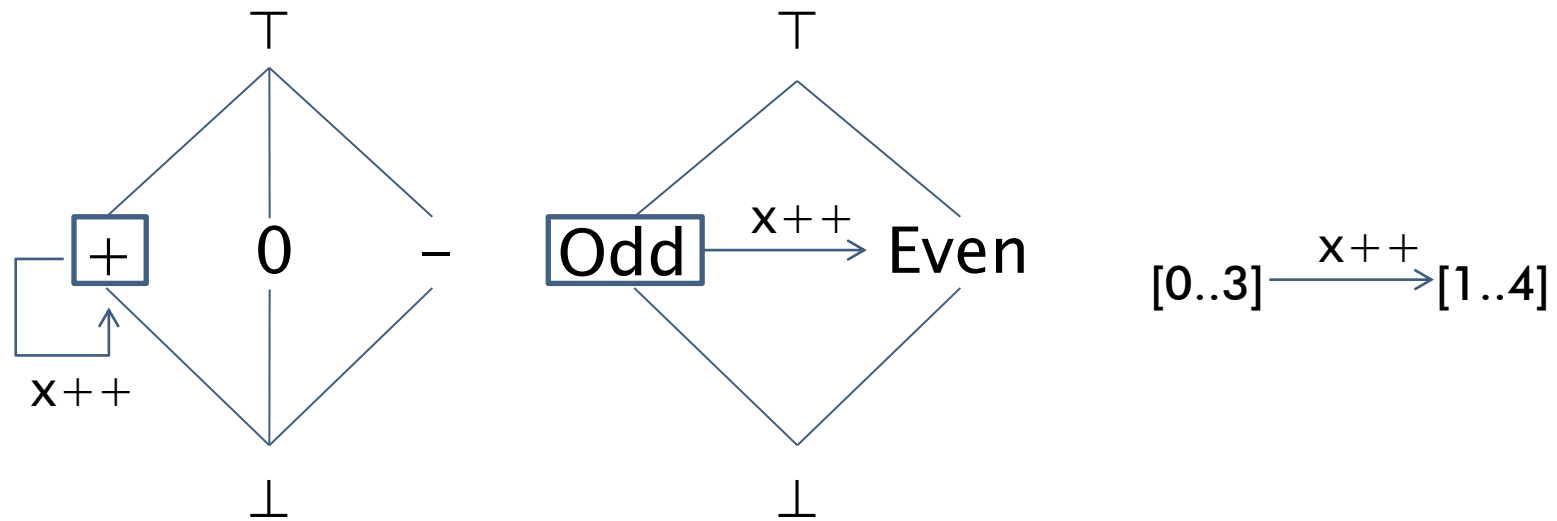
→ if(...)  
x=0;  
else x=1;

Abstract



# Numerical analyses

- Common interface for several analyses



- Semantics of  $+$ ,  $-$ ,  $*$ ,  $/$ , constants, ...

# Outline

1. Introduction
2. **Generic framework for string analysis**
3. **String domains**
  - a) Character inclusion
  - b) Prefix and suffix
  - c) Bricks
  - d) String graphs
4. **Conclusion**

# String operators

- **Set of standard operators on strings:**
  - > `new String("str")`
    - or `"str"`
  - > `concat(s1, s2)`
    - or `s1 + s2`
  - > `readLine()`
  - > `substring(b, e, s)`
  - > `contains(c, s)`
- **Each domain has a lattice structure**

# Running example

```
string x = "a";
```

```
while (...)
```

```
    x = "0" + x + "1";
```

```
return x;
```

Because of  
approximation/user input/...

$x = \{"0a1", "00a11", \dots\} =$   
 $= "0"^n + "a" + "1"^n$   
with  $n > 0$

$x = \{"a", "0a1", "00a11", \dots\} =$   
 $= "0"^n + "a" + "1"^n$  with  $n \geq 0$

