Static program analysis for string manipulation programs

Vincenzo Arceri Isabella Mastroeni

VPT 2019

April 2nd, 2019

UNIVERSITÀ
di VERONA
Dipartimento
di INFORMATICA

Dynamic languages

Dynamic languages have been increasingly used in a very wide range of fields and applications.

Dynamic languages

Dynamic languages have been increasingly used in a very wide range of fields and applications.

Dynamic languages

Dynamic languages have been increasingly used in a very wide range of fields and applications.

Multiple usage of strings:

- Implicit type conversion
- Object-property access
- String-to-code
- ….

Dynamic languages

Dynamic languages have been increasingly used in a very wide range of fields and applications.

Multiple usage of strings:

- Implicit type conversion
- Object-property access
- String-to-code
- ….

We address the problem of *statically* reasoning (using abstract interpretation) about these kind of programs.

Dynamic languages

Dynamic languages have been increasingly used in a very wide range of fields and applications.

Multiple usage of strings:

- Implicit type conversion
- Object-property access
- String-to-code
- ….

We address the problem of *statically* reasoning (using abstract interpretation) about these kind of programs.

JavaScript is the most popular scripting language, used to interact with HTML

```
<!DOCTYPE html>
<title>Example</title>

<p>
  <span id="timestamp"></span>
  milliseconds since midnight, January 1, 1970.
</p>

<script>
  document.addEventListener("DOMContentLoaded", function(event) {
    window.setTimeout(function(){
          document.location.reload(true);
      }, 3000);

  document.getElementById("timestamp").innerText = new Date().getTime
    ();
});
</script>
```

HTML

```
<!DOCTYPE html>
<title>Example</title>

<p>
  <span id="timestamp"></span>
  milliseconds since midnight, January 1, 1970.
</p>

<script>
```

JS

```
  document.addEventListener("DOMContentLoaded", function(event) {
    window.setTimeout(function(){
          document.location.reload(true);
      }, 3000);

  document.getElementById("timestamp").innerText = new Date().getTime
    ();
});
</script>
```

```
<!DOCTYPE html>
<title>Example</title>

<p>
  <span id="timestamp"></span>
  milliseconds since midnight, January 1, 1970.
</p>

<script>
  document.addEventListener("DOMContentLoaded", function(event) {
    window.setTimeout(function(){
          document.location.reload(true);
      }, 3000);

  document.getElementById("timestamp").innerText = new Date().getTime
    ();
});
</script>
```

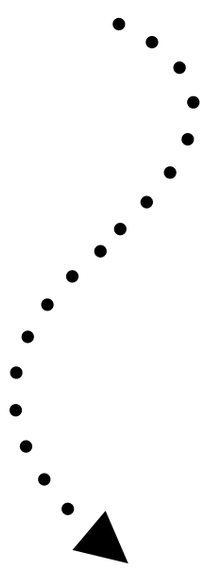
HTML

JS

JavaScript is dynamically typed

$$x = 5$$

```
x = 5
```



```
x = "str"
```

JavaScript has implicit type conversion

```
5 < 2 || "vpt"
```

```
5 < 2 || "vpt"
```

⋮

Type conversion

⋮

▼

```
5 < 2 || true
```

JavaScript has eval

or: "the goto of the web"

`eval` (”$a = a + 1$”)

String

`eval` (”$a = a + 1$”)

$\vdots$

Parsing of ”$a = a + 1$”

String

`eval` (”$a = a + 1$”)

⋮

Parsing of ”$a = a + 1$”

⋮

`a=a+1`

String

Code!

Assembling code at run-time is dangerous

```
eval('alert("Your query was ' + document.location.search + '");');
```

Assembling code at run-time is dangerous

```
eval('alert("Your query was ' + document.location.search + '");');
```

XSS, SSJS, CSRF, Code injection…

JavaScript malware

```
vd, ac, la = "";
v = "wZsZ"; m = "AYcYtYiYvYeYXY";
tt = "AObyaSZjectB";
l = "WYSYcYrYiYpYtY.YSYhYeYlYlY";

while (i+=2 < v.length)
  vd = vd + v.charAt(i);

while (j+=2 < m.length)
  ac = ac + m.charAt(j);

ac += tt.substring(tt.indexOf("O"), 3);
ac += tt.substring(tt.indexOf("j"), 11);

while (k+=2 < l.length)
  la = la + l.charAt(k);

d = vd + "=new " + ac + "(" + la + ")";
eval(d);
```

JavaScript malware

```
vd, ac, la = "";
v = "wZsZ"; m = "AYcYtYiYvYeYXY";
tt = "AObyaSZjectB";
l = "WYSYcYrYiYpYtY.YSYhYeYlYlY";

while (i+=2 < v.length)
  vd = vd + v.charAt(i);

while (j+=2 < m.length)
  ac = ac + m.charAt(j);

ac += tt.substring(tt.indexOf("O"), 3);
ac += tt.substring(tt.indexOf("j"), 11);

while (k+=2 < l.length)
  la = la + l.charAt(k);

d = vd + "=new " + ac + "(" + la + ")";
eval(d);
```

Which is the value of d before the eval execution?

JavaScript malware

```
vd, ac, la = "";
v = "wZsZ"; m = "AYcYtYiYvYeYXY";
tt = "AObyaSZjectB";
l = "WYSYcYrYiYpYtY.YSYhYeYlYlY";

while (i+=2 < v.length)
  vd = vd + v.charAt(i);

while (j+=2 < m.length)
  ac = ac + m.charAt(j);

ac += tt.substring(tt.indexOf("O"), 3);
ac += tt.substring(tt.indexOf("j"), 11);

while (k+=2 < l.length)
  la = la + l.charAt(k);

d = vd + "=new " + ac + "(" + la + ")";
eval(d);
```

JavaScript malware

```
vd, ac, la = "";
v = "wZsZ"; m = "AYcYtYiYvYeYXY";
tt = "AObyaSZjectB";
l = "WYSYcYrYiYpYtY.YSYhYeYlYlY";

while (i+=2 < v.length)
  vd = vd + v.charAt(i);

while (j+=2 < m.length)
  ac = ac + m.charAt(j);

ac += tt.substring(tt.indexOf("O"), 3);
ac += tt.substring(tt.indexOf("j"), 11);

while (k+=2 < l.length)
  la = la + l.charAt(k);

d = vd + "=new " + ac + "(" + la + ")";
eval(d);
```

TAJS SAFE JSAI

Static analysis of string manipulation programs

Static analysis of string manipulation programs

While many abstract domains for numerical values have been proposed, less work has been done on statically reasoning about strings.

Static analysis of string manipulation programs

While many abstract domains for numerical values have been proposed, less work has been done on statically reasoning about strings.

String value analyses become extremely important in scripting language context (e.g., object property access, eval, …)

Static analysis of string manipulation programs

While many abstract domains for numerical values have been proposed, less work has been done on statically reasoning about strings.

String value analyses become extremely important in scripting language context (e.g., object property access, eval, …)

This presentation

Static analysis of string manipulation programs

While many abstract domains for numerical values have been proposed, less work has been done on statically reasoning about strings.

String value analyses become extremely important in scripting language context (e.g., object property access, eval, …)

This presentation

- Formal semantics of a toy string manipulation language

Static analysis of string manipulation programs

While many abstract domains for numerical values have been proposed, less work has been done on statically reasoning about strings.

String value analyses become extremely important in scripting language context (e.g., object property access, eval, …)

This presentation

- Formal semantics of a toy string manipulation language
- Finite state automata abstract domain

Static analysis of string manipulation programs

While many abstract domains for numerical values have been proposed, less work has been done on statically reasoning about strings.

String value analyses become extremely important in scripting language context (e.g., object property access, eval, …)

This presentation

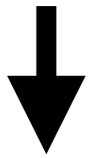
- Formal semantics of a toy string manipulation language
- Finite state automata abstract domain
- Formal abstract semantics of common string operations

IMP language

$$
\begin{array}{rcl}
\textbf{Exp} & ::= & \textbf{Id} \mid n \in \mathbb{Z} \mid \texttt{true} \mid \texttt{false} \mid \sigma \in \Sigma^* \\
& \mid & \textbf{Exp + Exp} \mid \ldots \\
& \mid & \text{// Other arithmetic and boolean expressions} \\
& \mid & \textbf{Exp}\texttt{.substring(}\textbf{Exp}\texttt{,}\textbf{Exp}\texttt{)} \\
& \mid & \textbf{Exp}\texttt{.charAt(}\textbf{Exp}\texttt{)} \\
& \mid & \textbf{Exp}\texttt{.indexOf(}\textbf{Exp}\texttt{)} \\
& \mid & \textbf{Exp}\texttt{.length} \\
\textbf{Block} & ::= & \texttt{\{ \}} \mid \texttt{\{ }\textbf{Stmt}\texttt{ \}} \\
\textbf{Stmt} & ::= & \textbf{Id}\texttt{ = }\textbf{Exp}\texttt{;} \\
& \mid & \texttt{if (}\textbf{Exp}\texttt{) }\textbf{Block}\texttt{ else }\textbf{Block} \\
& \mid & \texttt{while (}\textbf{Exp}\texttt{) }\textbf{Block} \\
& \mid & \textbf{Block} \\
& \mid & \textbf{Stmt Stmt} \\
& \mid & \texttt{;}
\end{array}
$$

IMP language

Integers



$$
\begin{array}{rcl}
\textbf{Exp} & ::= & \textbf{Id} \mid n \in \mathbb{Z} \mid \texttt{true} \mid \texttt{false} \mid \sigma \in \Sigma^* \\
& \mid & \textbf{Exp + Exp} \mid \ldots \\
& \mid & \text{// Other arithmetic and boolean expressions} \\
& \mid & \textbf{Exp}\texttt{.substring(}\textbf{Exp}\texttt{,}\textbf{Exp}\texttt{)} \\
& \mid & \textbf{Exp}\texttt{.charAt(}\textbf{Exp}\texttt{)} \\
& \mid & \textbf{Exp}\texttt{.indexOf(}\textbf{Exp}\texttt{)} \\
& \mid & \textbf{Exp}\texttt{.length} \\
\textbf{Block} & ::= & \texttt{\{ \}} \mid \texttt{\{ } \textbf{Stmt} \texttt{ \}} \\
\textbf{Stmt} & ::= & \textbf{Id} \texttt{ = } \textbf{Exp}\texttt{;} \\
& \mid & \texttt{if (}\textbf{Exp}\texttt{) } \textbf{Block} \texttt{ else } \textbf{Block} \\
& \mid & \texttt{while (}\textbf{Exp}\texttt{) } \textbf{Block} \\
& \mid & \textbf{Block} \\
& \mid & \textbf{Stmt Stmt} \\
& \mid & \texttt{;}
\end{array}
$$

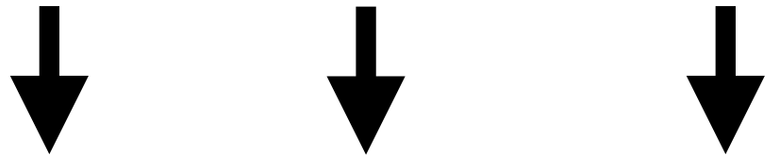
IMP language

Integers Booleans

$$
\begin{array}{rcl}
\textbf{Exp} & ::= & \textbf{Id} \mid n \in \mathbb{Z} \mid \texttt{true} \mid \texttt{false} \mid \sigma \in \Sigma^* \\
& \mid & \textbf{Exp + Exp} \mid \ldots \\
& \mid & \text{// Other arithmetic and boolean expressions} \\
& \mid & \textbf{Exp}\texttt{.substring(}\textbf{Exp}\texttt{,}\textbf{Exp}\texttt{)} \\
& \mid & \textbf{Exp}\texttt{.charAt(}\textbf{Exp}\texttt{)} \\
& \mid & \textbf{Exp}\texttt{.indexOf(}\textbf{Exp}\texttt{)} \\
& \mid & \textbf{Exp}\texttt{.length} \\
\textbf{Block} & ::= & \texttt{\{ \}} \mid \texttt{\{ }\textbf{Stmt}\texttt{ \}} \\
\textbf{Stmt} & ::= & \textbf{Id}\texttt{ = }\textbf{Exp}\texttt{;} \\
& \mid & \texttt{if (}\textbf{Exp}\texttt{) }\textbf{Block}\texttt{ else }\textbf{Block} \\
& \mid & \texttt{while (}\textbf{Exp}\texttt{) }\textbf{Block} \\
& \mid & \textbf{Block} \\
& \mid & \textbf{Stmt Stmt} \\
& \mid & \texttt{;}
\end{array}
$$

IMP language

Integers Booleans Strings



$$
\begin{array}{rcl}
\textbf{Exp} & ::= & \textbf{Id} \mid n \in \mathbb{Z} \mid \texttt{true} \mid \texttt{false} \mid \sigma \in \Sigma^* \\
& \mid & \textbf{Exp + Exp} \mid \ldots \\
& \mid & \text{// Other arithmetic and boolean expressions} \\
& \mid & \textbf{Exp}\texttt{.substring(}\textbf{Exp}\texttt{,}\textbf{Exp}\texttt{)} \\
& \mid & \textbf{Exp}\texttt{.charAt(}\textbf{Exp}\texttt{)} \\
& \mid & \textbf{Exp}\texttt{.indexOf(}\textbf{Exp}\texttt{)} \\
& \mid & \textbf{Exp}\texttt{.length} \\
\textbf{Block} & ::= & \texttt{\{ \}} \mid \texttt{\{ }\textbf{Stmt}\texttt{ \}} \\
\textbf{Stmt} & ::= & \textbf{Id}\texttt{ = }\textbf{Exp}\texttt{;} \\
& \mid & \texttt{if (}\textbf{Exp}\texttt{) }\textbf{Block}\texttt{ else }\textbf{Block} \\
& \mid & \texttt{while (}\textbf{Exp}\texttt{) }\textbf{Block} \\
& \mid & \textbf{Block} \\
& \mid & \textbf{Stmt Stmt} \\
& \mid & \texttt{;}
\end{array}
$$

IMP language

Integers Booleans Strings

$$
\begin{array}{rcl}
\textbf{Exp} & ::= & \textbf{Id} \mid n \in \mathbb{Z} \mid \texttt{true} \mid \texttt{false} \mid \sigma \in \Sigma^* \\
& \mid & \textbf{Exp + Exp} \mid \ldots \\
& \mid & \text{// Other arithmetic and boolean expressions} \\
& \mid & \textbf{Exp}\texttt{.substring(}\textbf{Exp}\texttt{,}\textbf{Exp}\texttt{)} \\
& \mid & \textbf{Exp}\texttt{.charAt(}\textbf{Exp}\texttt{)} \\
& \mid & \textbf{Exp}\texttt{.indexOf(}\textbf{Exp}\texttt{)} \\
& \mid & \textbf{Exp}\texttt{.length} \\
\textbf{Block} & ::= & \texttt{\{ \}} \mid \texttt{\{ } \textbf{Stmt} \texttt{ \}} \\
\textbf{Stmt} & ::= & \textbf{Id} \texttt{ = } \textbf{Exp}\texttt{;} \\
& \mid & \texttt{if (}\textbf{Exp}\texttt{) } \textbf{Block} \texttt{ else } \textbf{Block} \\
& \mid & \texttt{while (}\textbf{Exp}\texttt{) } \textbf{Block} \\
& \mid & \textbf{Block} \\
& \mid & \textbf{Stmt Stmt} \\
& \mid & \texttt{;}
\end{array}
$$

Implicit type conversion

```
if ("hello") {
   [code]
}
```

Implicit type conversion

```
if ("hello") {
   [code]
}
```

Implicit type conversion

```
if ("hello") {
   [code]
}
```

Str-to-bool →

Implicit type conversion

```
if ("hello") {
   [code]
}
```

Str-to-bool →

```
if (true) {
   [code]
}
```

Implicit type conversion

```
if ("hello") {
   [code]
}
```

Str-to-bool →

```
if (true) {
   [code]
}
```

```
5 + "12"
```

Implicit type conversion

```
if ("hello") {
   [code]
}
```

Str-to-bool →

```
if (true) {
   [code]
}
```

```
5 + "12"
```

Implicit type conversion

```
if ("hello") {
   [code]
}
```

Str-to-bool →

```
if (true) {
   [code]
}
```

```
5 + "12"
```

Int-to-str →

Implicit type conversion

```
if ("hello") {
   [code]
}
```

Str-to-bool →

```
if (true) {
   [code]
}
```

```
5 + "12"
```

Int-to-str →

```
"5" + "12" = "512"
```

Implicit type conversion

```
if ("hello") {
   [code]
}
```

Str-to-bool →

```
if (true) {
   [code]
}
```

```
5 + "12"
```

Int-to-str →

```
"5" + "12" = "512"
```

```
5 * "12"
```

Implicit type conversion

```
if ("hello") {
   [code]
}
```

Str-to-bool →

```
if (true) {
   [code]
}
```

```
5 + "12"
```

Int-to-str →

```
"5" + "12" = "512"
```

```
5 * "12"
```

Implicit type conversion

```
if ("hello") {
   [code]
}
```

Str-to-bool →

```
if (true) {
   [code]
}
```

`5` + `"12"` → Int-to-str → `"5" + "12" = "512"`

`5 * "12"` → Str-to-int →

Implicit type conversion

```
if ("hello") {
   [code]
}
```

Str-to-bool →

```
if (true) {
   [code]
}
```

```
5 + "12"
```

Int-to-str →

```
"5" + "12" = "512"
```

```
5 * "12"
```

Str-to-int →

```
5 * 12 = 60
```

String operations

String operations

Substring - string between two indexes

String operations

Substring - string between two indexes

```
"hello".substring(2,4) = "ll"
```

String operations

Substring - string between two indexes

```
"hello".substring(2,4) = "ll"

"hello".substring(4,2)
```

String operations

Substring - string between two indexes

```
"hello".substring(2,4) = "ll"

"hello".substring(4,2) = "hello".substring(2,4)
```

String operations

Substring - string between two indexes

```
"hello".substring(2,4) = "ll"

"hello".substring(4,2) = "hello".substring(2,4)
```

CharAt - similar to substring

String operations

Substring - string between two indexes

```
"hello".substring(2,4) = "ll"

"hello".substring(4,2) = "hello".substring(2,4)
```

CharAt - similar to substring

```
"hello".charAt(0) = "h"
```

String operations

Substring - string between two indexes

```
"hello".substring(2,4) = "ll"

"hello".substring(4,2) = "hello".substring(2,4)
```

CharAt - similar to substring

```
"hello".charAt(0) = "h"     "hello".charAt(-1) = ""
```

String operations

Substring - string between two indexes

```
"hello".substring(2,4) = "ll"
"hello".substring(4,2) = "hello".substring(2,4)
```

CharAt - similar to substring

```
"hello".charAt(0) = "h"     "hello".charAt(-1) = ""
```

Length - returns the length of the input string

String operations

Substring - string between two indexes

```
"hello".substring(2,4) = "ll"
"hello".substring(4,2) = "hello".substring(2,4)
```

CharAt - similar to substring

```
"hello".charAt(0) = "h"     "hello".charAt(-1) = ""
```

Length - returns the length of the input string

IndexOf - position of the first occurrence of a given substring

String operations

Substring - string between two indexes

```
"hello".substring(2,4) = "ll"

"hello".substring(4,2) = "hello".substring(2,4)
```

CharAt - similar to substring

```
"hello".charAt(0) = "h"     "hello".charAt(-1) = ""
```

Length - returns the length of the input string

IndexOf - position of the first occurrence of a given substring

```
"hello".indexOf("h") = 0
```

String operations

Substring - string between two indexes

```
"hello".substring(2,4) = "ll"
"hello".substring(4,2) = "hello".substring(2,4)
```

CharAt - similar to substring

```
"hello".charAt(0) = "h"     "hello".charAt(-1) = ""
```

Length - returns the length of the input string

IndexOf - position of the first occurrence of a given substring

```
"hello".indexOf("h") = 0  "hello".indexOf("abc") = -1
```

