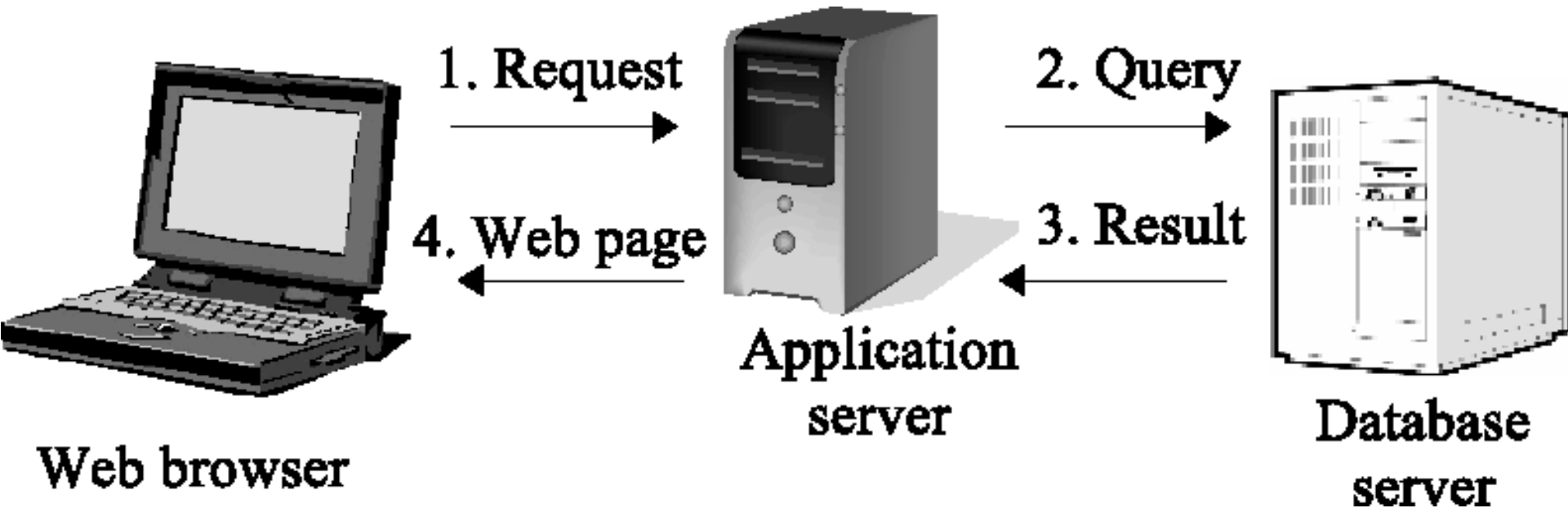
Generating String Inputs Using Constrained Symbolic Execution