# Outline

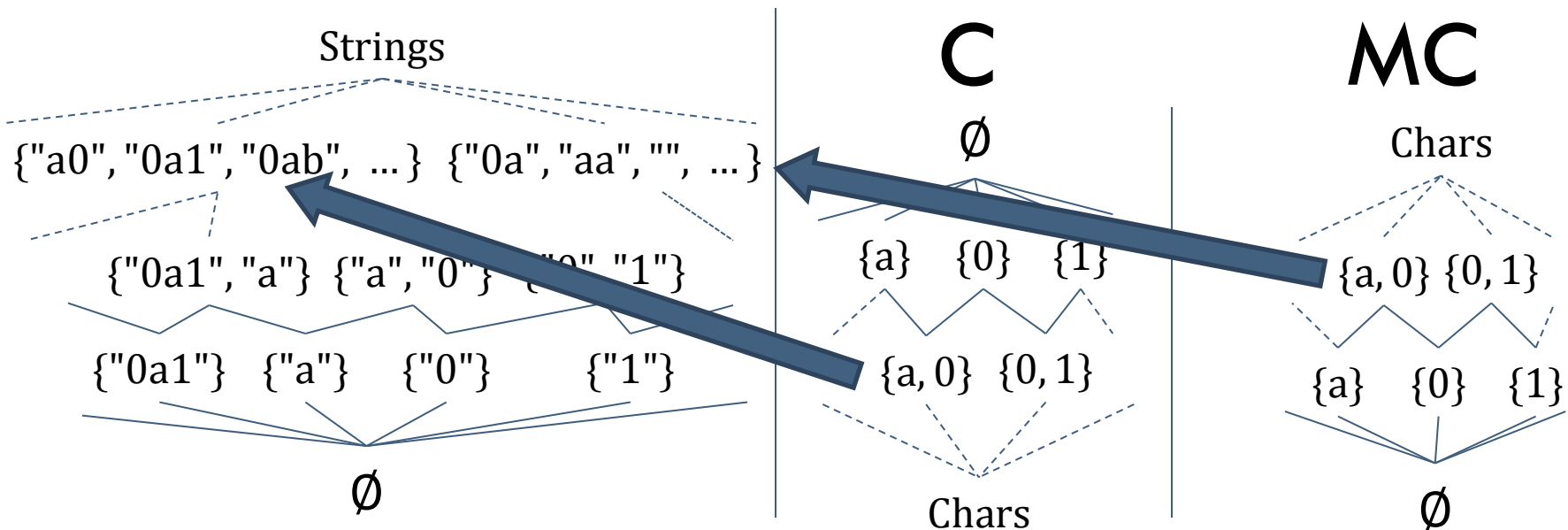
1. Introduction
2. Generic framework for string analysis
- 3. String domains**
  - a) Character inclusion
  - b) Prefix and suffix
  - c) Bricks
  - d) String graphs
- 4. Conclusion**

# Character inclusion

- Strings approximated through
  - > C: characters surely contained
  - > MC: characters possibly contained

Concrete

Abstract



# Character inclusion – Running example

```
string x = "a";
```

$\mathbb{S}_{CZ}[\text{new String}(\text{str})]() = (\text{char}(\text{str}), \text{char}(\text{str}))$

$C : \{a\}$

$MC : \{a\}$

```
while (...)
```

```
    x = "0" + x + "1";
```

$\overline{\mathbb{S}_{CZ}}[\text{concat}]((\overline{C}_1, \overline{MC}_1), (\overline{C}_2, \overline{MC}_2)) = (\overline{C}_1 \cup \overline{C}_2, \overline{MC}_1 \cup \overline{MC}_2)$

$C : \{a, 0, 1\}$

$MC : \{a, 0, 1\}$

```
return x;
```

$C : \{a\} \sqcup \{a, 0, 1\} = \{a\} \cap \{a, 0, 1\} = \{a\}$   
 $MC : \{a\} \sqcup \{a, 0, 1\} = \{a\} \cup \{a, 0, 1\} = \{a, 0, 1\}$