The finite state automata domain

The finite state automata domain

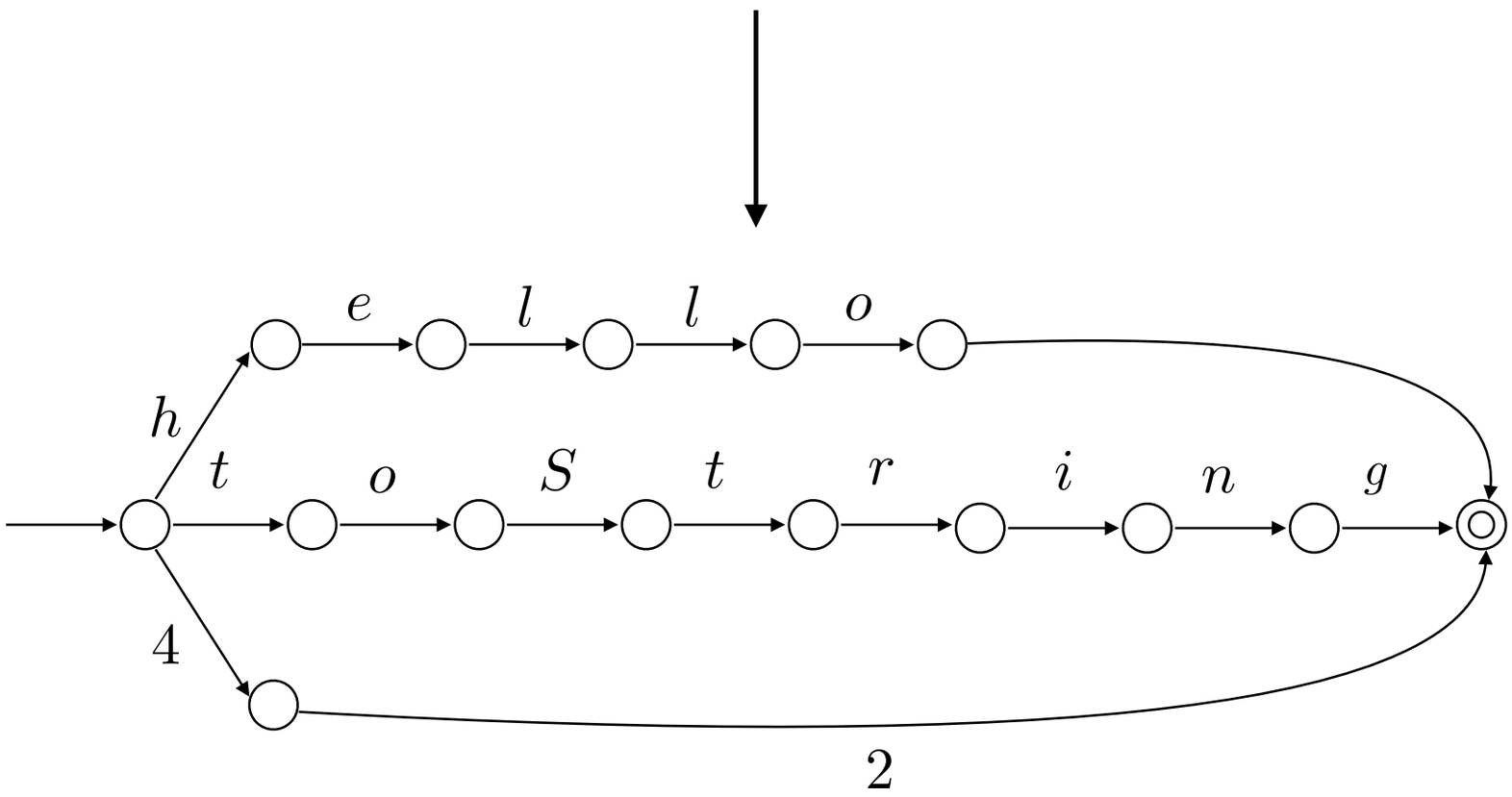
We abstract a strings set to the finite state automaton that recognizes them

```
{ "42",
"toString",
"hello"}
```

The finite state automata domain

We abstract a strings set to the finite state automaton that recognizes them

```
{ "42",
"toString",
"hello"}
```



The finite state automata domain

```
if (...)
   x = "a++;";
else
   x = "b++;";
```

The finite state automata domain

The guard value is unknown

```
if (...)
   x = "a++;";
else
   x = "b++;";
```

The finite state automata domain

The guard value is unknown

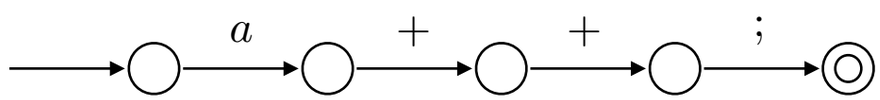
```
if (...)
   x = "a++;";
else
   x = "b++;";
```

Which is the value of `x` after the `if` execution?

The finite state automata domain

The guard value is unknown

```
if (...)
   x = "a++;";
else
   x = "b++;";
```

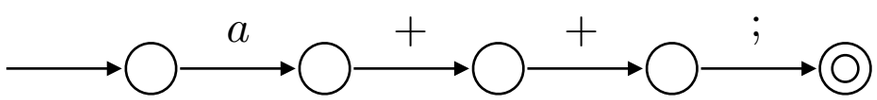


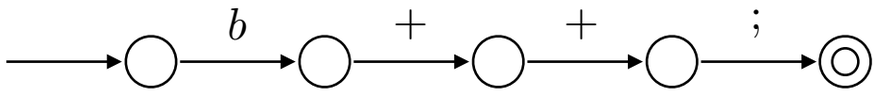
Which is the value of `x` after the `if` execution?

The finite state automata domain

The guard value is unknown

```
if (...)
   x = "a++;";
else
   x = "b++;";
```



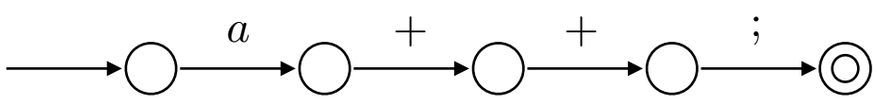


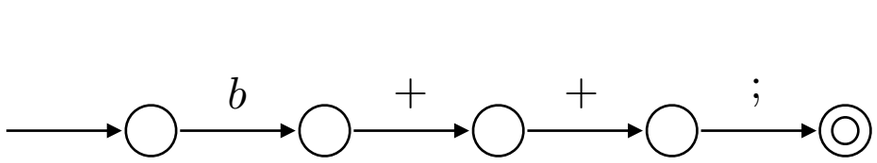
Which is the value of `x` after the `if` execution?

The finite state automata domain

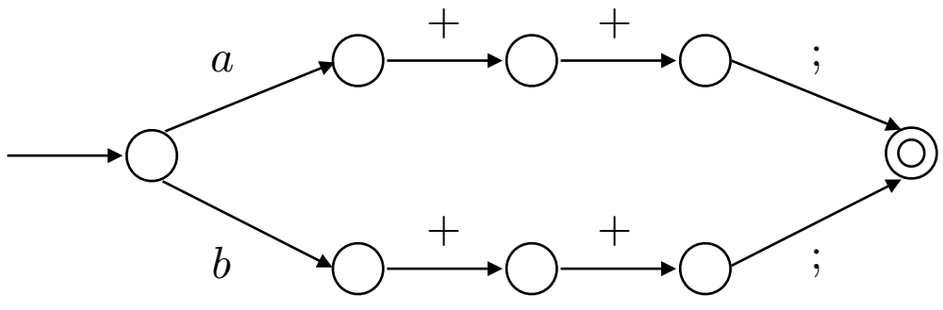
The guard value is unknown

```
if (...)
   x = "a++;";
else
   x = "b++;";
```





Which is the value of `x` after the `if` execution?



The finite state automata domain

```
while (...)
  x = x + "s++;";
```

The finite state automata domain

The guard value is unknown

```
while (...)
  x = x + "s++;";
```

The finite state automata domain

The guard value is unknown

```
while (...)
  x = x + "s++;";
```

```
"s++;s++;s++;s++;..."
```

The finite state automata domain

The guard value is unknown

```
while (...)
  x = x + "s++;";
```

`"s++;s++;s++;s++;..."`

Which is the value of `x` after the `while` execution?

The finite state automata domain

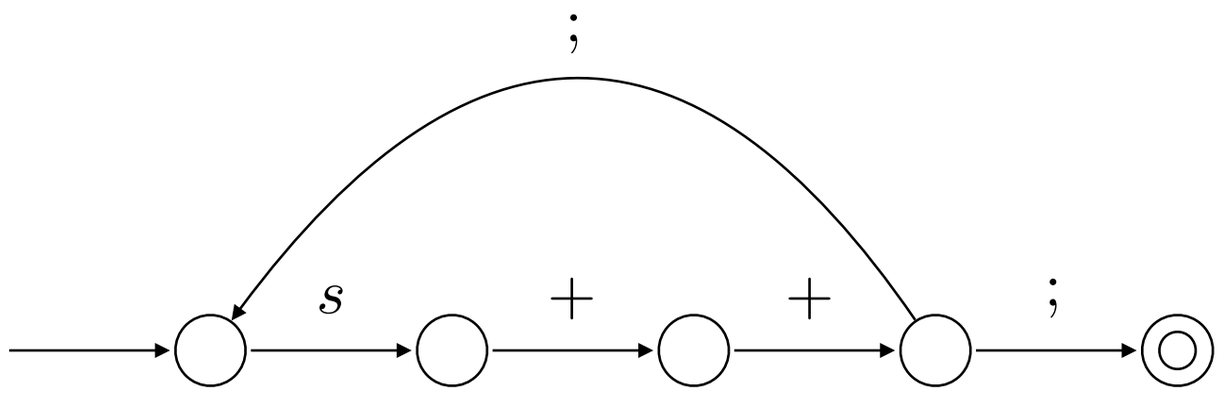
The guard value is unknown

```
while (...)
  x = x + "s++;";
```

"s++;s++;s++;s++;..."

Which is the value of `x` after the `while` execution?

Widening for automata

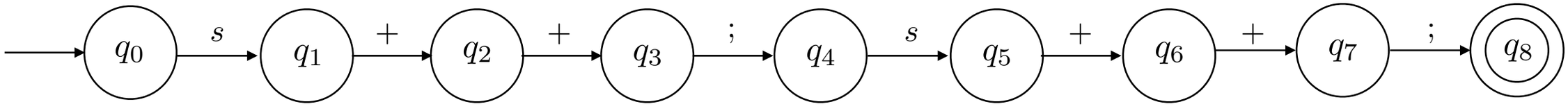


"Widening for automata", V. D'Silva, PhD Thesis 2006

Finite state automata abstract domain

Widening for automata

After two iterations:

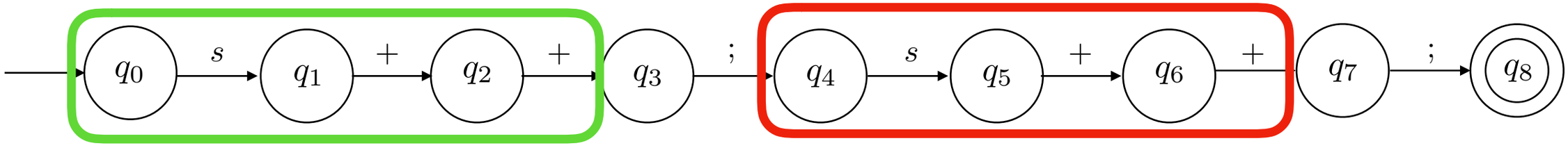


Now we apply ∇_3 : it merges the states that recognized the same strings of length 3.

Finite state automata abstract domain

Widening for automata

After two iterations:



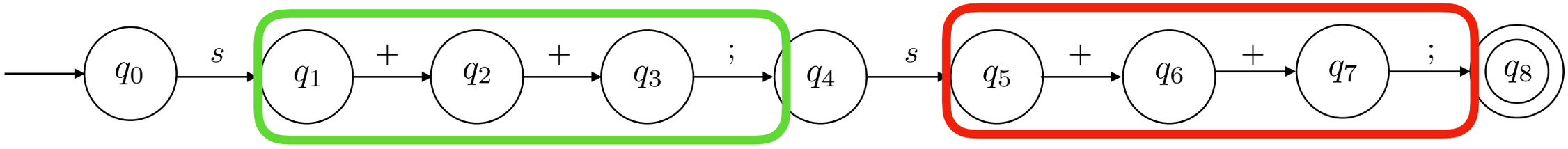
Now we apply ∇_3 : it merges the states that recognized the same strings of length 3.

$\{q_0, q_4\}$

Finite state automata abstract domain

Widening for automata

After two iterations:



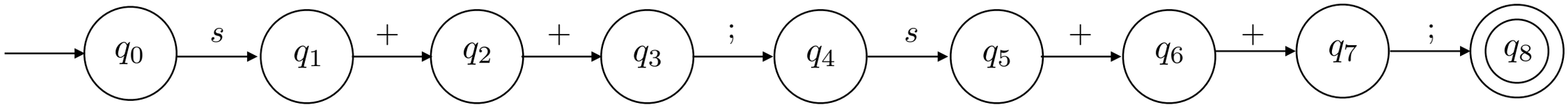
Now we apply ∇_3 : it merges the states that recognized the same strings of length 3.

$\{q_0, q_4\}$ $\{q_1, q_5\}$

Finite state automata abstract domain

Widening for automata

After two iterations:



Now we apply ∇_3 : it merges the states that recognized the same strings of length 3.

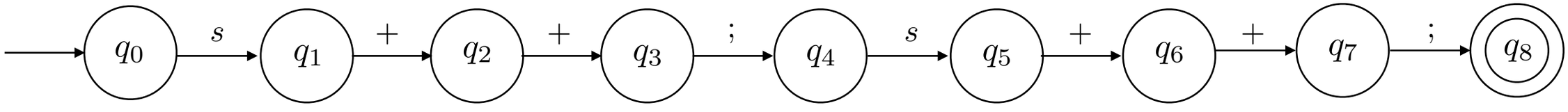
$\{q_6\}$ $\{q_7\}$ $\{q_8\}$

$\{q_0, q_4\}$ $\{q_1, q_5\}$ $\{q_2\}$ $\{q_3\}$

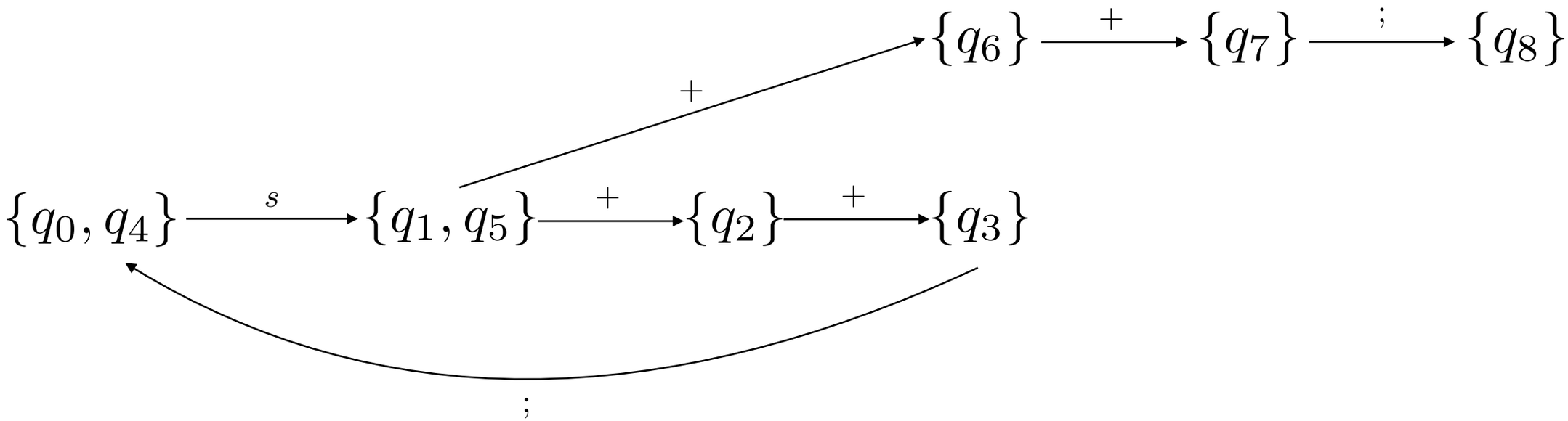
Finite state automata abstract domain

Widening for automata

After two iterations:



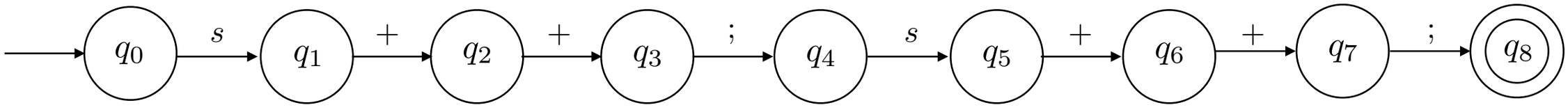
Now we apply ∇_3 : it merges the states that recognized the same strings of length 3.



Finite state automata abstract domain

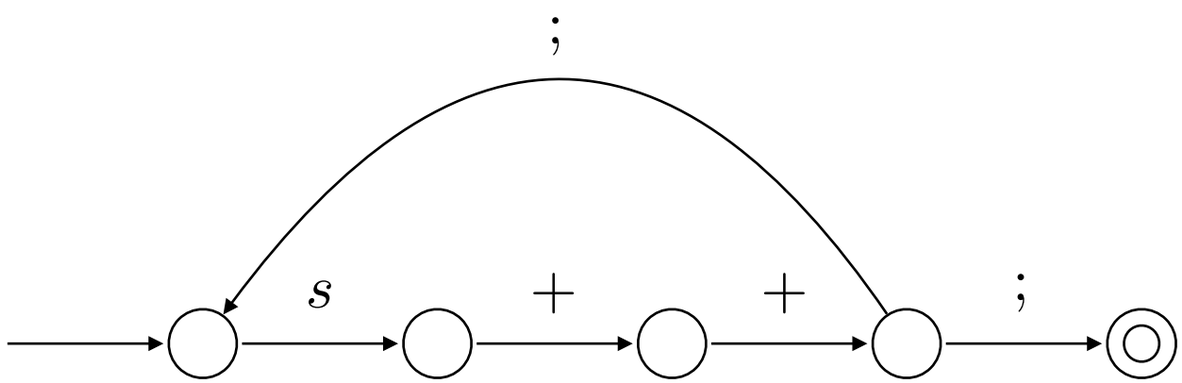
Widening for automata

After two iterations:



Now we apply ∇_3 : it merges the states that recognized the same strings of length 3.

```
while (...)
   x = x + "s++;";
```