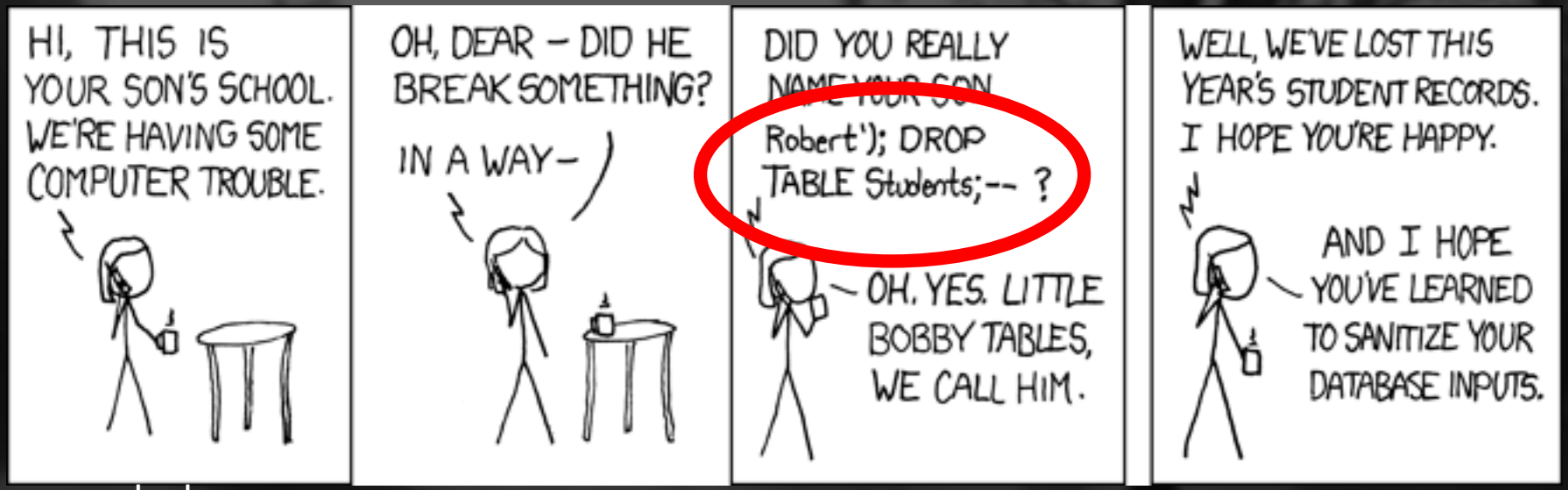
Pieter Hooimeijer

University of Virginia

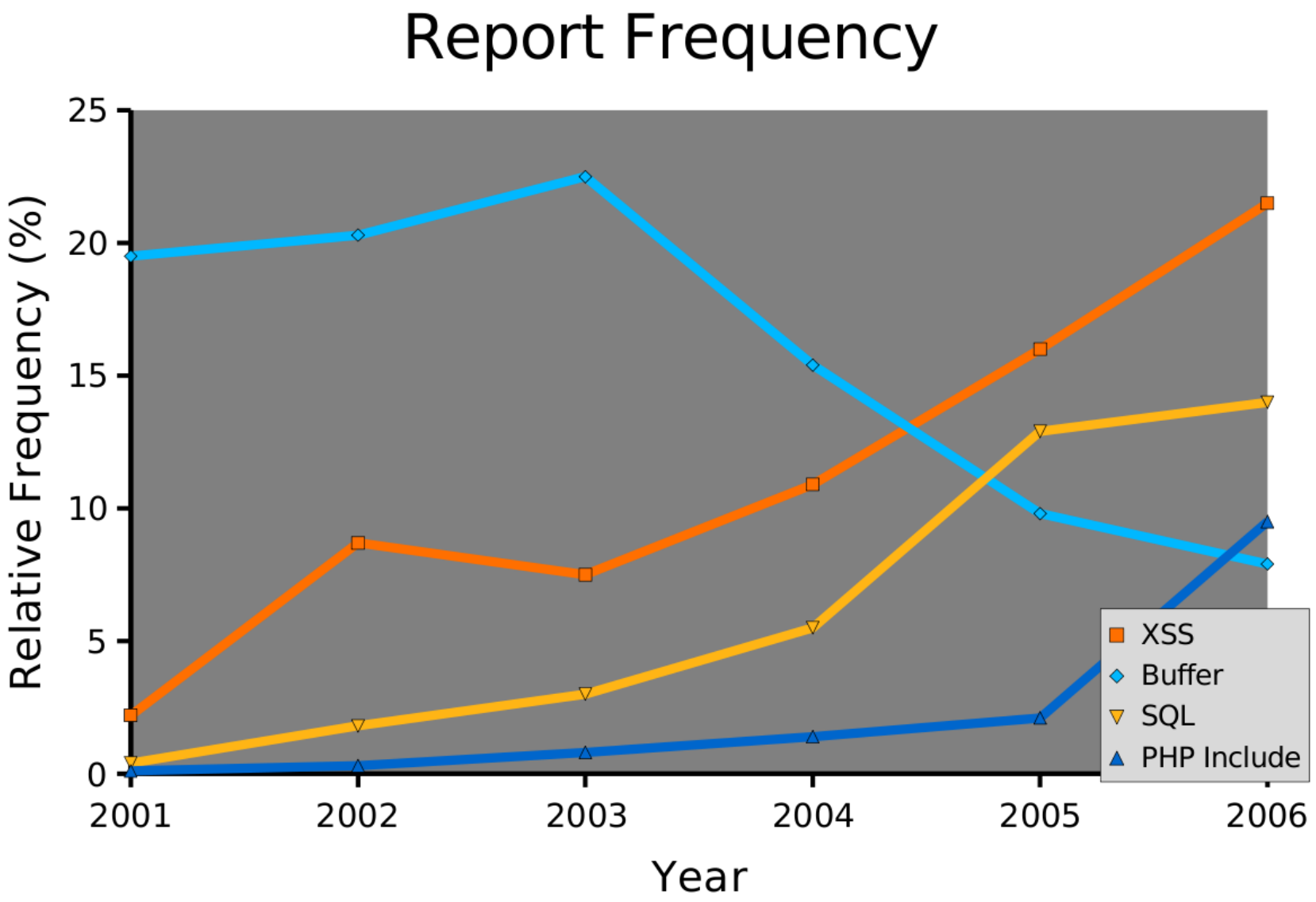
The Problem: SQL Injection



Motivation Montage