Concretization

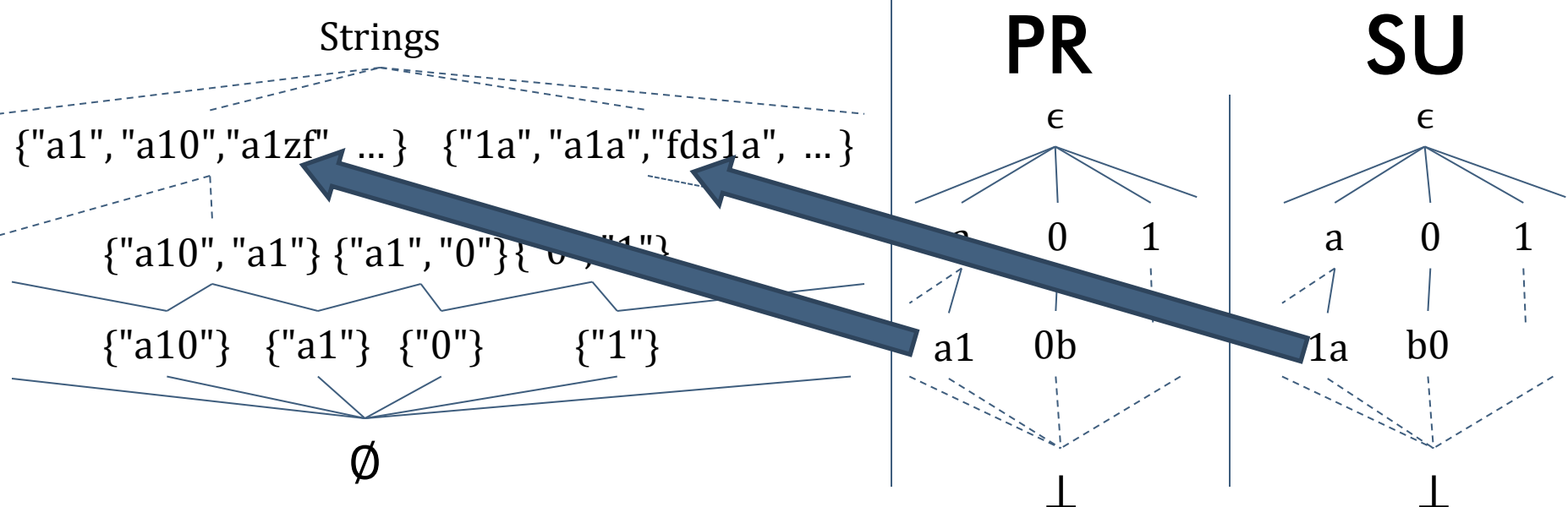
$\{"a", "a0", "000a1", "0101a010", \dots\}$

# Prefix & Suffix

- Strings approximated through
  - > PR: prefix of the string
  - > SU: suffix of the string