Abstract semantics of string operations

Abstract semantics of string operations

Abstract domains

Intervals

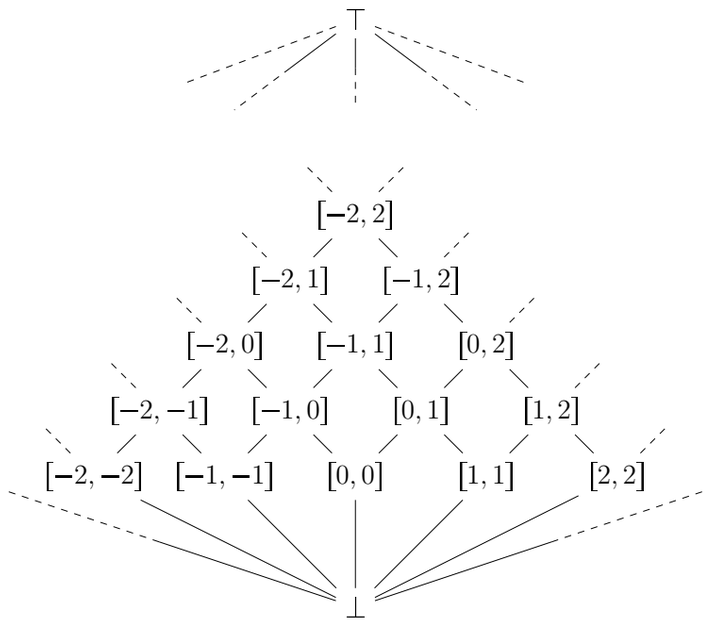
Booleans

Automata

Abstract semantics of string operations

Abstract domains

Intervals



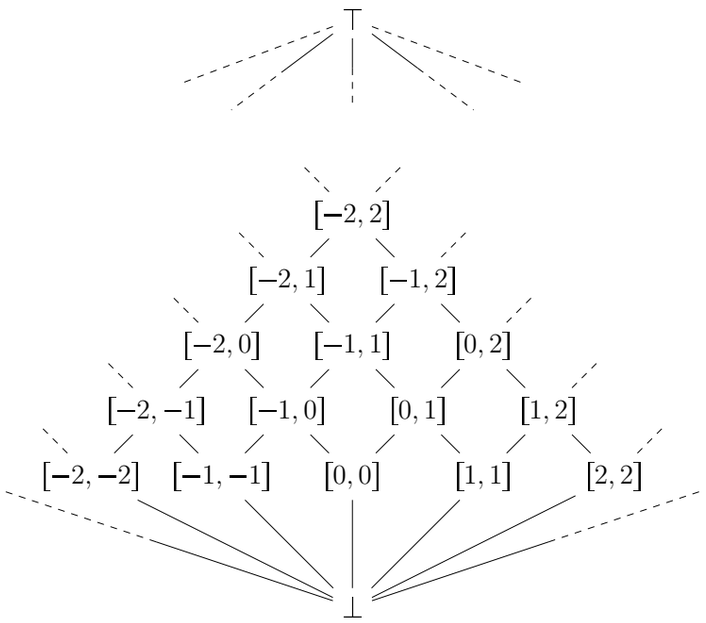
Booleans

Automata

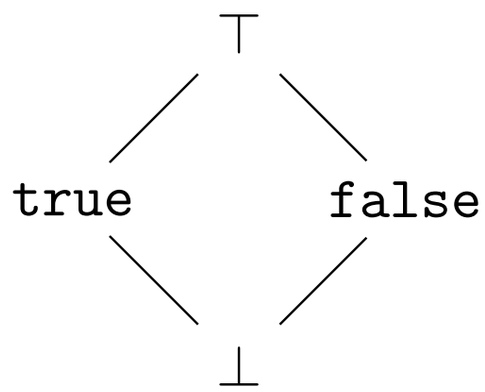
Abstract semantics of string operations

Abstract domains

Intervals



Booleans

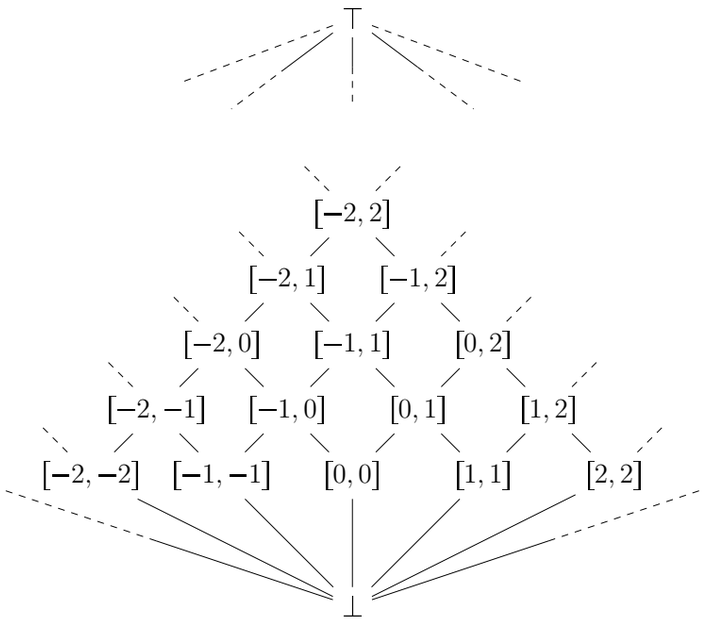


Automata

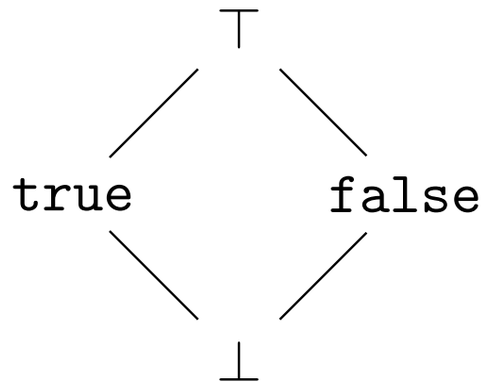
Abstract semantics of string operations

Abstract domains

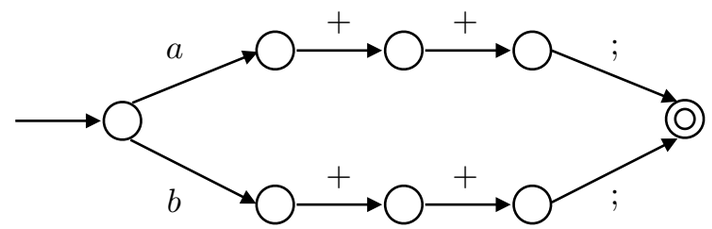
Intervals



Booleans



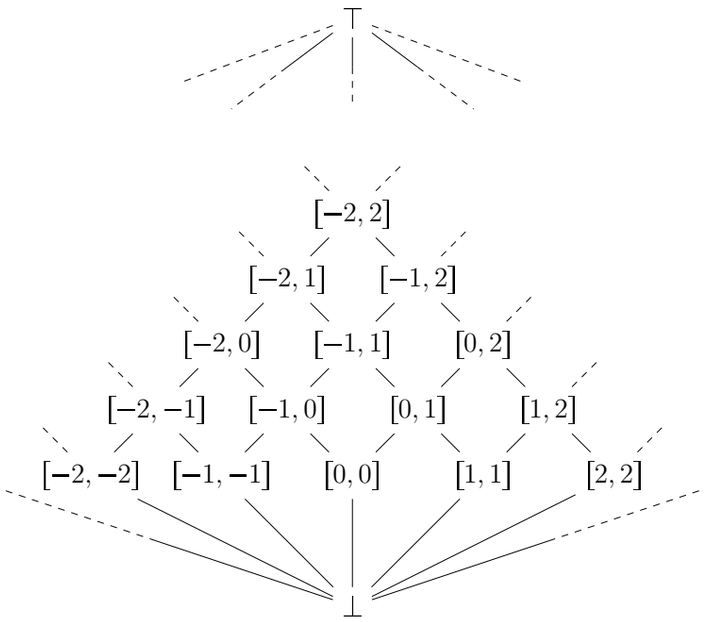
Automata



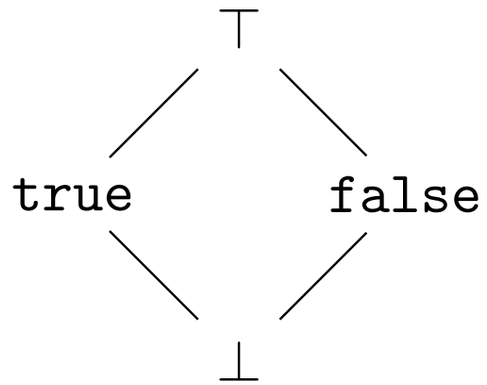
Abstract semantics of string operations

Abstract domains

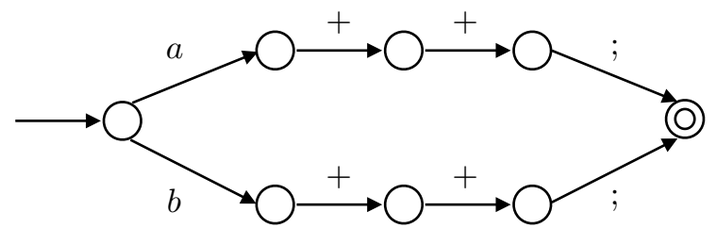
Intervals



Booleans



Automata



We define the abstract semantics of the following string operations:

- substring
- charAt
- length
- indexOf

Substring

Substring

$$\texttt{substr}^\sharp(A, [i, j], [l, k])$$

Substring

Substring

We first define substring on automata between two indexes i and j

Substring

We first define substring on automata between two indexes i and j

$$\texttt{substr}(A, i, j) = \mathcal{RQ}(\mathcal{SU}(A, i), \mathcal{SU}(A, j)) \cap \Sigma^{j-i} \cup \mathcal{SU}(A, i)) \cap \Sigma^{<j-i}$$

Substring

We first define substring on automata between two indexes i and j

$$\texttt{substr}(A, i, j) = \mathcal{RQ}(\mathcal{SU}(A, i), \mathcal{SU}(A, j)) \cap \Sigma^{j-i} \cup \mathcal{SU}(A, i)) \cap \Sigma^{<j-i}$$

Let's consider $L = \{\texttt{papers}, \texttt{lan}, \texttt{hello}\}$

Substring

We first define substring on automata between two indexes i and j

$$\texttt{substr}(A, i, j) = \mathcal{RQ}(\mathcal{SU}(A, i), \mathcal{SU}(A, j)) \cap \Sigma^{j-i} \cup \mathcal{SU}(A, i)) \cap \Sigma^{<j-i}$$

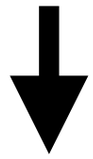
Let's consider $L = \{\texttt{papers}, \texttt{lan}, \texttt{hello}\}$

Substrings from 1 to 4: $\{\texttt{ape}, \texttt{ell}, \texttt{an}\}$

Substring

We first define substring on automata between two indexes i and j

Suffixes of A
starting from i



$$\texttt{substr}(A, i, j) = \mathcal{RQ}(\mathcal{SU}(A, i), \mathcal{SU}(A, j)) \cap \Sigma^{j-i} \cup \mathcal{SU}(A, i)) \cap \Sigma^{<j-i}$$

Let's consider $L = \{\texttt{papers}, \texttt{lan}, \texttt{hello}\}$

Substrings from 1 to 4: $\{\texttt{ape}, \texttt{ell}, \texttt{an}\}$

Substring

We first define substring on automata between two indexes i and j

$$\texttt{substr}(A, i, j) = \mathcal{RQ}(\mathcal{SU}(A, i), \mathcal{SU}(A, j)) \cap \Sigma^{j-i} \cup \mathcal{SU}(A, i)) \cap \Sigma^{<j-i}$$

Let's consider $L = \{\texttt{papers}, \texttt{lan}, \texttt{hello}\}$

Substrings from 1 to 4: $\{\texttt{ape}, \texttt{ell}, \texttt{an}\}$

Substring

We first define substring on automata between two indexes i and j

$$\texttt{substr}(A, i, j) = \mathcal{RQ}(\mathcal{SU}(A, i), \mathcal{SU}(A, j)) \cap \Sigma^{j-i} \cup \mathcal{SU}(A, i)) \cap \Sigma^{<j-i}$$

$$\downarrow$$

$$\{\texttt{apers}, \texttt{an}, \texttt{ello}\}$$

Let's consider $L = \{\texttt{papers}, \texttt{lan}, \texttt{hello}\}$

Substrings from 1 to 4: $\{\texttt{ape}, \texttt{ell}, \texttt{an}\}$

Substring

We first define substring on automata between two indexes i and j

$$\texttt{substr}(A, i, j) = \mathcal{RQ}(\mathcal{SU}(A, i), \mathcal{SU}(A, j)) \cap \Sigma^{j-i} \cup \mathcal{SU}(A, i)) \cap \Sigma^{<j-i}$$

$\{\texttt{apers}, \texttt{an}, \texttt{ello}\}$ $\{\texttt{rs}, \texttt{o}\}$

Let's consider $L = \{\texttt{papers}, \texttt{lan}, \texttt{hello}\}$

Substrings from 1 to 4: $\{\texttt{ape}, \texttt{ell}, \texttt{an}\}$

Substring

We first define substring on automata between two indexes i and j

Right quotient

$$\texttt{substr}(A, i, j) = \mathcal{RQ}(\mathcal{SU}(A, i), \mathcal{SU}(A, j)) \cap \Sigma^{j-i} \cup \mathcal{SU}(A, i)) \cap \Sigma^{<j-i}$$

$\{\texttt{apers}, \texttt{an}, \texttt{ello}\}$ $\{\texttt{rs}, \texttt{o}\}$

Let's consider $L = \{\texttt{papers}, \texttt{lan}, \texttt{hello}\}$

Substrings from 1 to 4: $\{\texttt{ape}, \texttt{ell}, \texttt{an}\}$

Substring

We first define substring on automata between two indexes i and j

$$\texttt{substr}(A, i, j) = \mathcal{RQ}(\mathcal{SU}(A, i), \mathcal{SU}(A, j)) \cap \Sigma^{j-i} \cup \mathcal{SU}(A, i)) \cap \Sigma^{<j-i}$$

$\{\texttt{apers}, \texttt{an}, \texttt{ello}\}$ $\{\texttt{rs}, \texttt{o}\}$

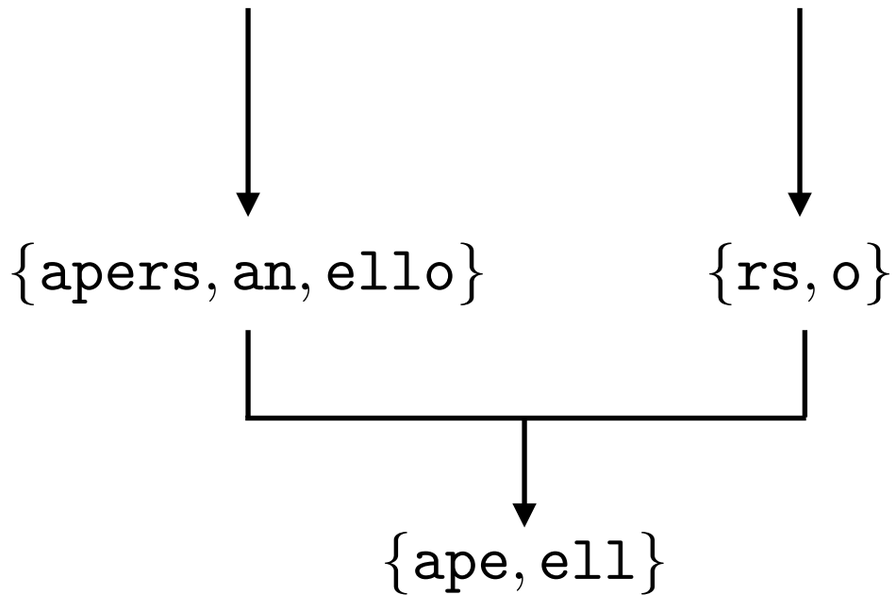
Let's consider $L = \{\texttt{papers}, \texttt{lan}, \texttt{hello}\}$

Substrings from 1 to 4: $\{\texttt{ape}, \texttt{ell}, \texttt{an}\}$

Substring

We first define substring on automata between two indexes i and j

$$\texttt{substr}(A, i, j) = \mathcal{RQ}(\mathcal{SU}(A, i), \mathcal{SU}(A, j)) \cap \Sigma^{j-i} \cup \mathcal{SU}(A, i)) \cap \Sigma^{<j-i}$$



Let's consider $L = \{\texttt{papers}, \texttt{lan}, \texttt{hello}\}$

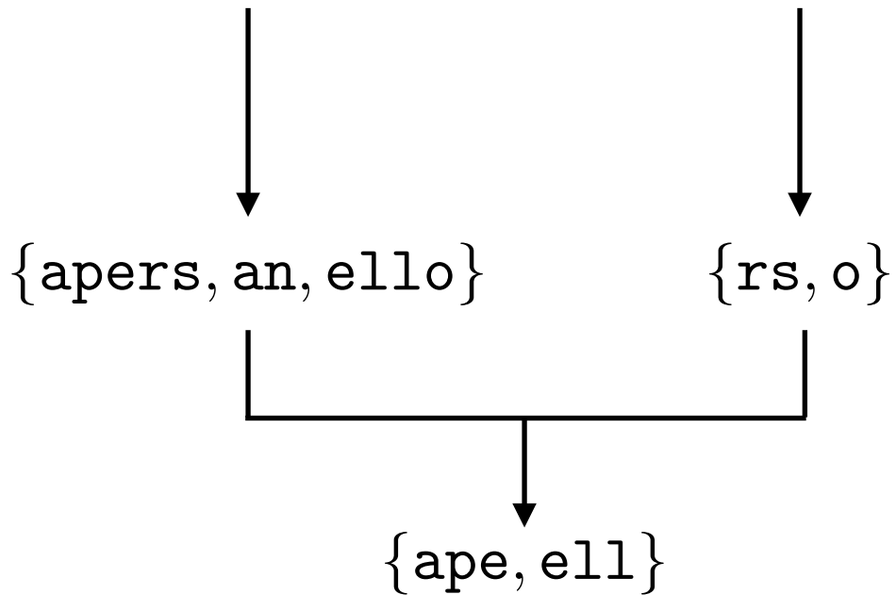
Substrings from 1 to 4: $\{\texttt{ape}, \texttt{ell}, \texttt{an}\}$

Substring

We first define substring on automata between two indexes i and j

Strings of length j-i

$$\texttt{substr}(A, i, j) = \mathcal{RQ}(\mathcal{SU}(A, i), \mathcal{SU}(A, j)) \cap \Sigma^{j-i} \cup \mathcal{SU}(A, i)) \cap \Sigma^{<j-i}$$



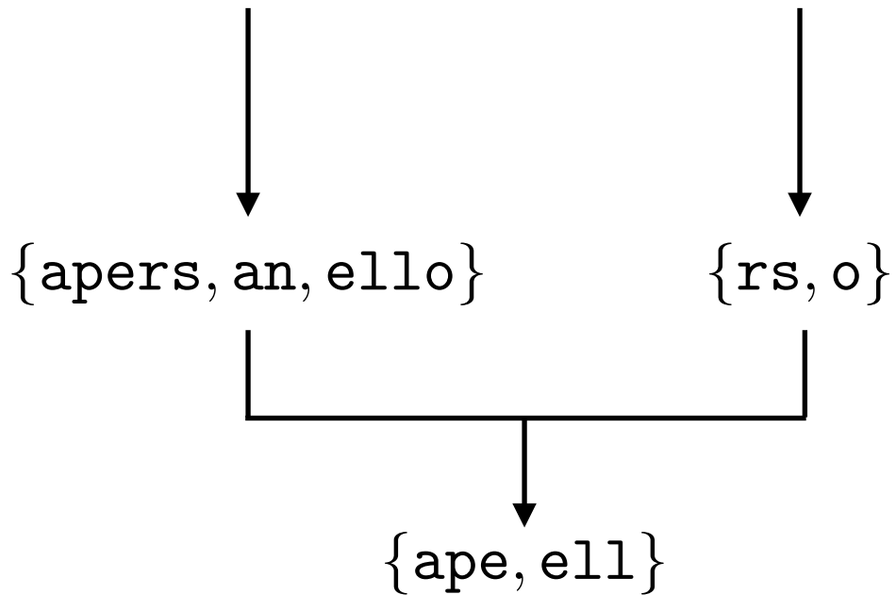
Let's consider $L = \{\texttt{papers}, \texttt{lan}, \texttt{hello}\}$

Substrings from 1 to 4: $\{\texttt{ape}, \texttt{ell}, \texttt{an}\}$

Substring

We first define substring on automata between two indexes i and j

$$\texttt{substr}(A, i, j) = \mathcal{RQ}(\mathcal{SU}(A, i), \mathcal{SU}(A, j)) \cap \Sigma^{j-i} \cup \mathcal{SU}(A, i)) \cap \Sigma^{<j-i}$$



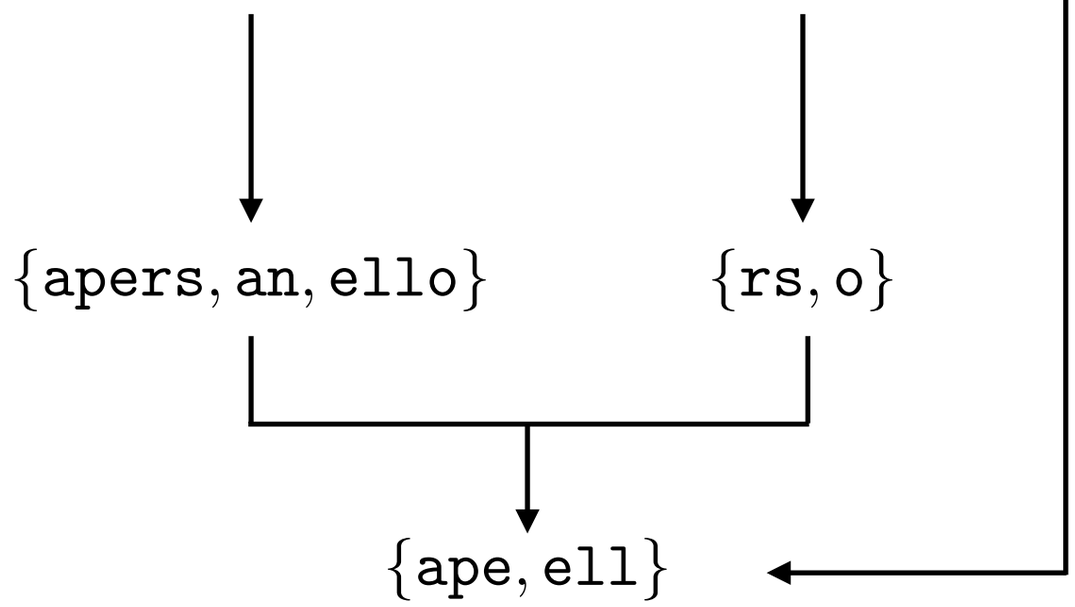
Let's consider $L = \{\texttt{papers}, \texttt{lan}, \texttt{hello}\}$