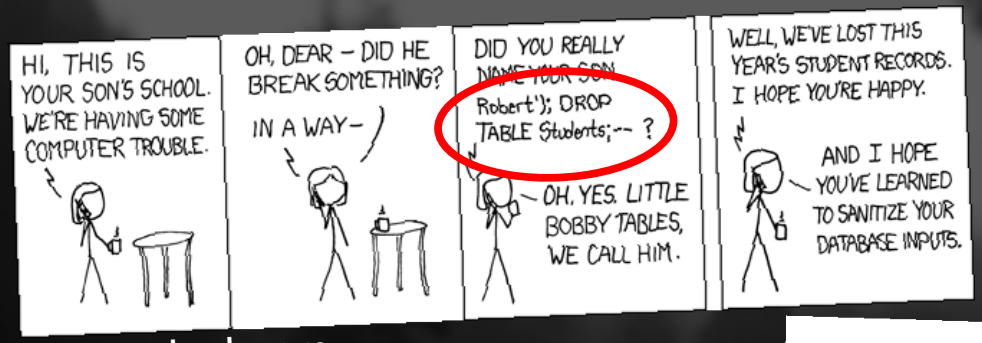


www.xkcd.com

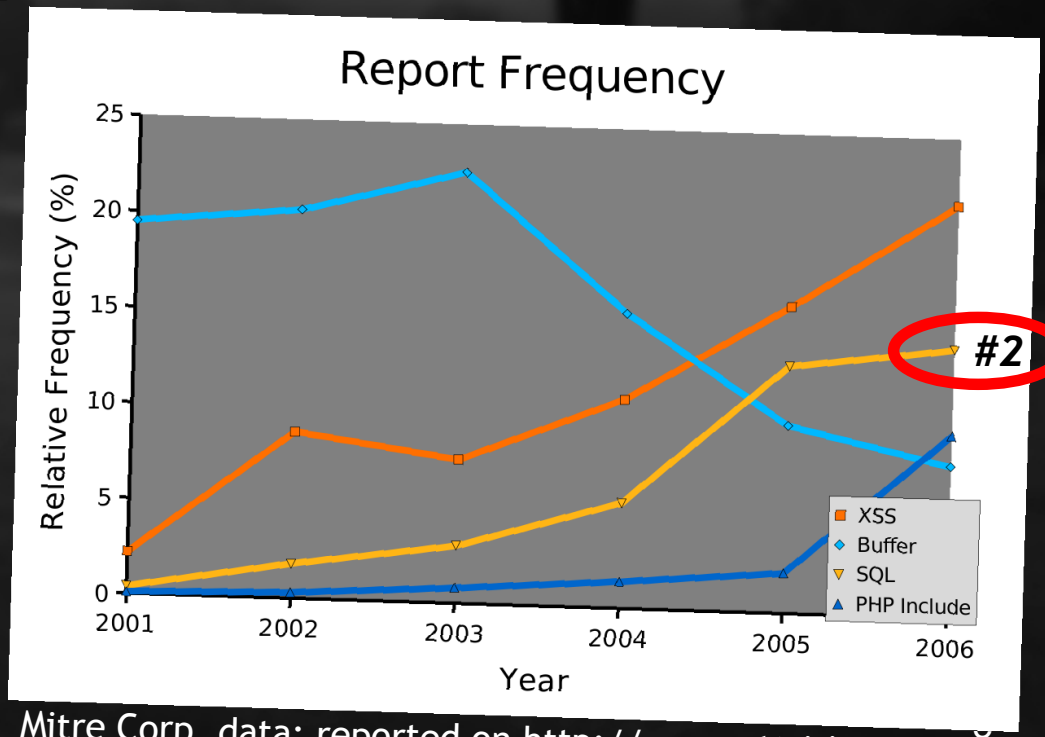


Mitre Corp. data reported on http://www.attrition.org/