Concrete

Abstract



# Prefix & Suffix – Running example

```
string x = "a";
```

$\mathbb{S}_{PR}[\text{new String}(\text{str})]() = \text{str}$

$\mathbb{S}_{SU}[\text{new String}(\text{str})]() = \text{str}$

**PR : a , SU : a**

```
while (...)
```

```
    x = "0" + x + "1";
```

$\mathbb{S}_{PR}[\text{concat}](\bar{p}_1, \bar{p}_2) = \bar{p}_1$

$\mathbb{S}_{SU}[\text{concat}](\bar{s}_1, \bar{s}_2) = \bar{s}_2$

**PR : 0, SU : 1**

```
return x;
```

**PR : a  $\sqcup$  0 =  $\epsilon$**

**SU : a  $\sqcup$  1 =  $\epsilon$**

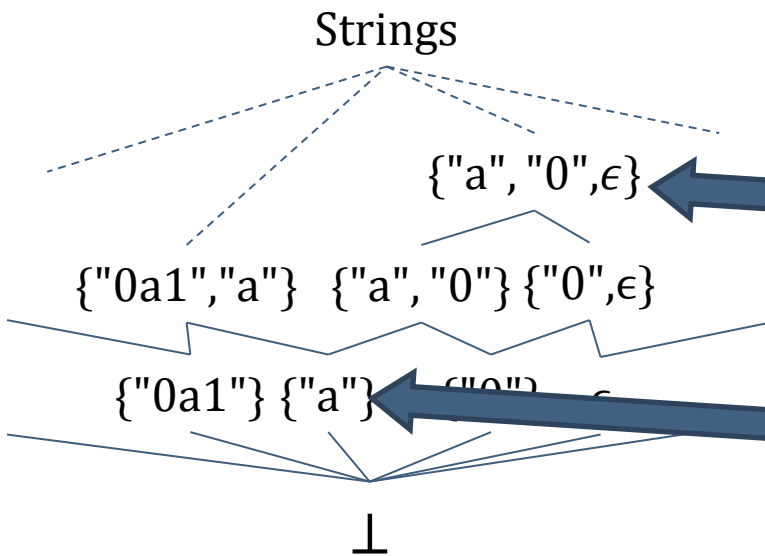
Concretization

Strings

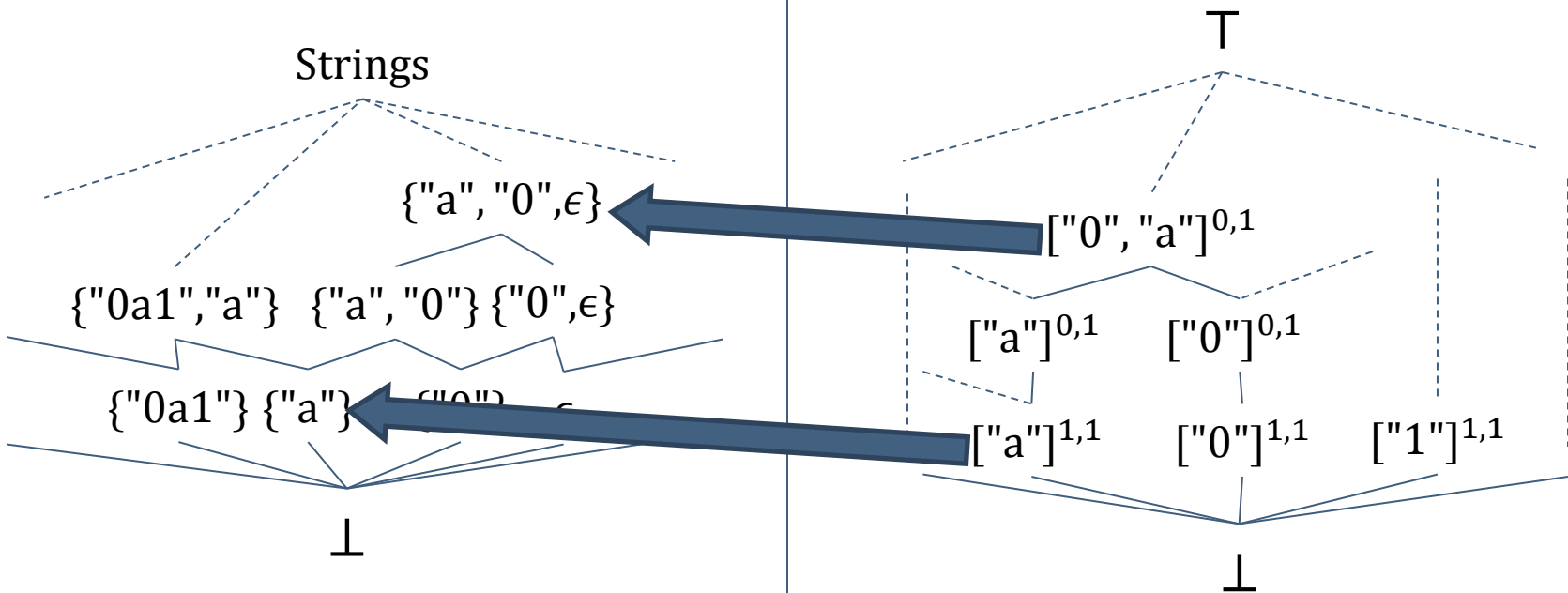
# Bricks

- Sequence of  $[\wp(\text{Strings})]^{\min, \max}$   
 $> ["a"]^{0,1} ["b", "c"]^{1,1} = \{"b", "c", "ab", "ac"\}$

## Concrete



## Abstract



# Bricks – Running example

string x = "a";

$$\mathbb{S}_{BR}[\text{new String(str)}]() = [\{\text{str}\}]^{1,1}$$

$$["a"]^{1,1}$$

while (...)

$$\overline{\mathbb{S}}_{BR}[\text{concat}](b_1, b_2) = \text{normBricks}(\text{concatList}(b_1, b_2))$$

$$\text{normBricks}(["0"]^{1,1} ["a"]^{1,1} ["1"]^{1,1}) = ["0a1"]^{1,1}$$

x = "0" + x + "1";

return x;