Substrings from 1 to 4: $\{\texttt{ape}, \texttt{ell}, \texttt{an}\}$

Substring

We first define substring on automata between two indexes i and j

$$\texttt{substr}(A, i, j) = \mathcal{RQ}(\mathcal{SU}(A, i), \mathcal{SU}(A, j)) \cap \Sigma^{j-i} \cup \mathcal{SU}(A, i)) \cap \Sigma^{<j-i}$$



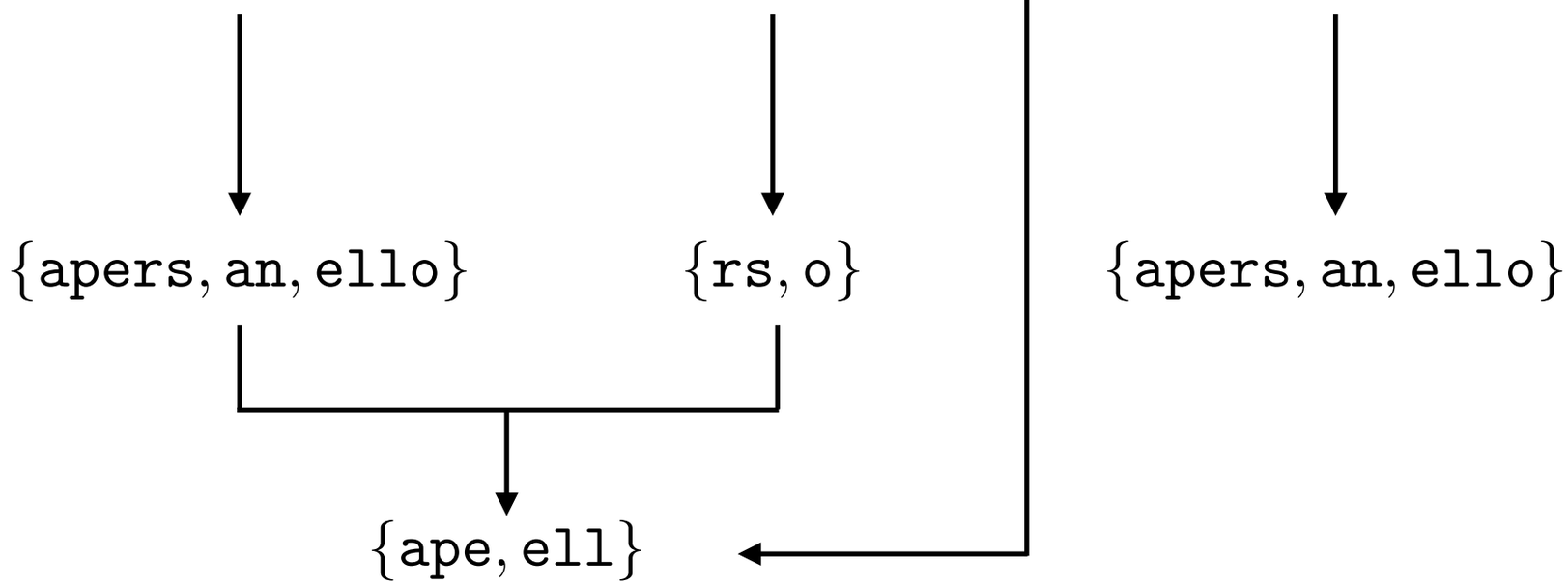
Let's consider $L = \{\texttt{papers}, \texttt{lan}, \texttt{hello}\}$

Substrings from 1 to 4: $\{\texttt{ape}, \texttt{ell}, \texttt{an}\}$

Substring

We first define substring on automata between two indexes i and j

$$\texttt{substr}(A, i, j) = \mathcal{RQ}(\mathcal{SU}(A, i), \mathcal{SU}(A, j)) \cap \Sigma^{j-i} \cup \mathcal{SU}(A, i)) \cap \Sigma^{<j-i}$$



Let's consider $L = \{\texttt{papers}, \texttt{lan}, \texttt{hello}\}$

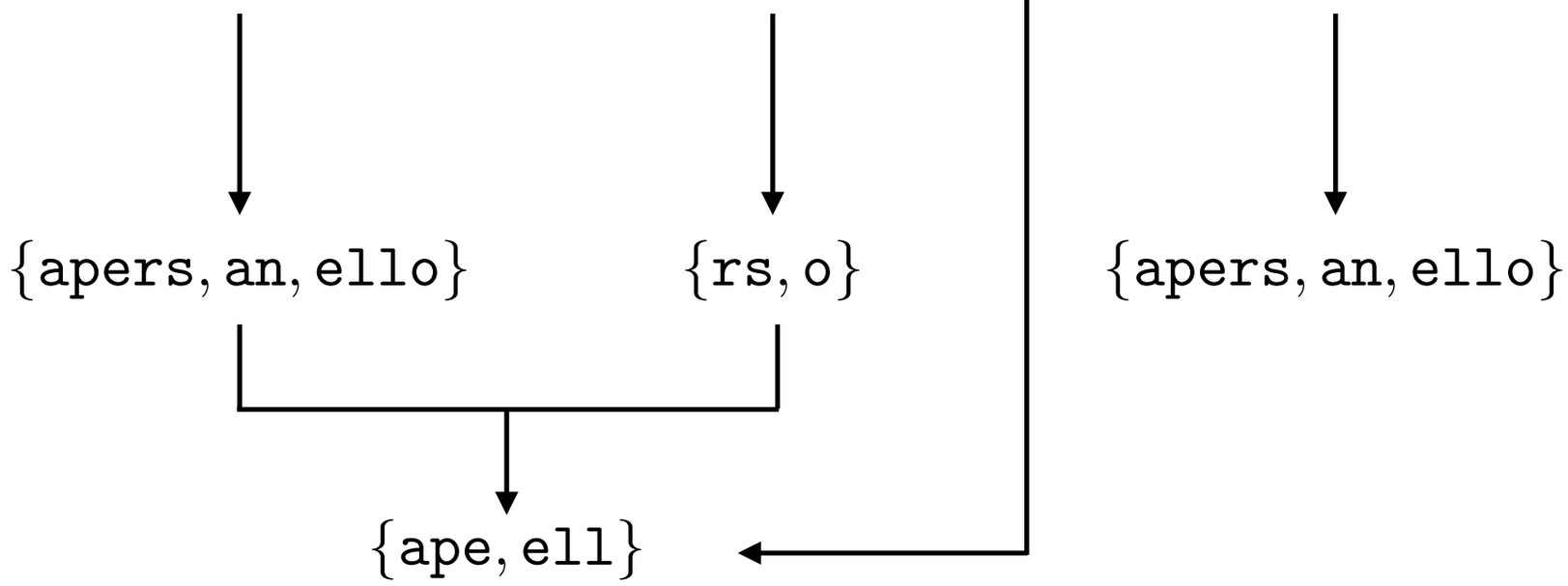
Substrings from 1 to 4: $\{\texttt{ape}, \texttt{ell}, \texttt{an}\}$

Substring

We first define substring on automata between two indexes i and j

Strings of length less than j-i

$$\texttt{substr}(A, i, j) = \mathcal{RQ}(\mathcal{SU}(A, i), \mathcal{SU}(A, j)) \cap \Sigma^{j-i} \cup \mathcal{SU}(A, i)) \cap \Sigma^{<j-i}$$



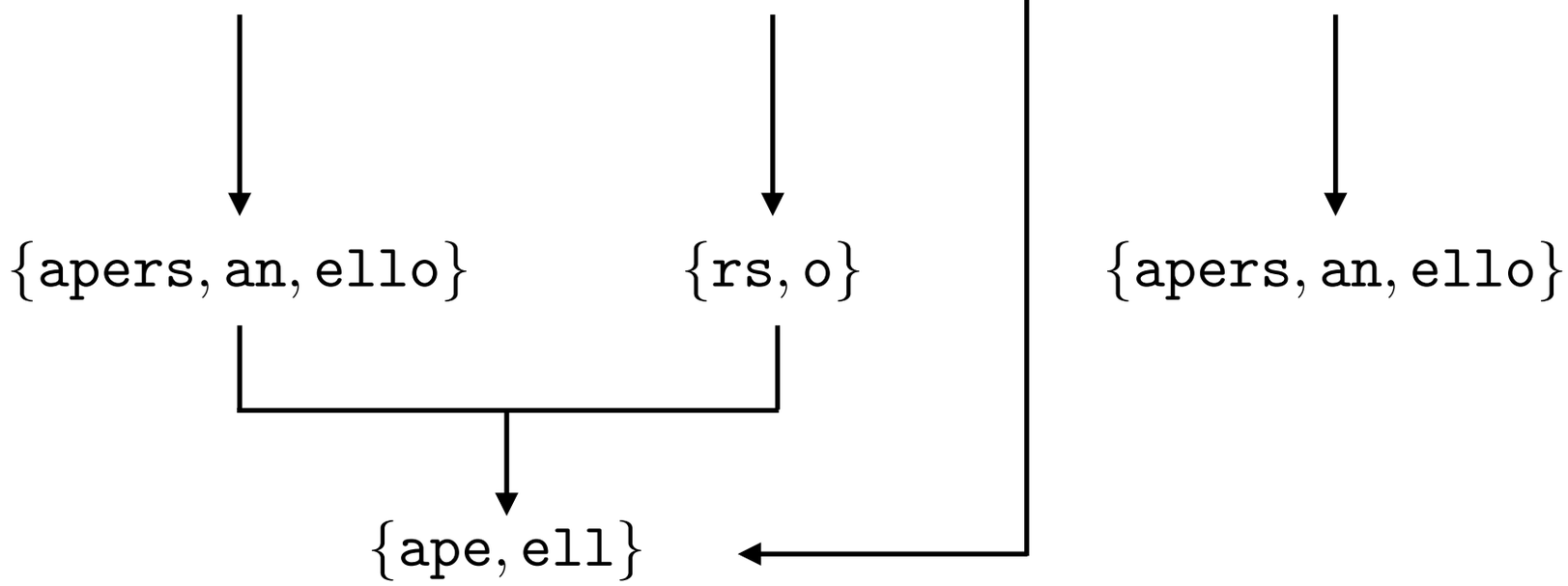
Let's consider $L = \{\texttt{papers}, \texttt{lan}, \texttt{hello}\}$

Substrings from 1 to 4: $\{\texttt{ape}, \texttt{ell}, \texttt{an}\}$

Substring

We first define substring on automata between two indexes i and j

$$\texttt{substr}(A, i, j) = \mathcal{RQ}(\mathcal{SU}(A, i), \mathcal{SU}(A, j)) \cap \Sigma^{j-i} \cup \mathcal{SU}(A, i)) \cap \Sigma^{<j-i}$$



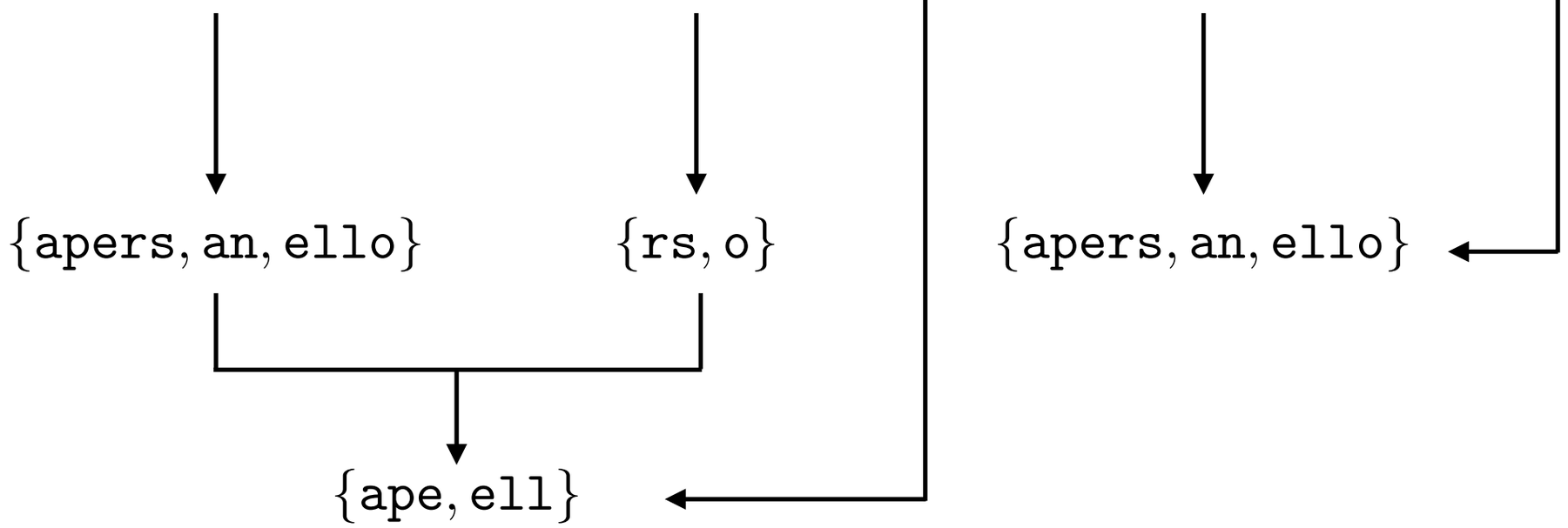
Let's consider $L = \{\texttt{papers}, \texttt{lan}, \texttt{hello}\}$

Substrings from 1 to 4: $\{\texttt{ape}, \texttt{ell}, \texttt{an}\}$

Substring

We first define substring on automata between two indexes i and j

$$\texttt{substr}(A, i, j) = \mathcal{RQ}(\mathcal{SU}(A, i), \mathcal{SU}(A, j)) \cap \Sigma^{j-i} \cup \mathcal{SU}(A, i)) \cap \Sigma^{<j-i}$$



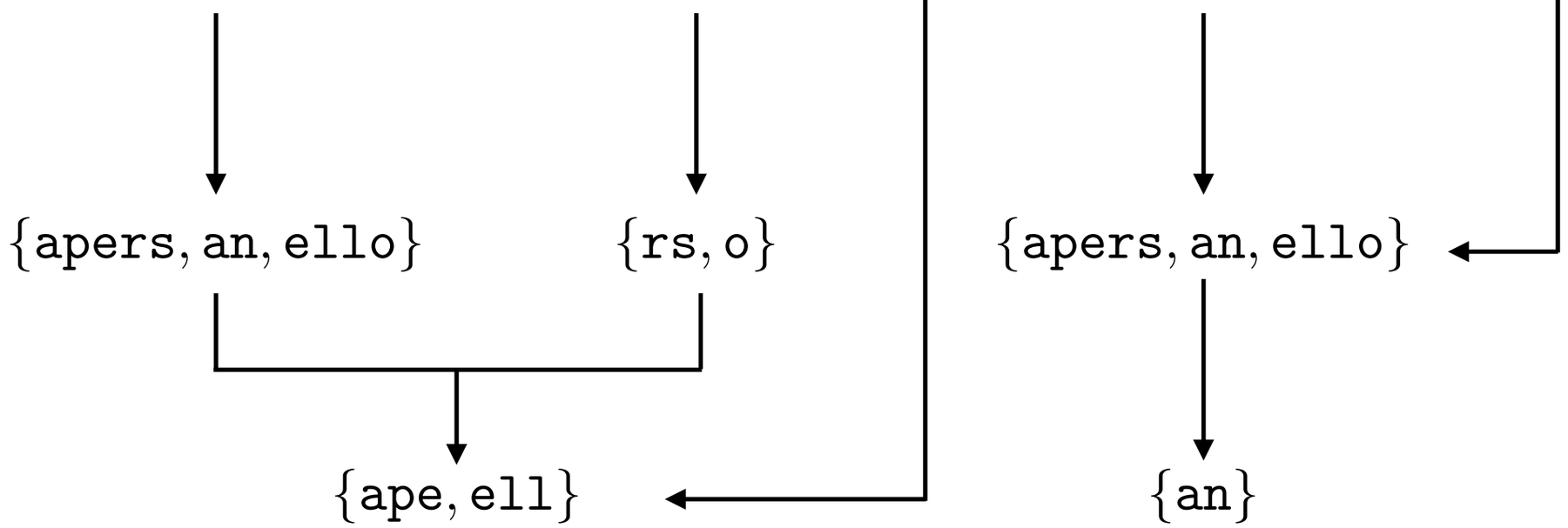
Let's consider $L = \{\texttt{papers}, \texttt{lan}, \texttt{hello}\}$

Substrings from 1 to 4: $\{\texttt{ape}, \texttt{ell}, \texttt{an}\}$

Substring

We first define substring on automata between two indexes i and j

$$\texttt{substr}(A, i, j) = \mathcal{RQ}(\mathcal{SU}(A, i), \mathcal{SU}(A, j)) \cap \Sigma^{j-i} \cup \mathcal{SU}(A, i)) \cap \Sigma^{<j-i}$$



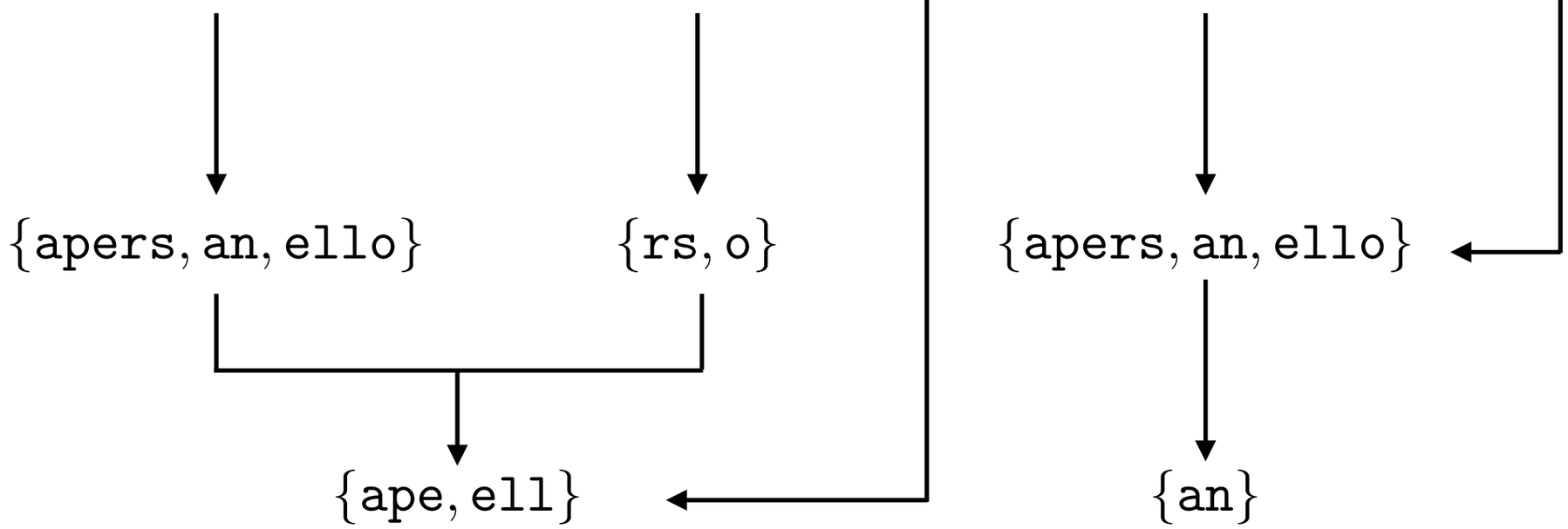
Let's consider $L = \{\texttt{papers}, \texttt{lan}, \texttt{hello}\}$