Motivation Montage

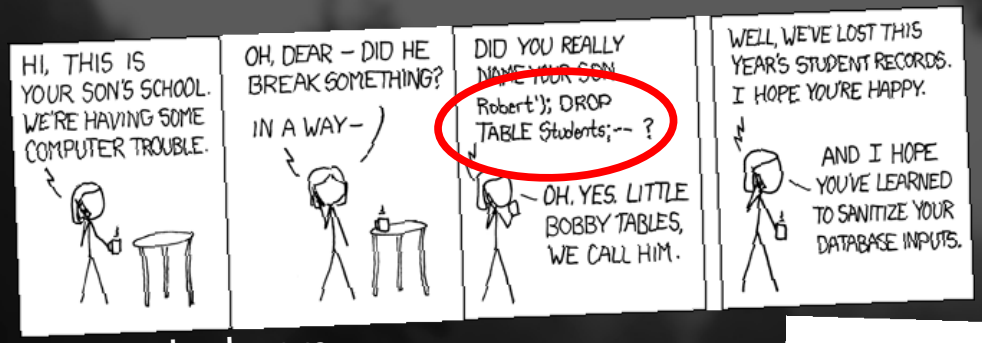


www.xkcd.com



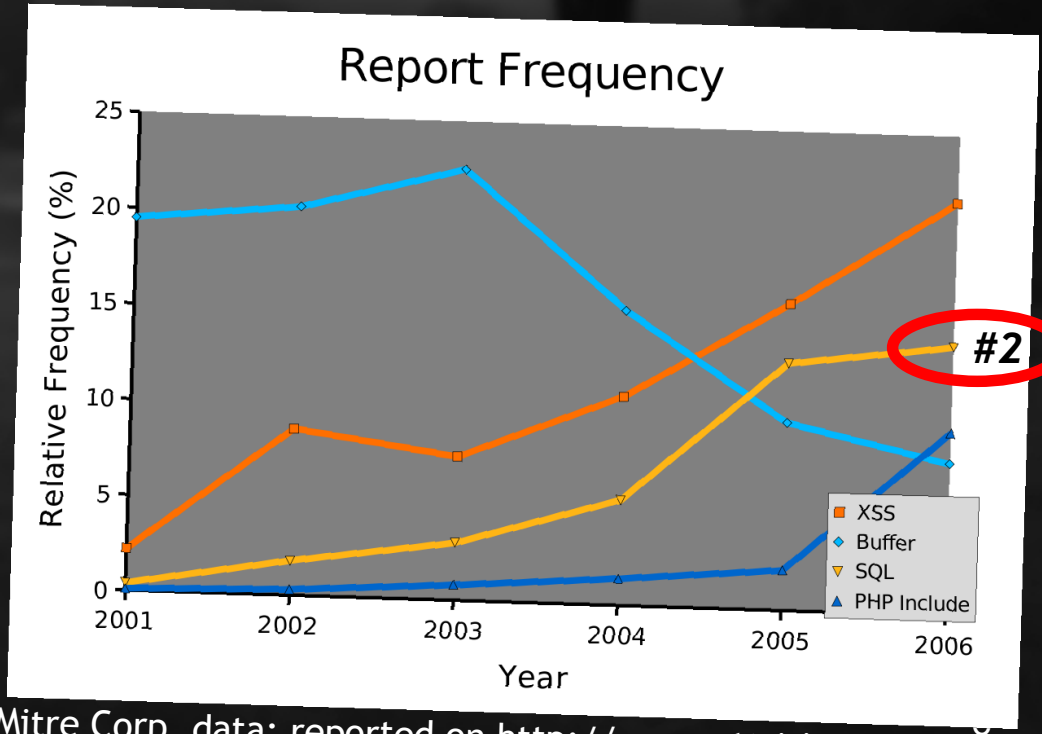
Mitre Corp. data; reported on http://www.attrition.org/