$$["a"]^{1,1} \sqcup ["0a1"]^{1,1} = ["a", "0a1"]^{1,1}$$

$$["a", "0a1"]^{1,1} \sqcup ["1"]^{1,1} = ["a", "0a1", "1"]^{1,1}$$

$$\dots = \top$$

**Widening!**

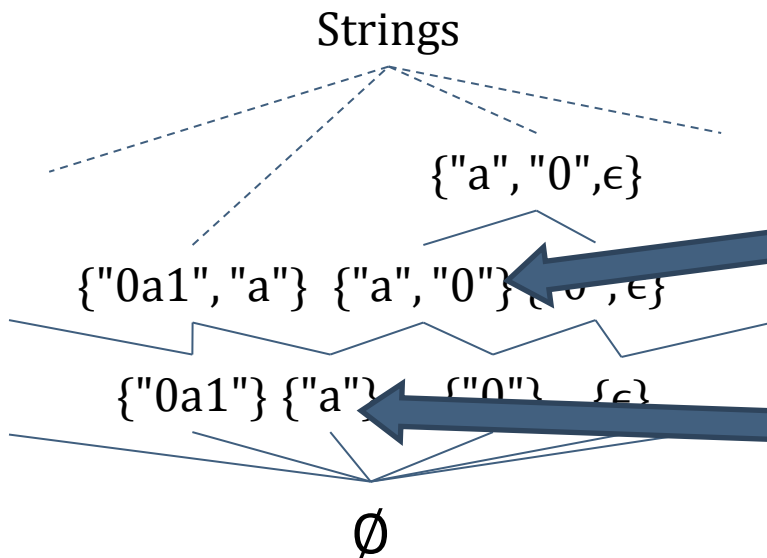
Concretization

Strings

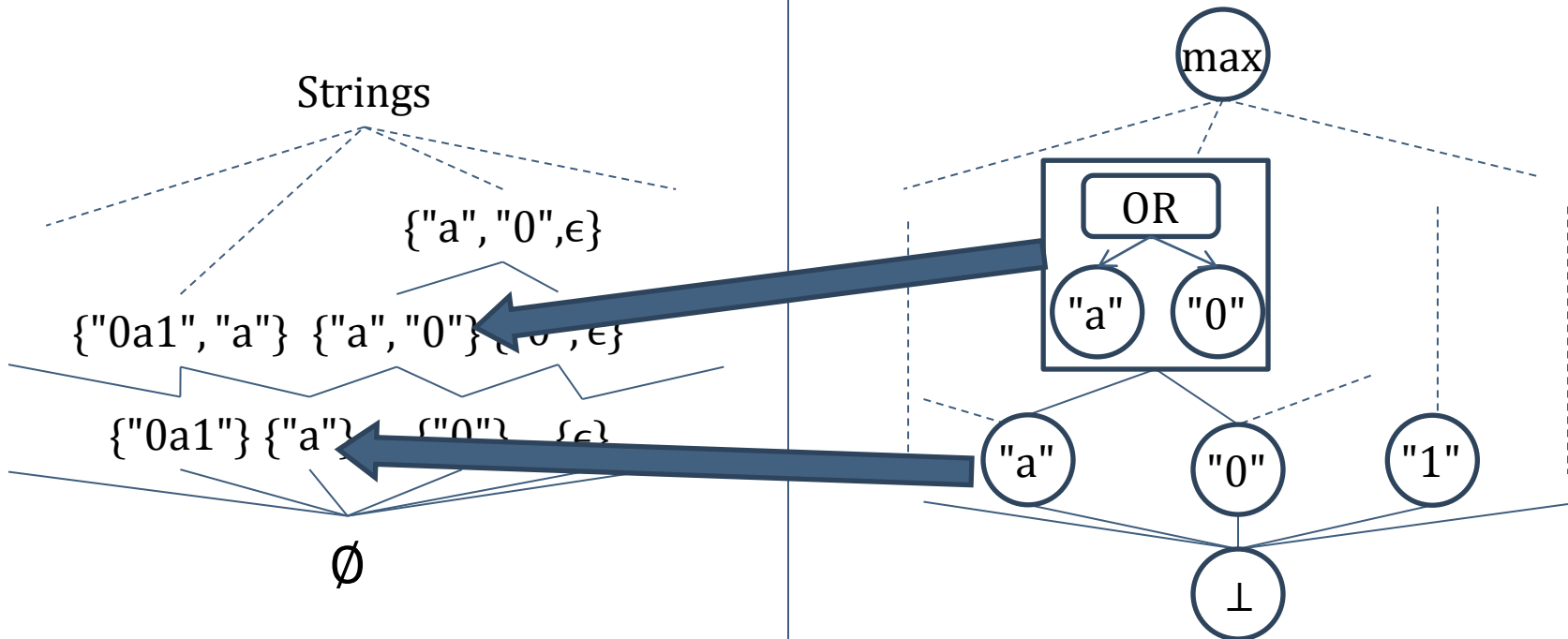
# String graphs

- Adaptation of type graphs (tree automata)
  - > Rely on their normalization and widening