Substrings from 1 to 4: $\{\texttt{ape}, \texttt{ell}, \texttt{an}\}$

Substring

We first define substring on automata between two indexes i and j

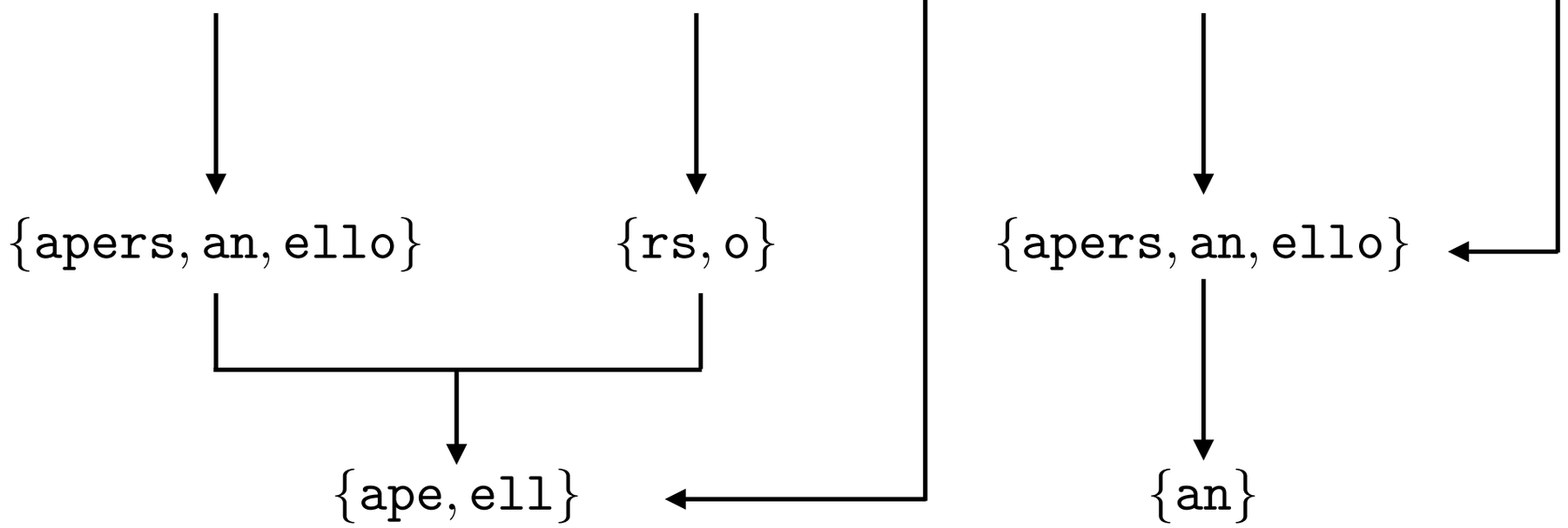
$$\texttt{substr}(A, i, j) = \mathcal{RQ}(\mathcal{SU}(A, i), \mathcal{SU}(A, j)) \cap \Sigma^{j-i} \cup \mathcal{SU}(A, i)) \cap \Sigma^{<j-i}$$



Substring

We first define substring on automata between two indexes i and j

$$\texttt{substr}(A, i, j) = \mathcal{RQ}(\mathcal{SU}(A, i), \mathcal{SU}(A, j)) \cap \Sigma^{j-i} \cup \mathcal{SU}(A, i)) \cap \Sigma^{<j-i}$$



Substrings of L $= \{\texttt{a}^n \mid n \in \mathbb{N}\}$ from 1 to 4: $\{\epsilon, \texttt{a}, \texttt{aa}, \texttt{aaa}\}$

Substring

Given an automaton A and two intervals [i,j] and [l,k] we have defined the full abstract semantics of substring.

$$\texttt{substr}^\sharp(A, [i,j], [l,k])$$

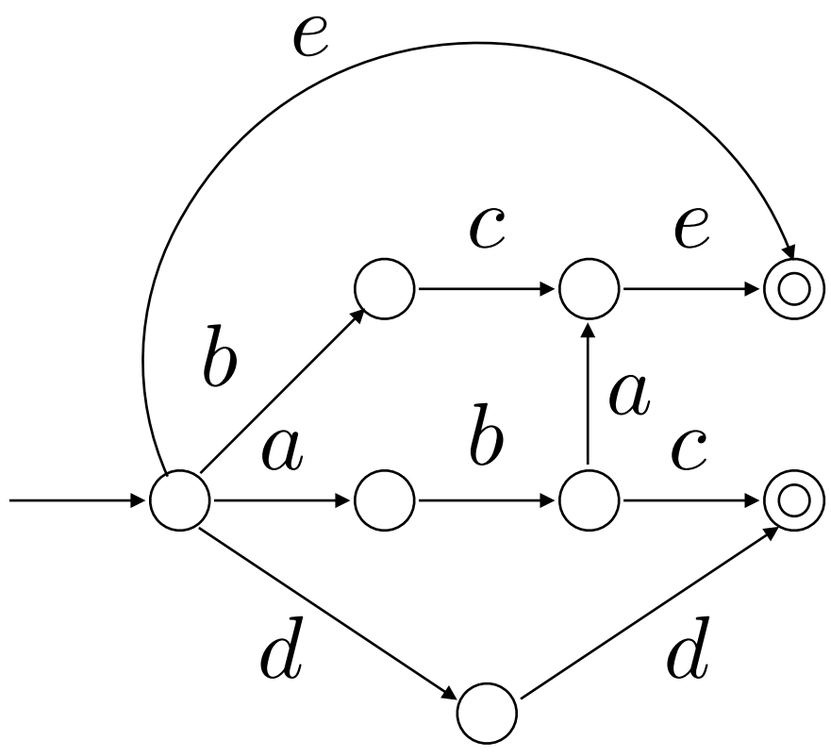
$\mathsf{SS}^\sharp(\mathtt{A}, [i,j], [l,k])$ $i, j \leq l \ (i \leq k)$	$l, k \in \mathbb{Z}$	$l = -\infty,\ k \in \mathbb{Z}$	$l \in \mathbb{Z},\ k = +\infty$	$l = -\infty,\ k = +\infty$
$i, j \in \mathbb{Z}$	$\mathsf{SS}(\mathtt{A}, [i,j], [l,k])$	$\mathsf{SS}^\sharp(\mathtt{A}, [i,j], [0,k])$	$\mathsf{SS}^{\rightarrow}(\mathtt{A}, [i,j], l)$	$\mathsf{SS}^\sharp(\mathtt{A}, [i,j], [0,+\infty])$
$i = -\infty,\ j \in \mathbb{Z}$	$\mathsf{SS}^\sharp(\mathtt{A}, [0,j], [l,k])$	$\mathsf{SS}^\sharp(\mathtt{A}, [0,j], [0,k])$	$\mathsf{SS}^\sharp(\mathtt{A}, [0,j], [l,+\infty])$	$\mathsf{SS}^\sharp(\mathtt{A}, [0,j], [0,+\infty])$
$i \in \mathbb{Z},\ j = +\infty$	$\mathsf{SS}^\sharp(\mathtt{A}, [l,k], [k,+\infty])$ $\sqcup \mathsf{SS}^\sharp(\mathtt{A}, [i,k], [l,k])$	$\mathsf{SS}^\sharp(\mathtt{A}, [i,+\infty], [0,k])$	$\mathsf{SS}^{\rightarrow}(\mathtt{A}, [i,l], l) \sqcup \mathsf{SS}^{\leftrightarrow}(\mathtt{A}, l)$	$\mathsf{SS}^\sharp(\mathtt{A}, [i,+\infty], [0,+\infty])$
$i = -\infty,\ j = +\infty$	$\mathsf{SS}^\sharp(A, [0,+\infty], [l,k])$	$\mathsf{SS}^\sharp(\mathtt{A}, [0,+\infty], [0,k])$	$\mathsf{SS}^\sharp(\mathtt{A}, [0,+\infty], [l,+\infty])$	$\mathsf{FA}(\mathtt{A})$

Table 1: Definition of $\mathsf{SS}^\sharp$ when $i, j \leq l \ (i \leq k)$

Similarly, we can define the abstract semantics of charAt

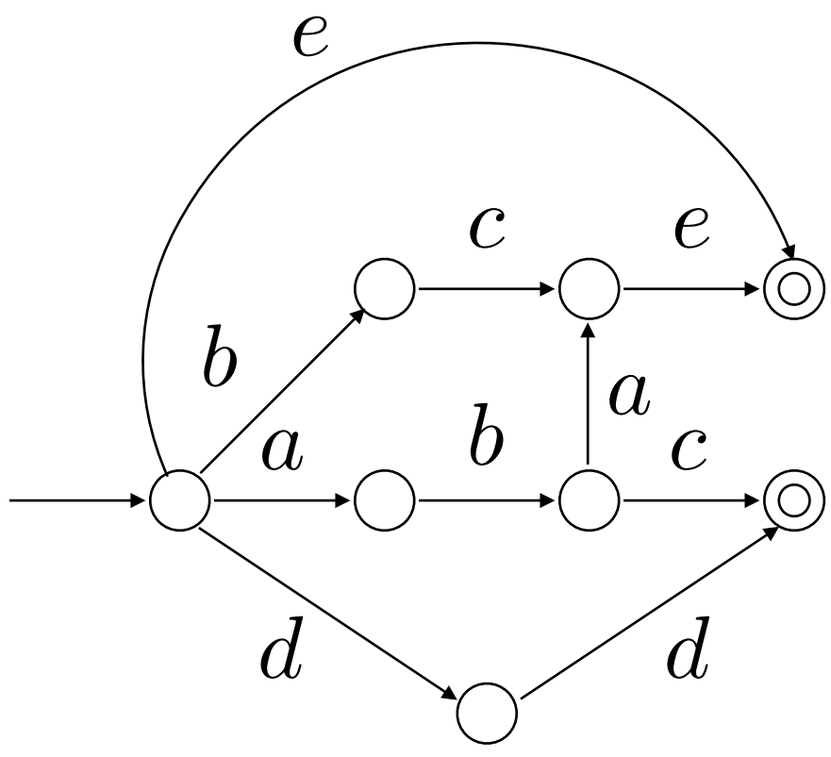
Length

We collect the sizes of the minimum and the maximum paths from the initial state and each final state, then we abstract the result to the interval abstract domain.



Length

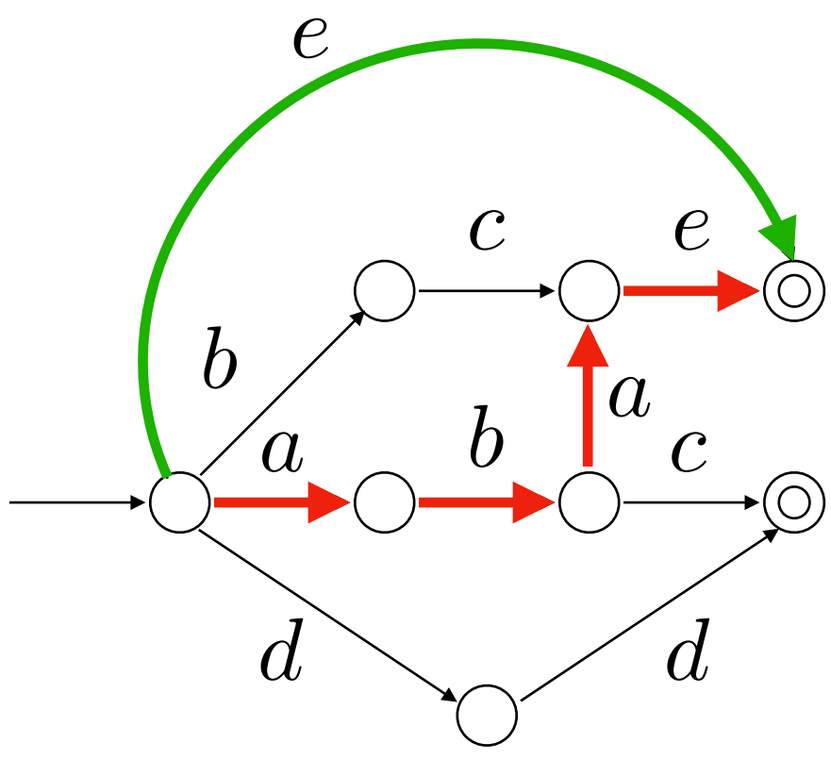
We collect the sizes of the minimum and the maximum paths from the initial state and each final state, then we abstract the result to the interval abstract domain.



Lengths:

Length

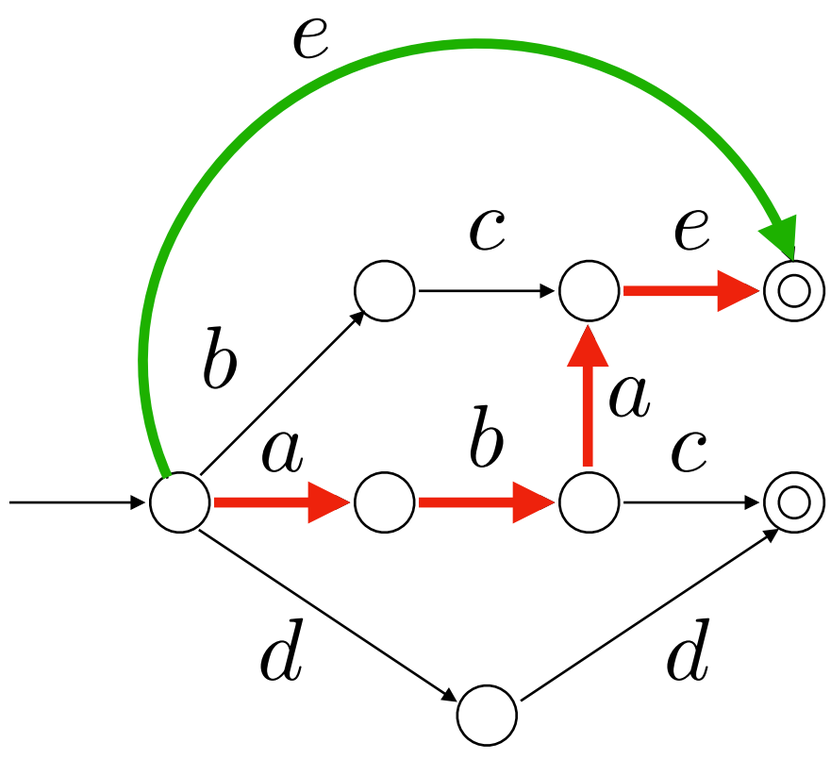
We collect the sizes of the minimum and the maximum paths from the initial state and each final state, then we abstract the result to the interval abstract domain.



Lengths:

Length

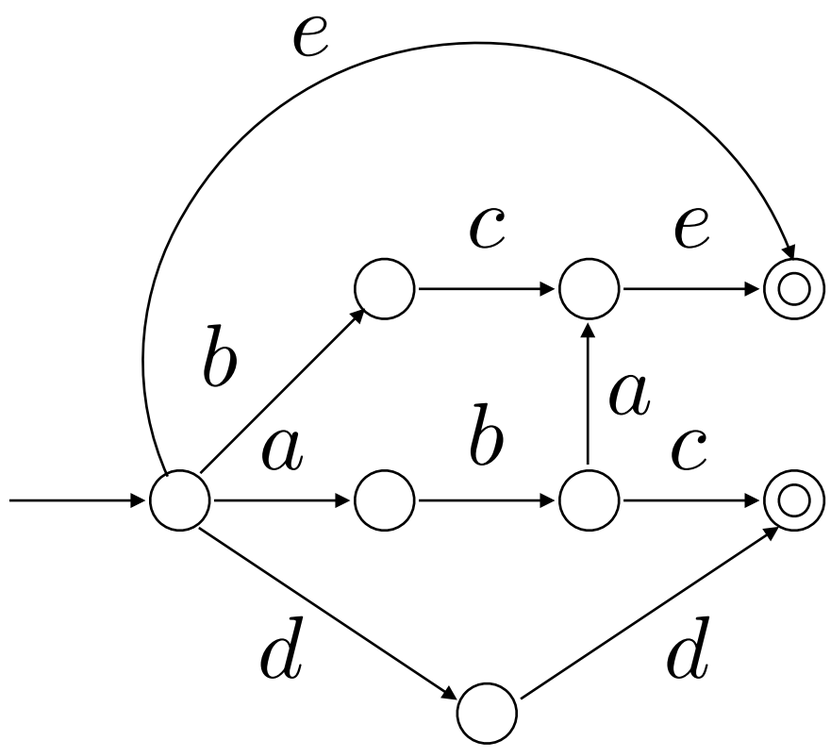
We collect the sizes of the minimum and the maximum paths from the initial state and each final state, then we abstract the result to the interval abstract domain.



Lengths: 1, 4,

Length

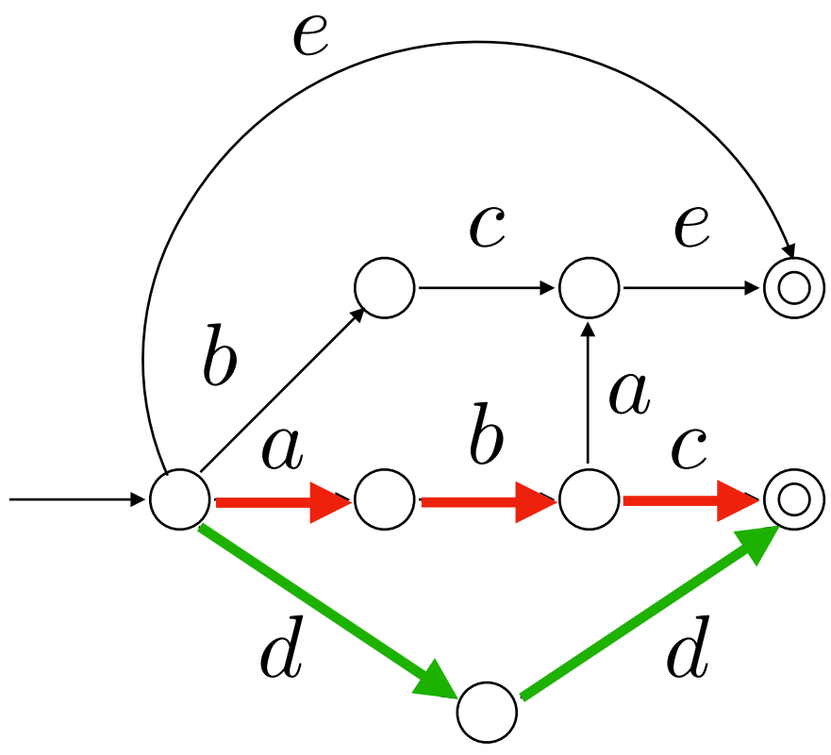
We collect the sizes of the minimum and the maximum paths from the initial state and each final state, then we abstract the result to the interval abstract domain.



Lengths: 1, 4,

Length

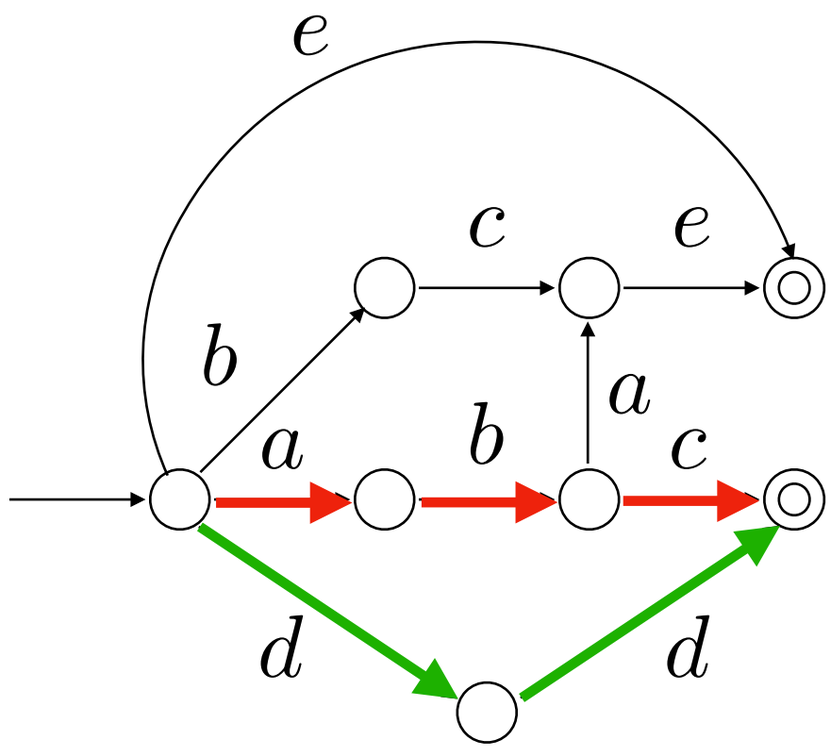
We collect the sizes of the minimum and the maximum paths from the initial state and each final state, then we abstract the result to the interval abstract domain.