Motivation Montage



www.xkcd.com

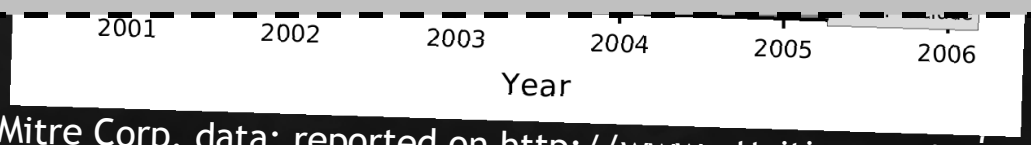
"String variables have lost their innocence..." [Thiemann05]



Mitre Corp. data; reported on http://www.attrition.org/

Motivation Montage

```
// $userid is untrusted

if (!eregi('[0-9]+', $userid)) {
  unp_msg('You entered an invalid user ID.');
  exit;
}

$user = $DB->query("SELECT * FROM `unp_user`".
                   "WHERE userid='$userid'");

if (!DB->is_single_row($user)) {
   unp_msg('You entered an invalid user ID.');
   exit;
}
```

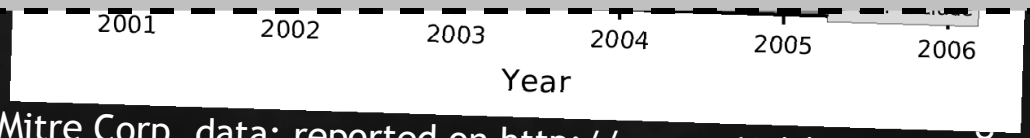


Mitre Corp. data; reported on http://www.attrition.org/

Motivation Montage

```
// $userid is untrusted

if (!eregi('[0-9]+', $userid)) {
   unp_msg('You entered an invalid user ID.');
   exit;
}

$user = $DB->query("SELECT * FROM `unp_user`
                    WHERE userid='$userid'");

if (!DB->is_single_row($user)) {
    unp_msg('You entered an invalid user ID.');
    exit;
}
```

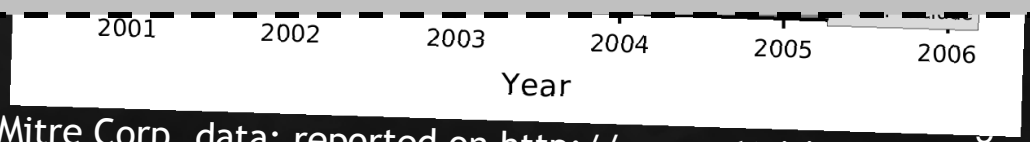
Matches any string that *contains* a sequence of digits...