## Concrete



## Abstract



# String graphs – Running example

```
string x = "a";
```

$$S_{SG}[\text{new String}(\text{str})]() = \text{str}$$

```
while (...)
```

```
    x = "0" + x + "1";
```

$$S_{SG}[\text{concat}](\bar{t}_1, \bar{t}_2) = \text{concat}$$
$$= \text{"0a1"}$$

```
return x;
```

$$\text{"a"} \sqcup \text{"0a1"} = \text{OR}$$

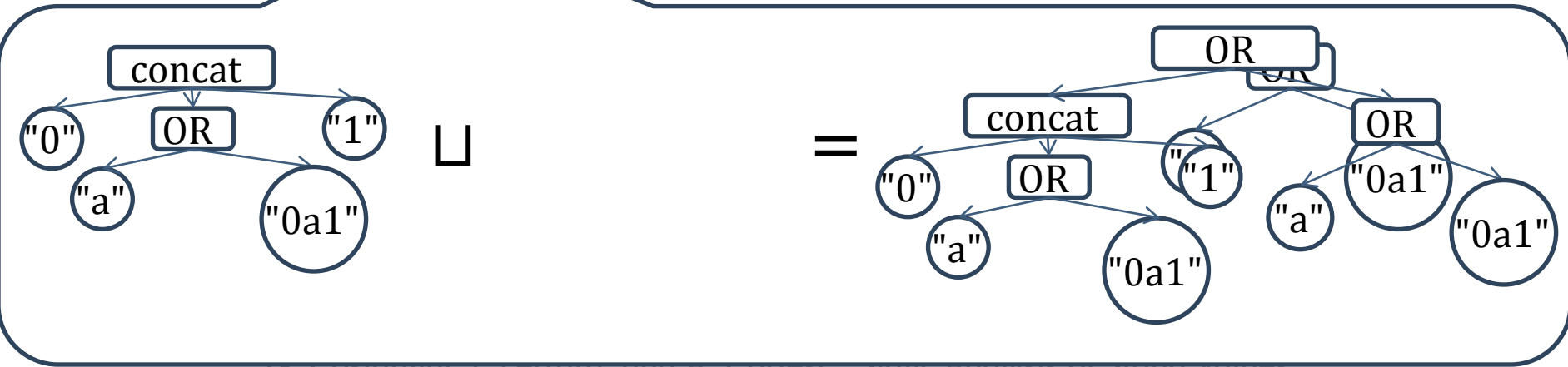
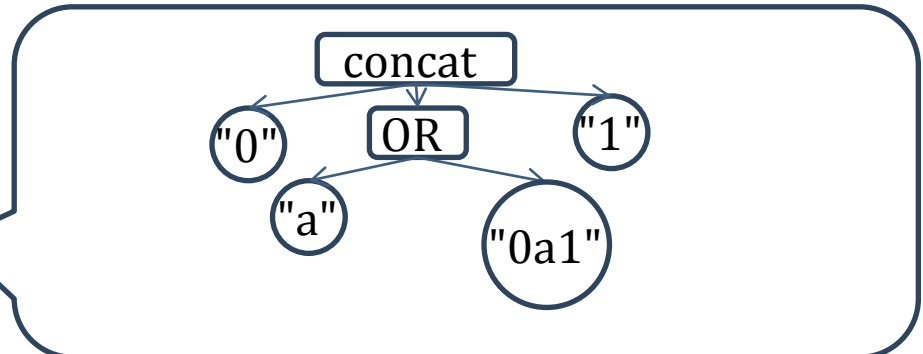
# String graphs – Running example

```
string x = "a";
```

```
while(...)
```