Lengths: 1, 4,

Length

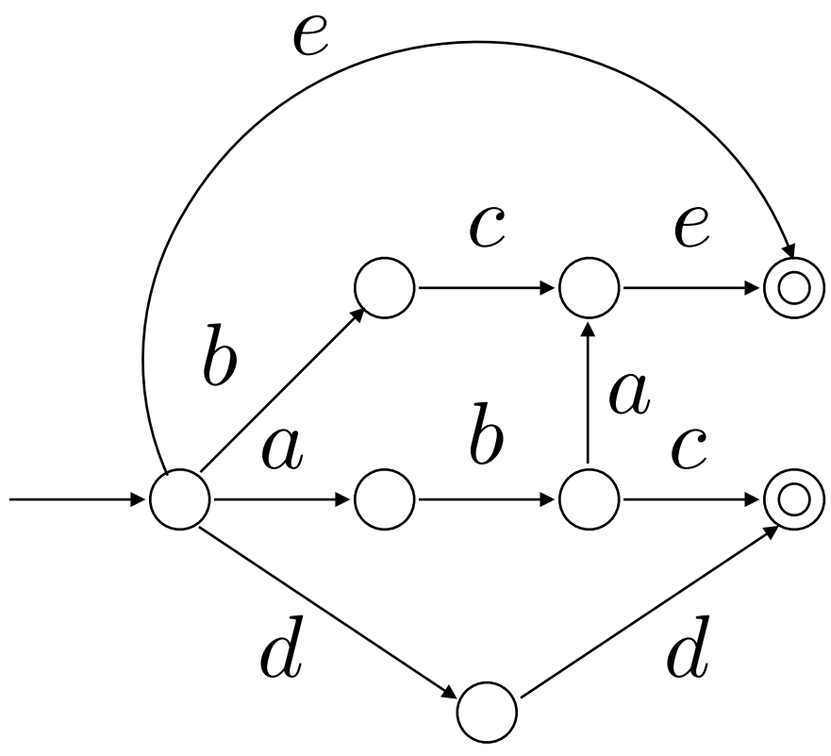
We collect the sizes of the minimum and the maximum paths from the initial state and each final state, then we abstract the result to the interval abstract domain.



Lengths: 1, 4, 2, 3

Length

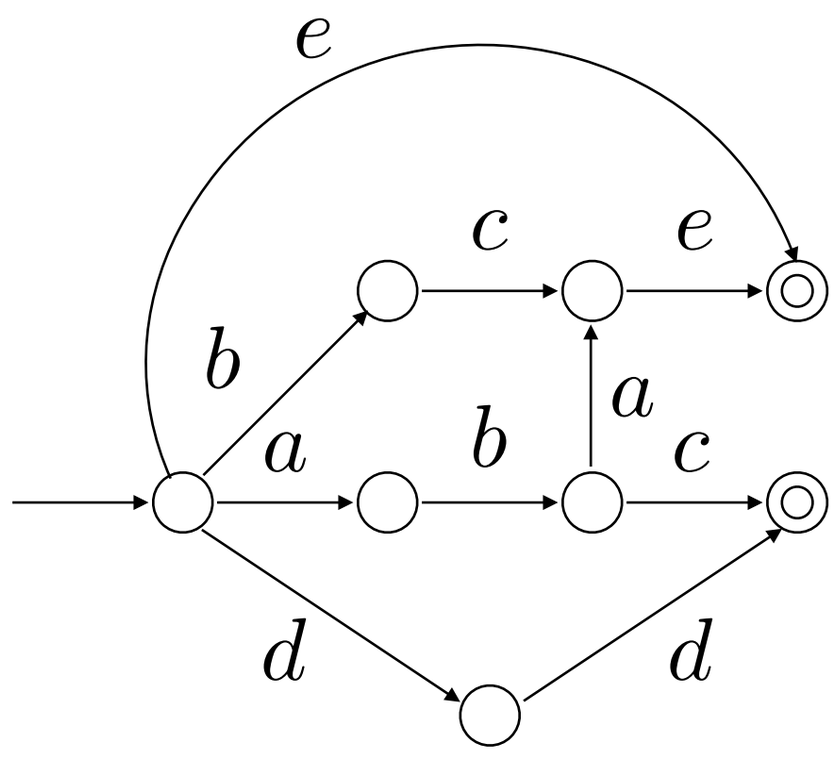
We collect the sizes of the minimum and the maximum paths from the initial state and each final state, then we abstract the result to the interval abstract domain.



Lengths: 1, 4, 2, 3

Length

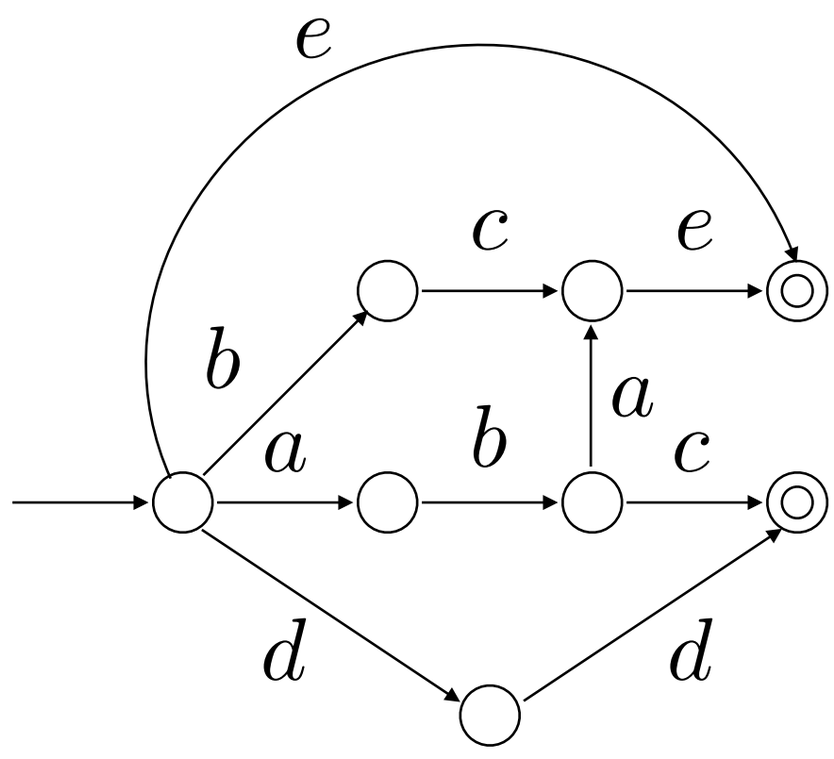
We collect the sizes of the minimum and the maximum paths from the initial state and each final state, then we abstract the result to the interval abstract domain.



Lengths: 1, 4, 2, 3 —Interval abstraction→

Length

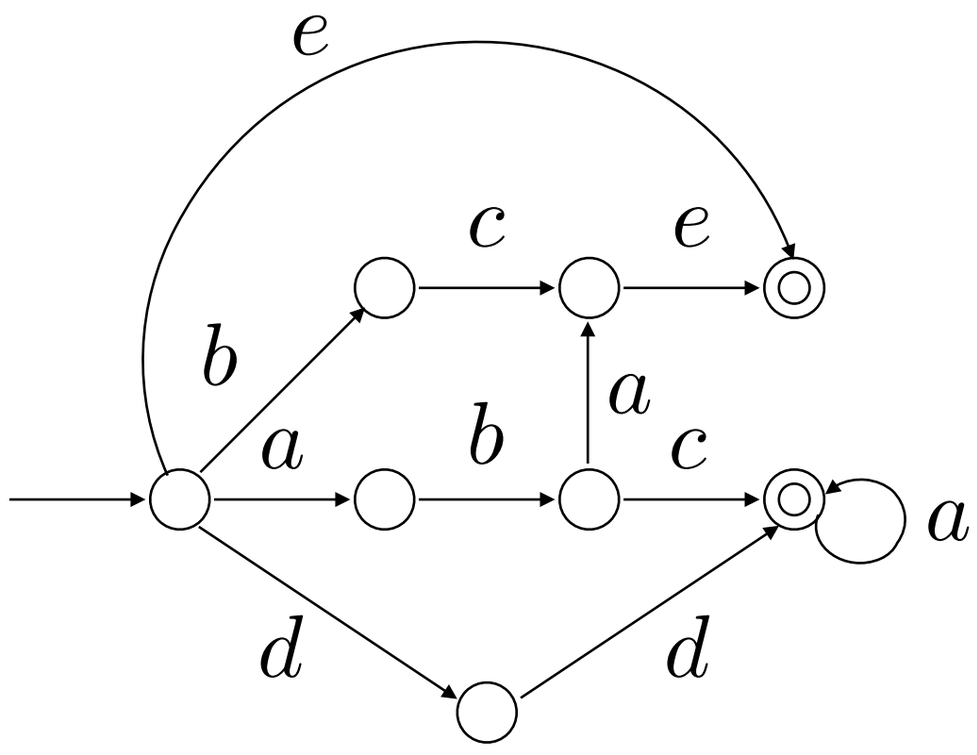
We collect the sizes of the minimum and the maximum paths from the initial state and each final state, then we abstract the result to the interval abstract domain.



Lengths: 1, 4, 2, 3 —Interval abstraction→ [1, 4]

Length

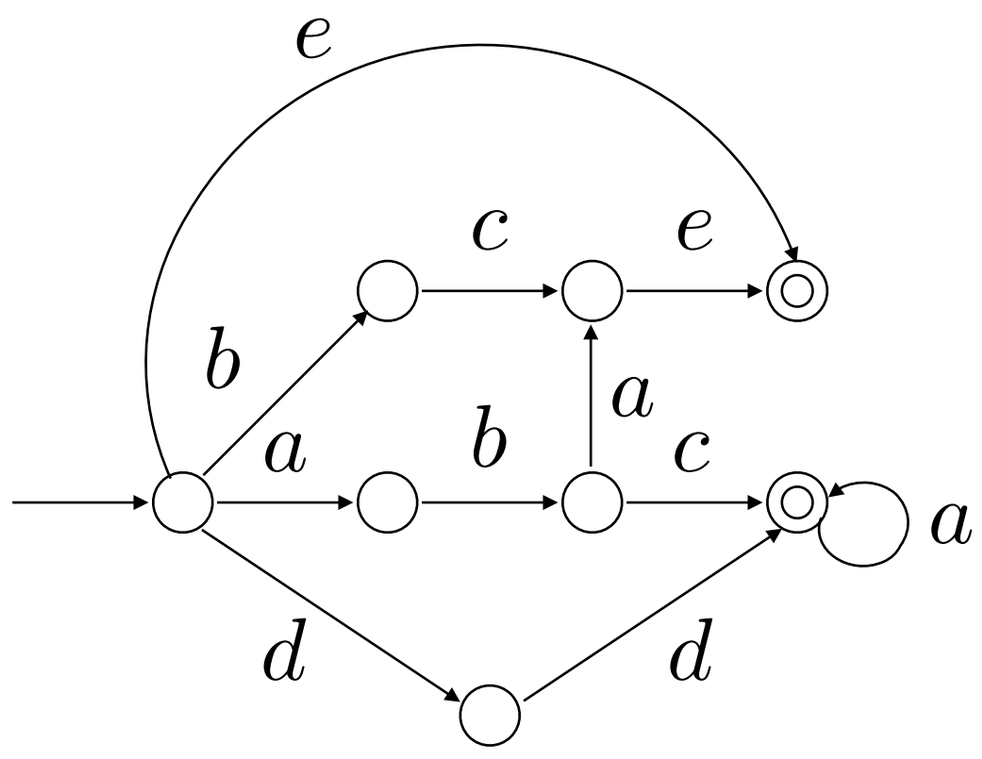
We collect the sizes of the minimum and the maximum paths from the initial state and each final state, then we abstract the result to the interval abstract domain.



Lengths: 1, 4, 2, 3 —Interval abstraction→

Length

We collect the sizes of the minimum and the maximum paths from the initial state and each final state, then we abstract the result to the interval abstract domain.

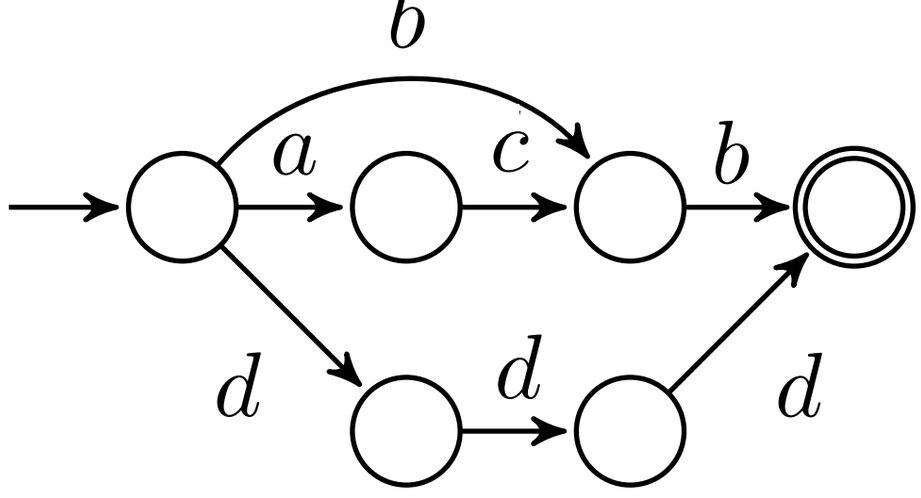


Lengths: 1, 4, 2, 3 —Interval abstraction→ [1, +∞]

IndexOf

$$\texttt{indexOf}^\sharp(A, A')$$

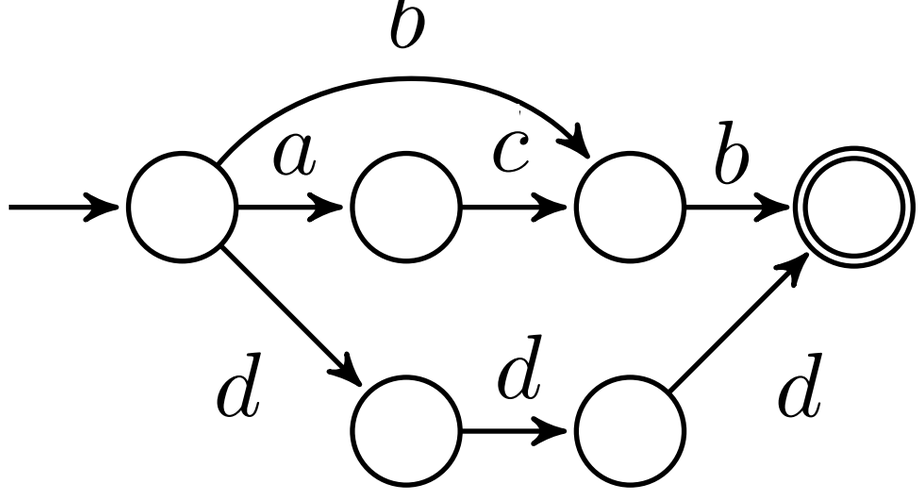
For each state q of A, we check if it is possible to recognize from q strings of A'. If so, we compute the length of the maximum path from q0 to q.



IndexOf

$$\texttt{indexOf}^\sharp(A, A')$$

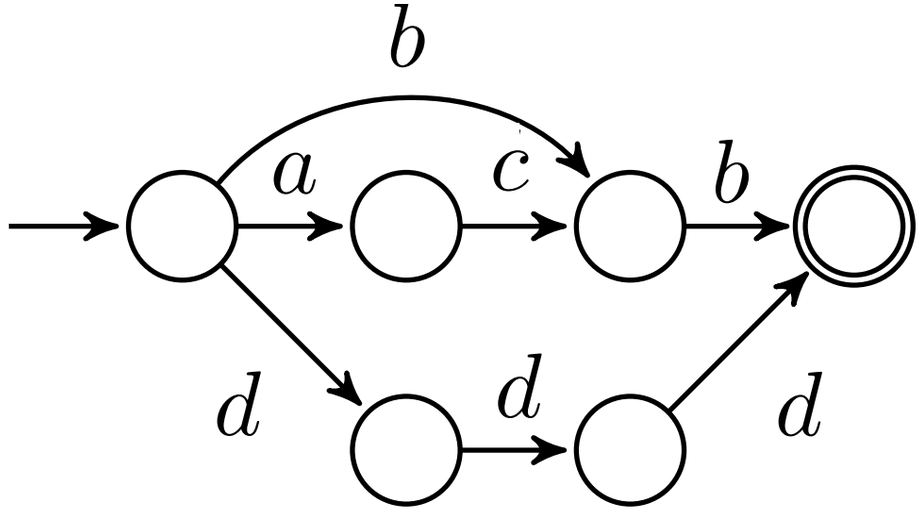
For each state q of A, we check if it is possible to recognize from q strings of A'. If so, we compute the length of the maximum path from q0 to q. Let A' be the automaton recognizing the string "b".



IndexOf

$$\texttt{indexOf}^\sharp(A, A')$$

For each state q of A, we check if it is possible to recognize from q strings of A'. If so, we compute the length of the maximum path from q0 to q. Let A' be the automaton recognizing the string "b".

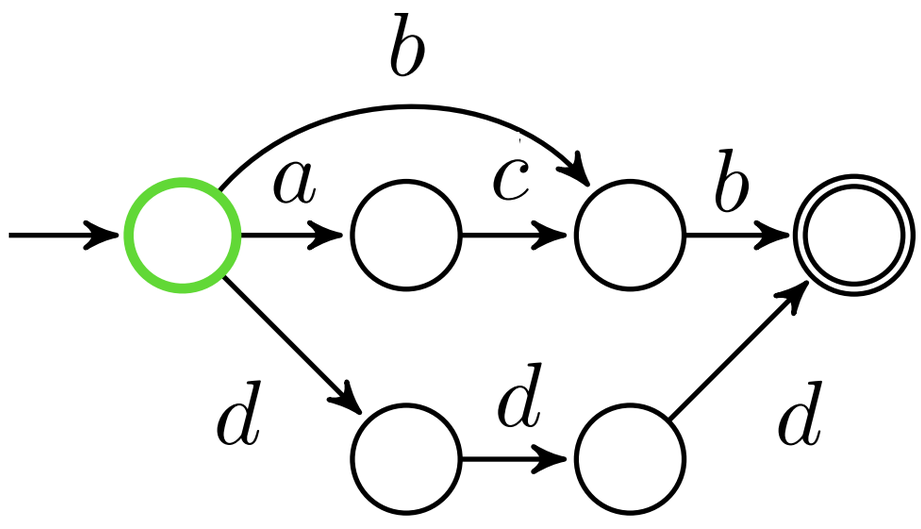


Indexes:

IndexOf

$$\texttt{indexOf}^\sharp(A, A')$$

For each state q of A, we check if it is possible to recognize from q strings of A'. If so, we compute the length of the maximum path from q0 to q. Let A' be the automaton recognizing the string "b".

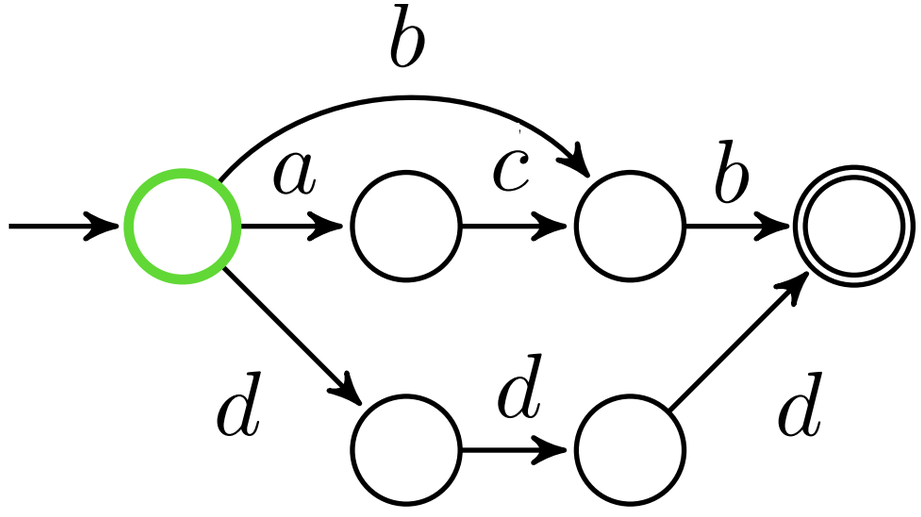


Indexes:

IndexOf

$$\texttt{indexOf}^\sharp(A, A')$$

For each state q of A, we check if it is possible to recognize from q strings of A'. If so, we compute the length of the maximum path from q0 to q. Let A' be the automaton recognizing the string "b".

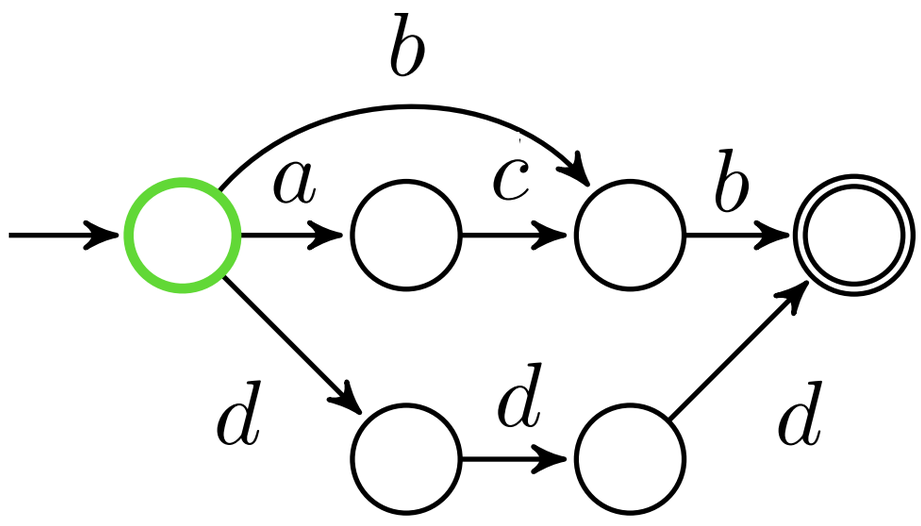


Indexes: 0,

IndexOf

$$\texttt{indexOf}^\sharp(A, A')$$

For each state q of A, we check if it is possible to recognize from q strings of A'. If so, we compute the length of the maximum path from q0 to q. Let A' be the automaton recognizing the string "b".

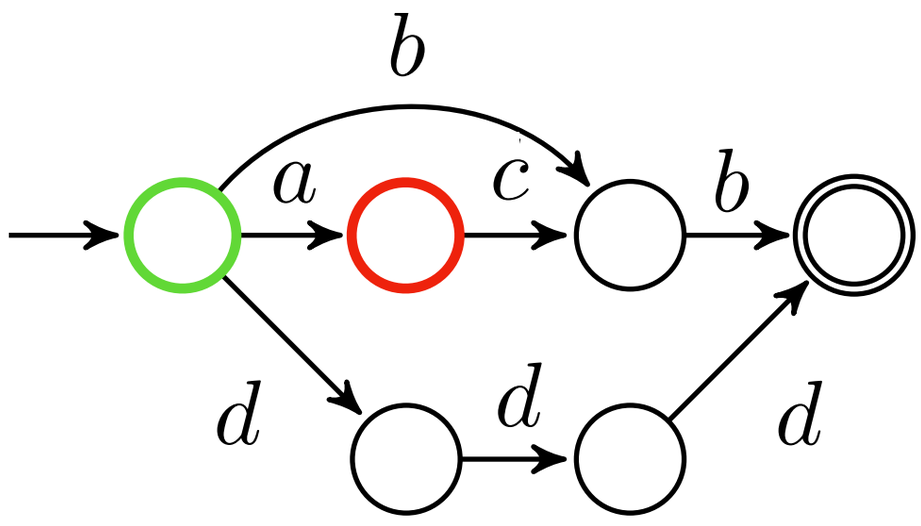


Indexes: 0, -1,

IndexOf

$$\texttt{indexOf}^\sharp(A, A')$$

For each state q of A, we check if it is possible to recognize from q strings of A'. If so, we compute the length of the maximum path from q0 to q. Let A' be the automaton recognizing the string "b".

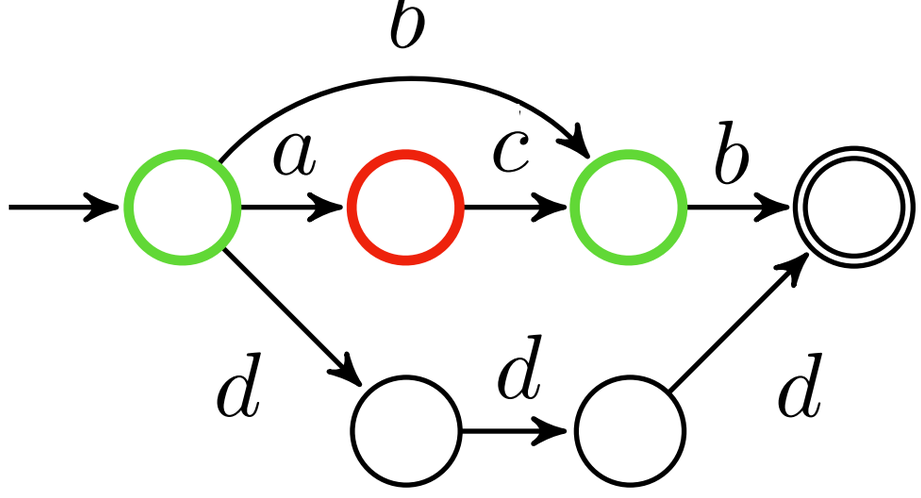


Indexes: 0, -1,

IndexOf

$$\texttt{indexOf}^\sharp(A, A')$$

For each state q of A, we check if it is possible to recognize from q strings of A'. If so, we compute the length of the maximum path from q0 to q. Let A' be the automaton recognizing the string "b".

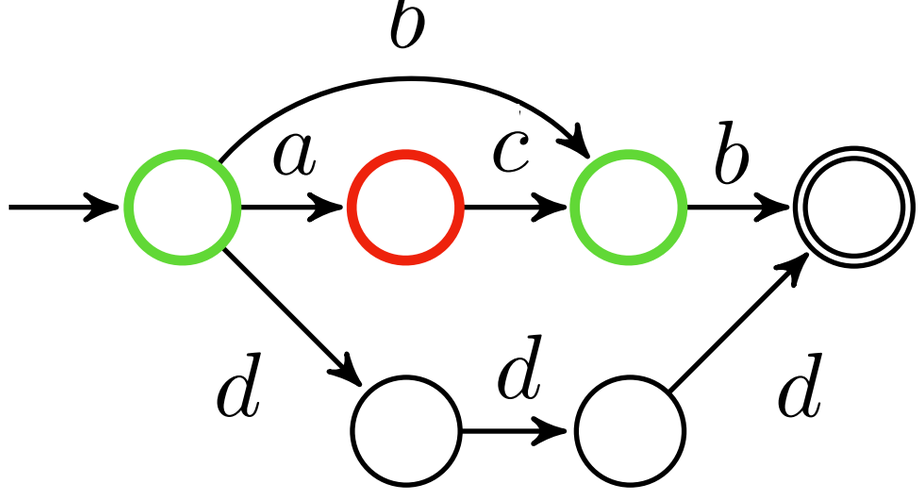


Indexes: 0, -1,

IndexOf

$$\texttt{indexOf}^\sharp(A, A')$$

For each state q of A, we check if it is possible to recognize from q strings of A'. If so, we compute the length of the maximum path from q0 to q. Let A' be the automaton recognizing the string "b".

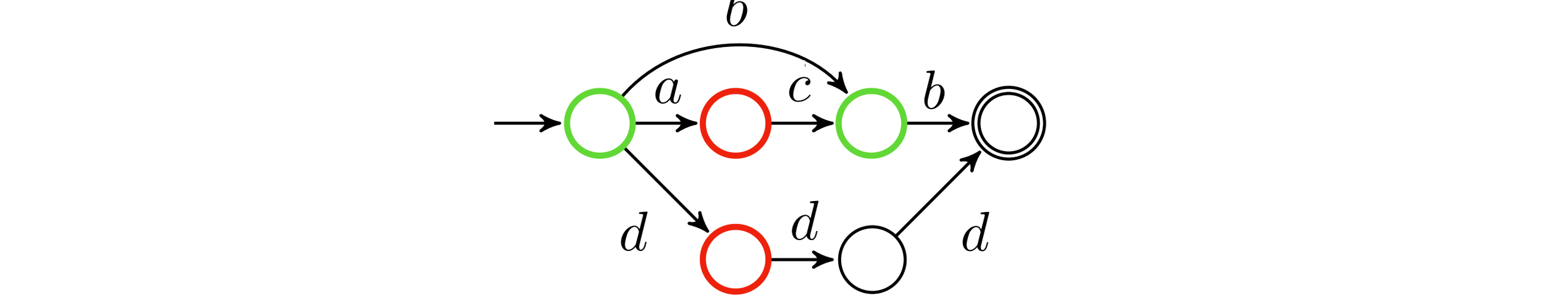


Indexes: 0, -1, 2

IndexOf

$$\texttt{indexOf}^\sharp(A, A')$$

For each state q of A, we check if it is possible to recognize from q strings of A'. If so, we compute the length of the maximum path from q0 to q. Let A' be the automaton recognizing the string "b".

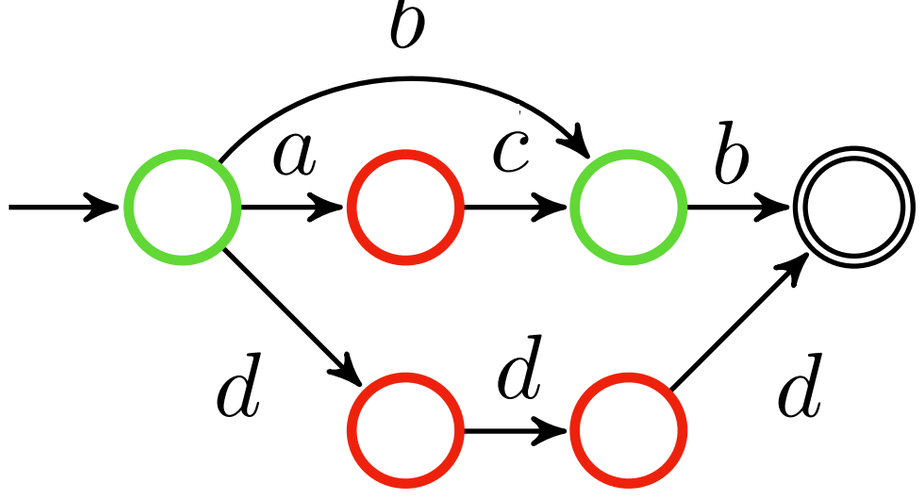


Indexes: 0, -1, 2

IndexOf

$$\texttt{indexOf}^\sharp(A, A')$$

For each state q of A, we check if it is possible to recognize from q strings of A'. If so, we compute the length of the maximum path from q0 to q. Let A' be the automaton recognizing the string "b".

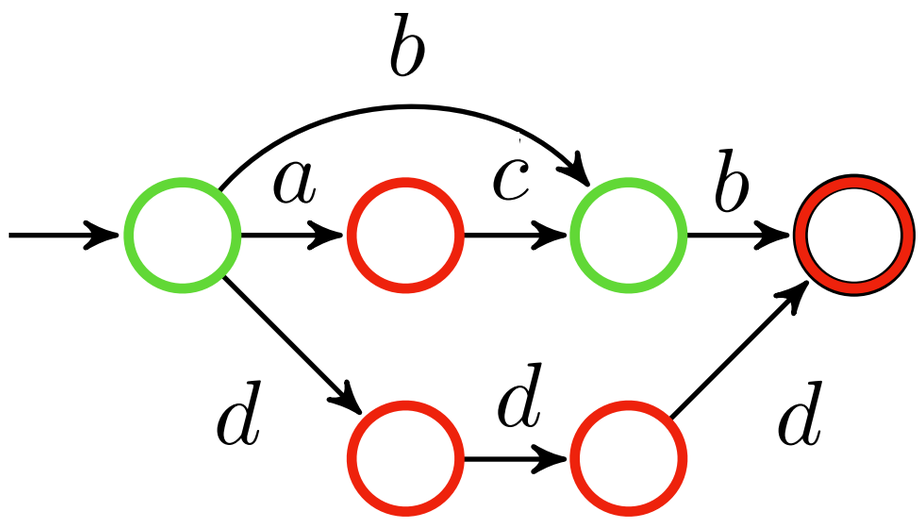


Indexes: 0, -1, 2

IndexOf

$$\texttt{indexOf}^\sharp(A, A')$$

For each state q of A, we check if it is possible to recognize from q strings of A'. If so, we compute the length of the maximum path from q0 to q. Let A' be the automaton recognizing the string "b".

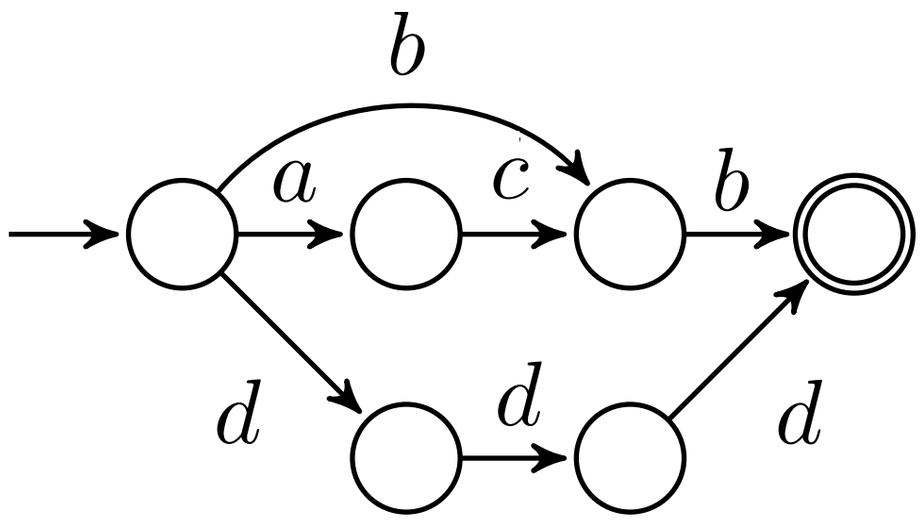


Indexes: 0, -1, 2

IndexOf

$$\texttt{indexOf}^\sharp(A, A')$$

For each state q of A, we check if it is possible to recognize from q strings of A'. If so, we compute the length of the maximum path from q0 to q. Let A' be the automaton recognizing the string "b".

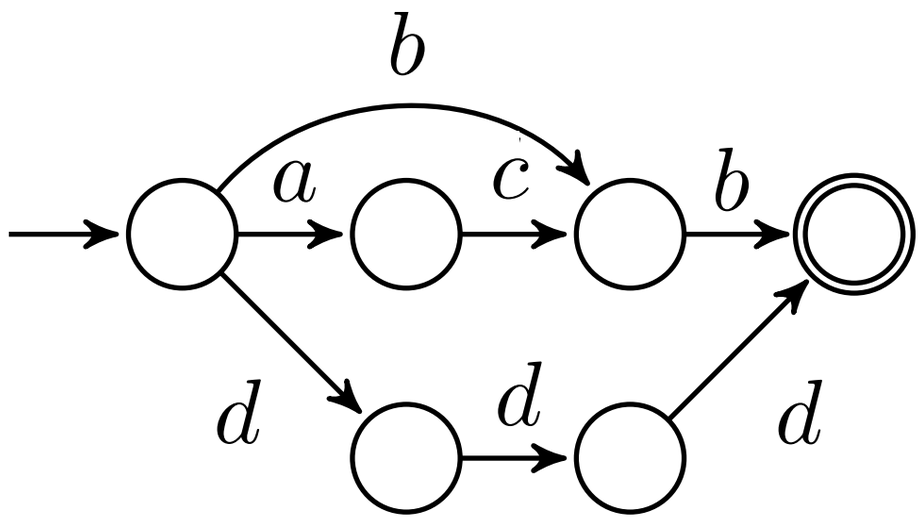


Indexes: 0, -1, 2

IndexOf

$$\texttt{indexOf}^{\sharp}(A, A')$$

For each state q of A, we check if it is possible to recognize from q strings of A'. If so, we compute the length of the maximum path from q0 to q. Let A' be the automaton recognizing the string "b".

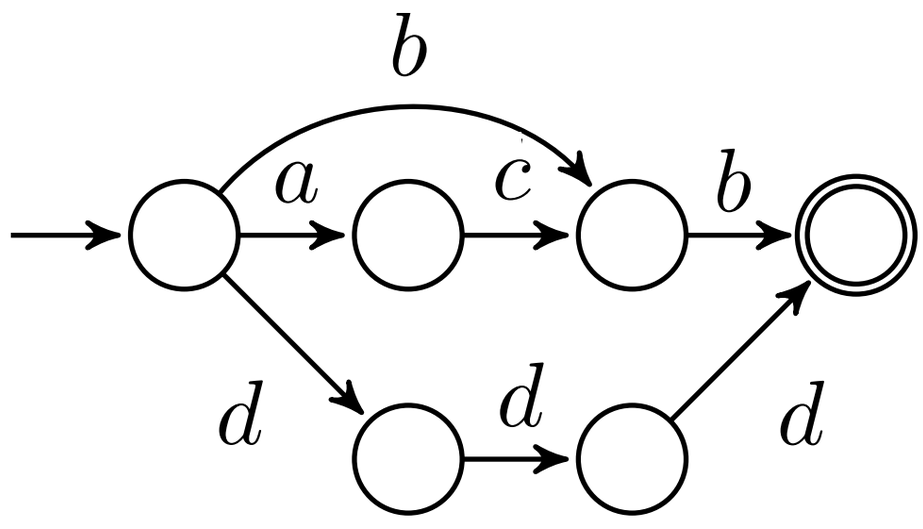


Indexes: 0, -1, 2 —Interval abstraction→

IndexOf

$$\texttt{indexOf}^{\sharp}(A, A')$$

For each state q of A, we check if it is possible to recognize from q strings of A'. If so, we compute the length of the maximum path from q0 to q. Let A' be the automaton recognizing the string "b".

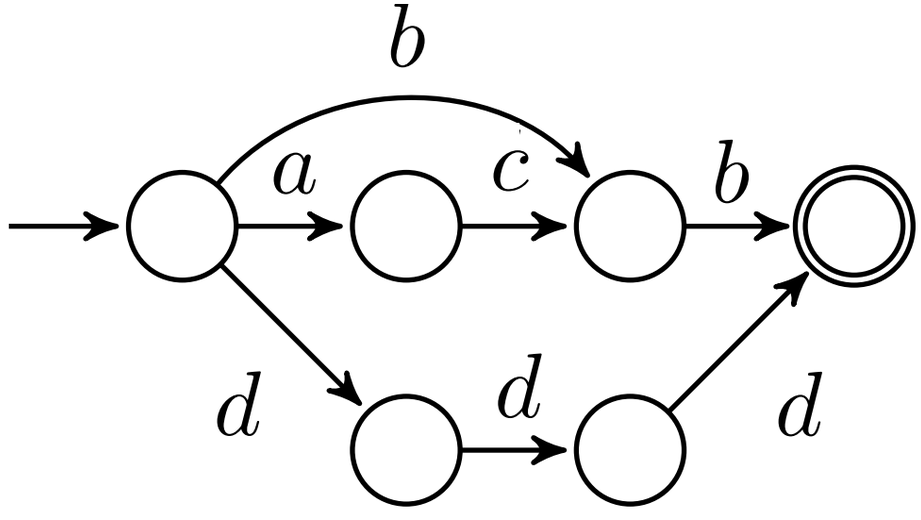


Indexes: 0, -1, 2 —Interval abstraction→ [-1,2]

IndexOf

$$\texttt{indexOf}^\sharp(A, A')$$

For each state q of A, we check if it is possible to recognize from q strings of A'. If so, we compute the length of the maximum path from q0 to q. Let A' be the automaton recognizing the string "b".



Indexes: 0, -1, 2 —Interval abstraction→ [-1,2]

The case when the automaton has cycles is treated similarly to the length abstract semantics.