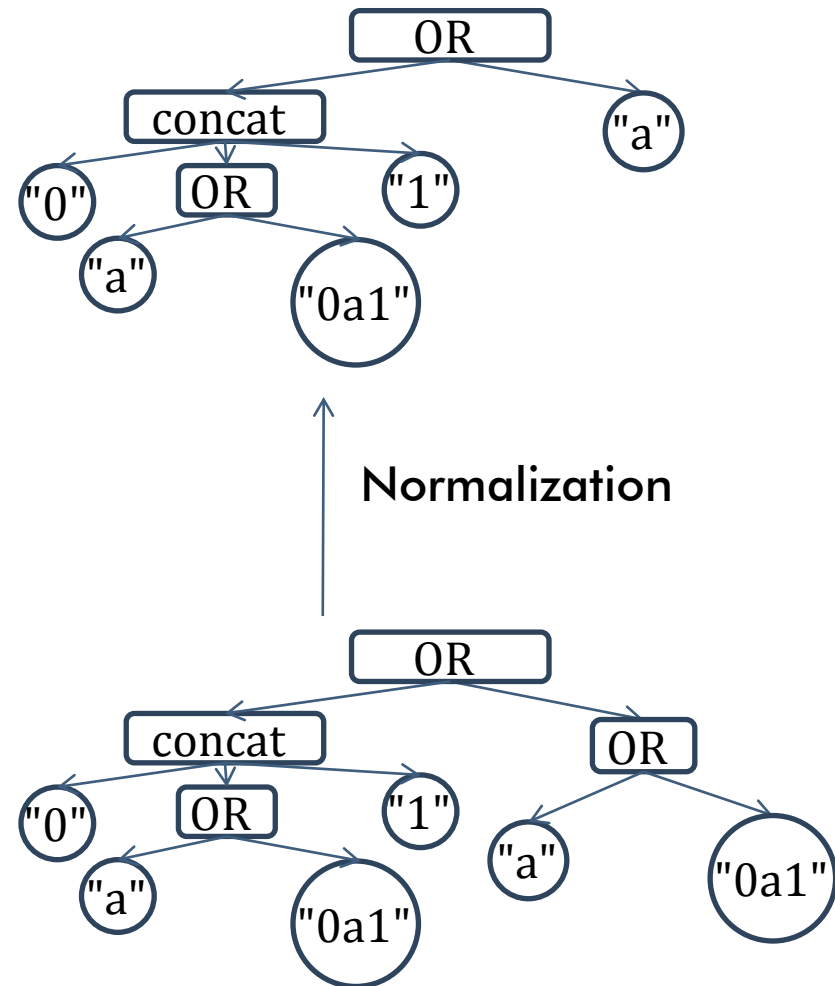
```
  x = "0" + x + "1";
```

```
return x;
```



# String graphs – Running example

```
string x = "a";  
while (...)  
    x = "0" + x + "1";  
return x;
```



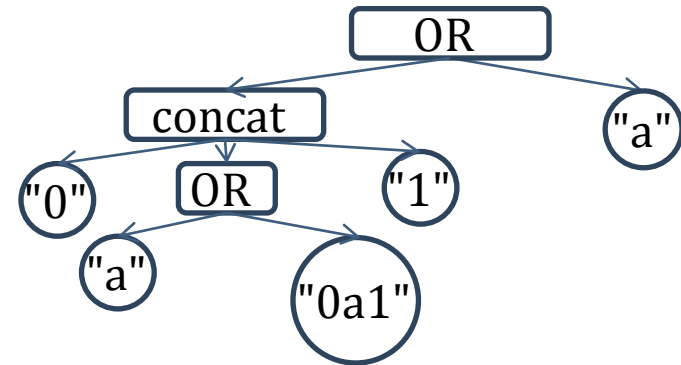
# String graphs – Running example

```
string x = "a";
```

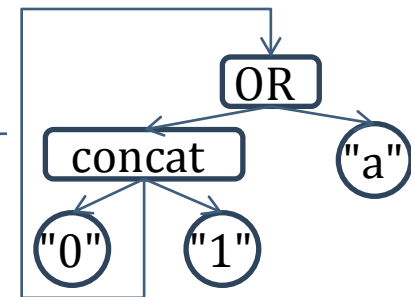
```
while (...)
```

```
    x = "0" + x + "1";
```

```
return x;
```



Widening!










"0"<sup>n</sup> + "a" + "1"<sup>n</sup> with  $n \geq 0$

Concretization

# Outline

1. Introduction
2. Generic framework for string analysis
3. String domains
  - a) Character inclusion
  - b) Prefix and suffix
  - c) Bricks
  - d) String graphs
4. Conclusion

# Conclusion

	Characters	Order	Complexity
Character inclusion			
Prefix and suffix			
Bricks			
String graphs			

**Thanks!**

**Questions?**