Example

```
vd, ac, la = "";
v = "wZsZ"; m = "AYcYtYiYvYeYXY";
tt = "AObyaSZjectB";
l = "WYSYcYrYiYpYtY.YSYhYeYlYlY";

while (i+=2 < v.length)
  vd = vd + v.charAt(i);

while (j+=2 < m.length)
  ac = ac + m.charAt(j);

ac += tt.substring(tt.indexOf("O"), 3);
ac += tt.substring(tt.indexOf("j"), 11);

while (k+=2 < l.length)
  la = la + l.charAt(k);

d = vd + "=new " + ac + "(" + la + ")";
eval(d);
```

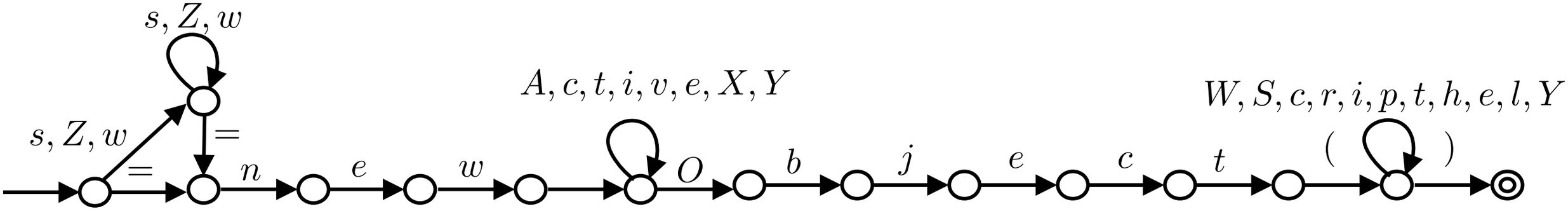
Example

```
vd, ac, la = "";
v = "wZsZ"; m = "AYcYtYiYvYeYXY";
tt = "AObyaSZjectB";
l = "WYSYcYrYiYpYtY.YSYhYeYlYlY";

while (i+=2 < v.length)
  vd = vd + v.charAt(i);

while (j+=2 < m.length)
  ac = ac + m.charAt(j);

ac += tt.substring(tt.indexOf("O"), 3);
ac += tt.substring(tt.indexOf("j"), 11);

while (k+=2 < l.length)
  la = la + l.charAt(k);

d = vd + "=new " + ac + "(" + la + ")";
eval(d);
```

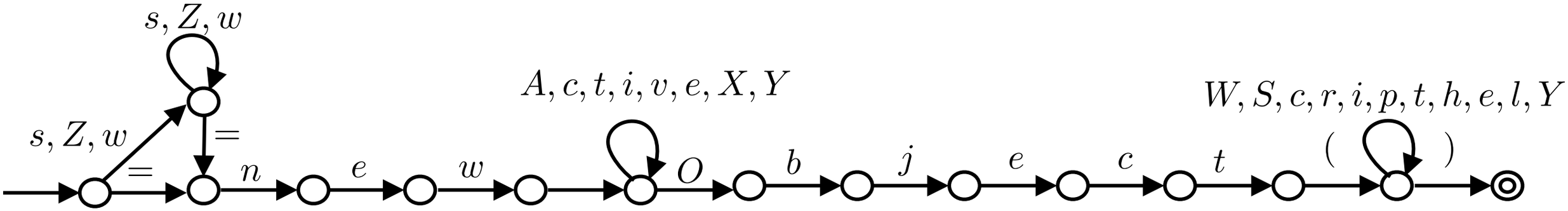
Which is the value of d before the eval execution?

Example

Example

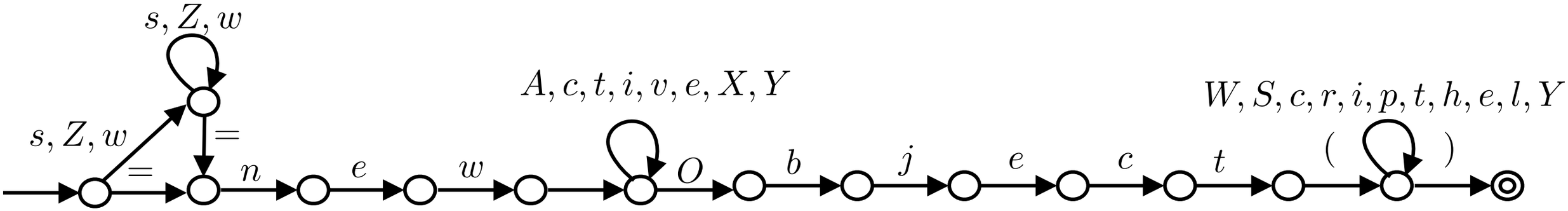


Example



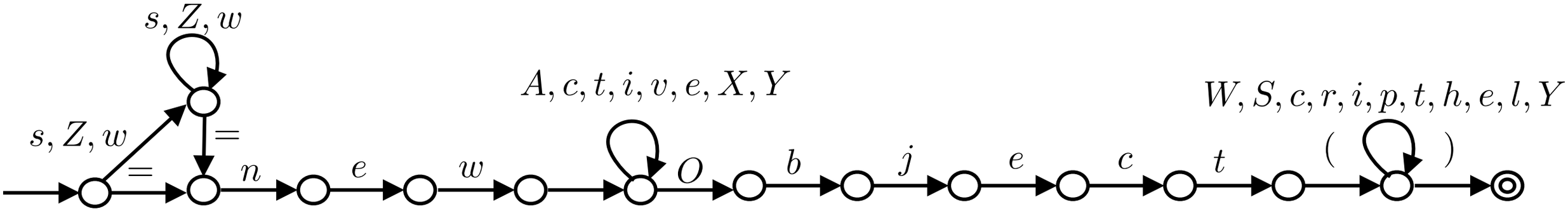
TAJS Don’t know

Example



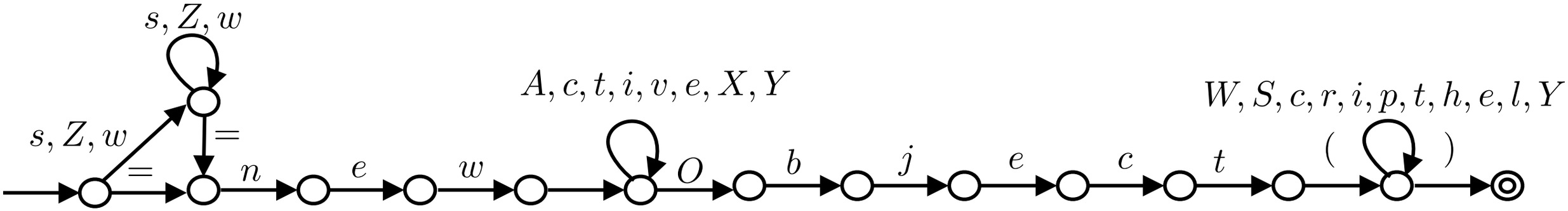
TAJS Don't know **SAFE** Don't know

Example



TAJS Don't know SAFE Don't know JSAI Don't know

Example

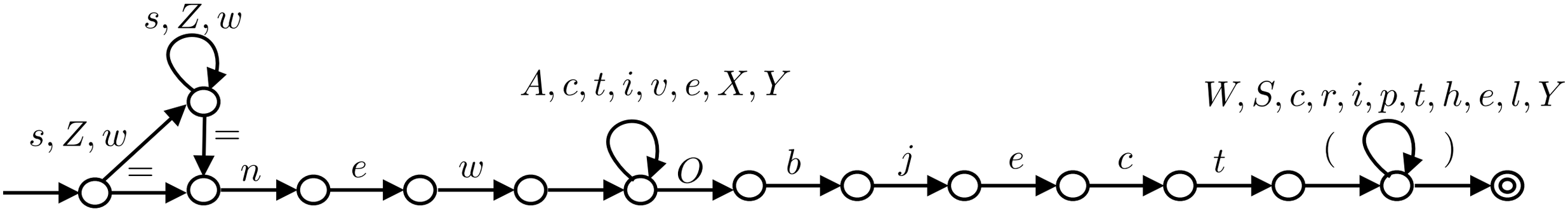


TAJS Don't know SAFE Don't know JSAI Don't know

- It adds new variable to the global scope

```
ws = ...
```

Example



TAJS Don't know SAFE Don't know JSAI Don't know

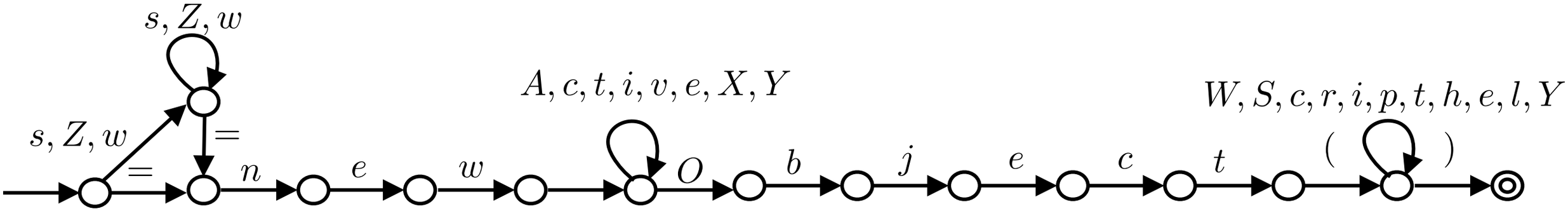
- It adds new variable to the global scope

```
ws = ...
```

- It may create an ActiveXObject and open a shell

```
ws = new ActiveXObject(Wshell.Script)
```

Example



TAJS Don't know SAFE Don't know JSAI Don't know

- It adds new variable to the global scope

```
ws = ...
```

- It may create an ActiveXObject and open a shell

```
ws = new ActiveXObject(Wshell.Script)
```

- No more eval calls occur

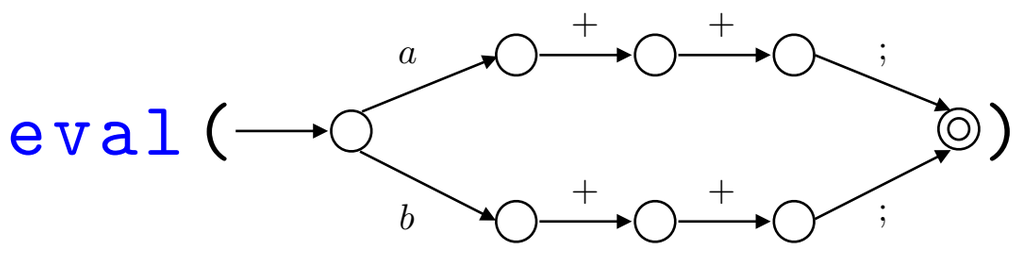
Future works

Future works

- Finite state automata can be useful for eval analysis

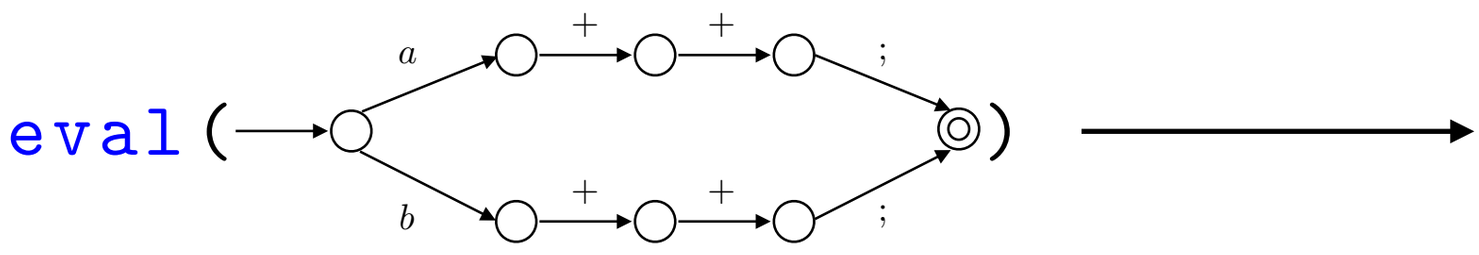
Future works

- Finite state automata can be useful for eval analysis



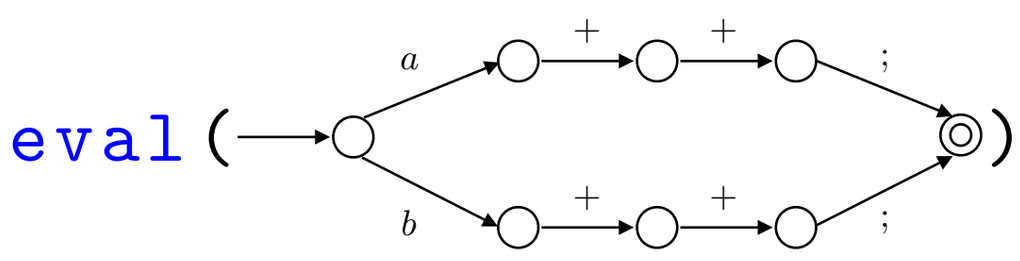
Future works

- Finite state automata can be useful for eval analysis



Future works

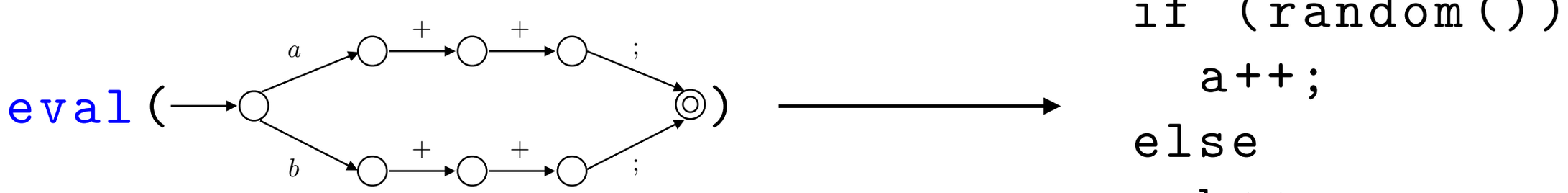
- Finite state automata can be useful for eval analysis




```
if (random())
  a++;
else
  b++;
```

Future works

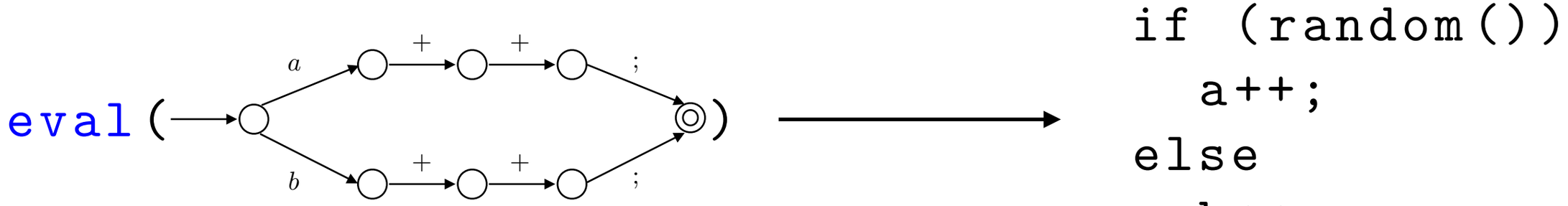
- Finite state automata can be useful for eval analysis



- Improvements on fix-point computations

Future works

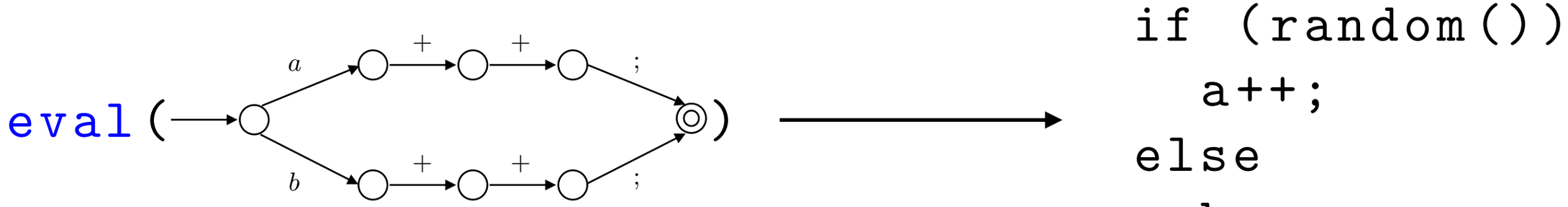
- Finite state automata can be useful for eval analysis



- Improvements on fix-point computations
  - Narrowing

Future works

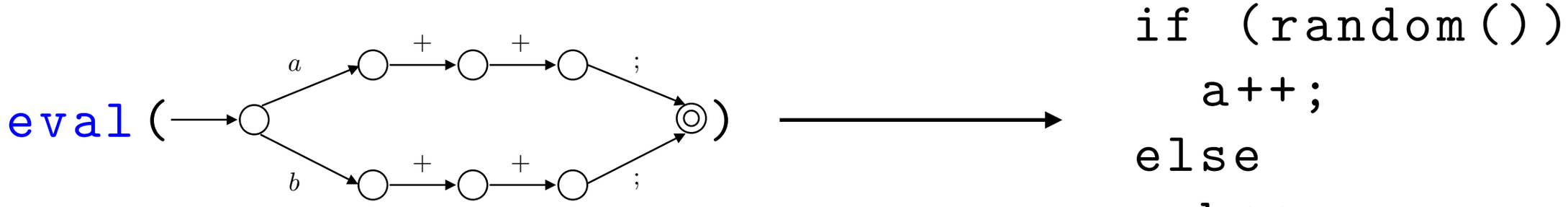
- Finite state automata can be useful for eval analysis



- Improvements on fix-point computations
  - Narrowing
  - Widening with threshold

Future works

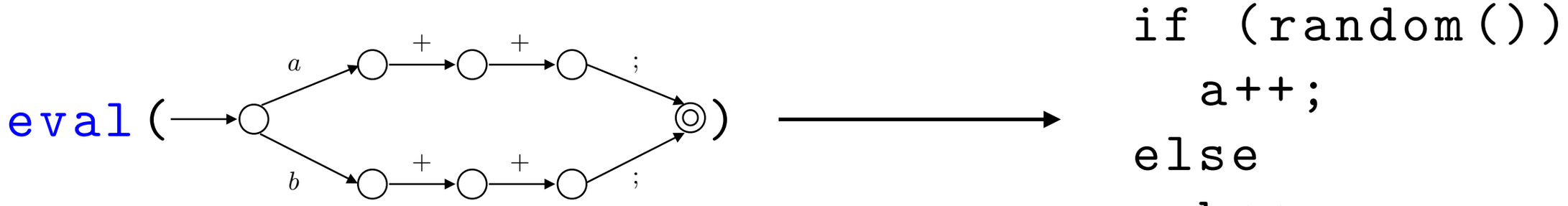
- Finite state automata can be useful for eval analysis



- Improvements on fix-point computations
  - Narrowing
  - Widening with threshold
  - Loop unrolling

Future works

- Finite state automata can be useful for eval analysis

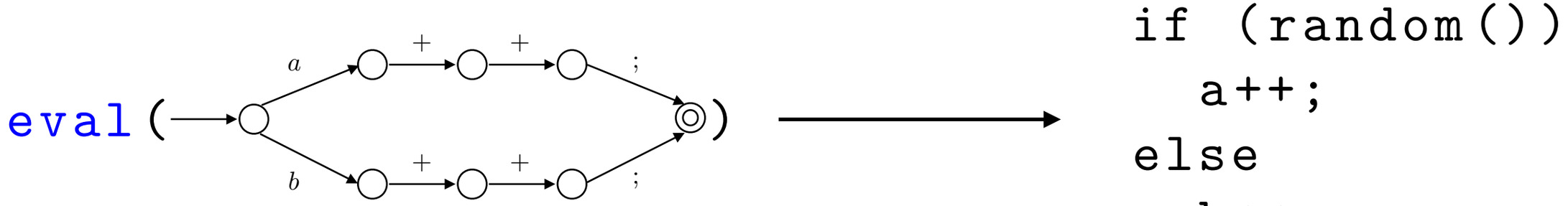


- Improvements on fix-point computations
  - Narrowing
  - Widening with threshold
  - Loop unrolling

- Cover the full JS String built-in object methods

Future works

- Finite state automata can be useful for eval analysis



- Improvements on fix-point computations
  - Narrowing
  - Widening with threshold
  - Loop unrolling

- Cover the full JS String built-in object methods

`replace` `slice` `startsWith` `repeat` `...`

Finite state automata implementation

https://github.com/SPY-Lab/fsa

Static analyzer

https://github.com/SPY-Lab/mu-js

Thank you for your attention!