Mitre Corp. data; reported on http://www.attrition.org/

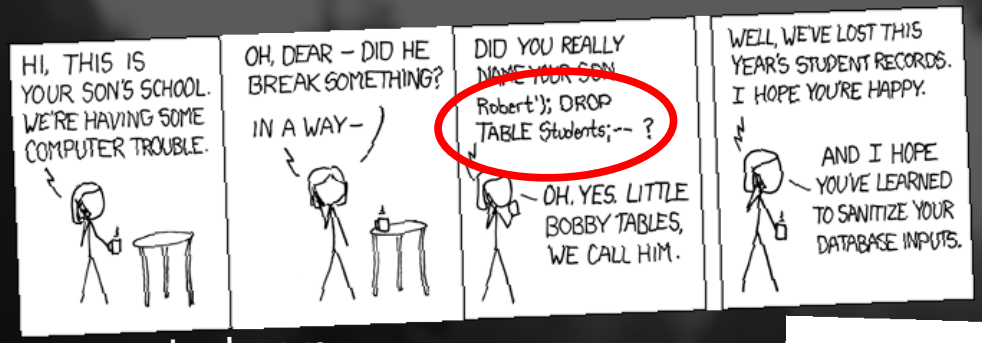
Motivation Montage

```
// $userid is untrusted
SELECT * FROM `unp_user`
                WHERE userid='1';
DROP TABLE unp_user;
__'
if                                                    ;
}

$user = $DB->query("SELECT * FROM `unp_user`".
                   "WHERE userid='$userid'");

if (!DB->is_single_row($user)) {
   unp_msg('You entered an invalid user ID.');
   exit;
}
```

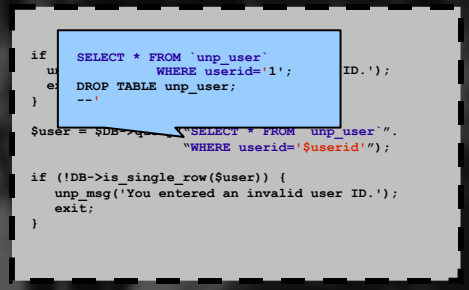


Mitre Corp. data; reported on http://www.attrition.org/

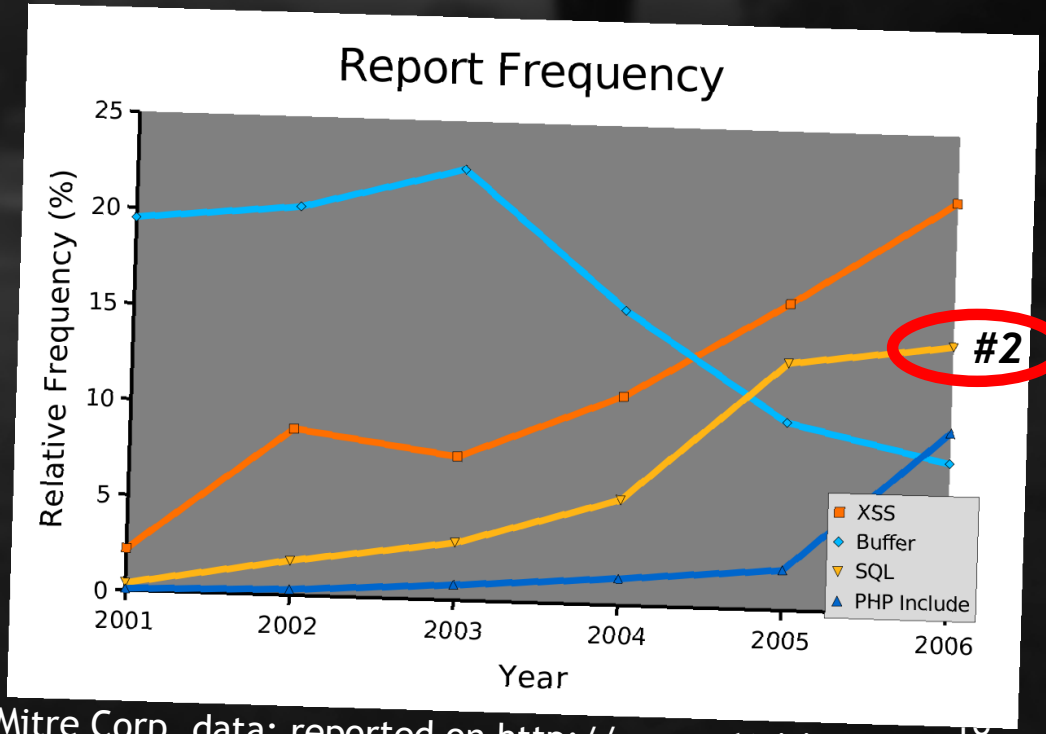
Motivation Montage



www.xkcd.com



"String variables have lost their innocence..." [Thiemann05]



Mitre Corp. data; reported on http://www.attrition.org/

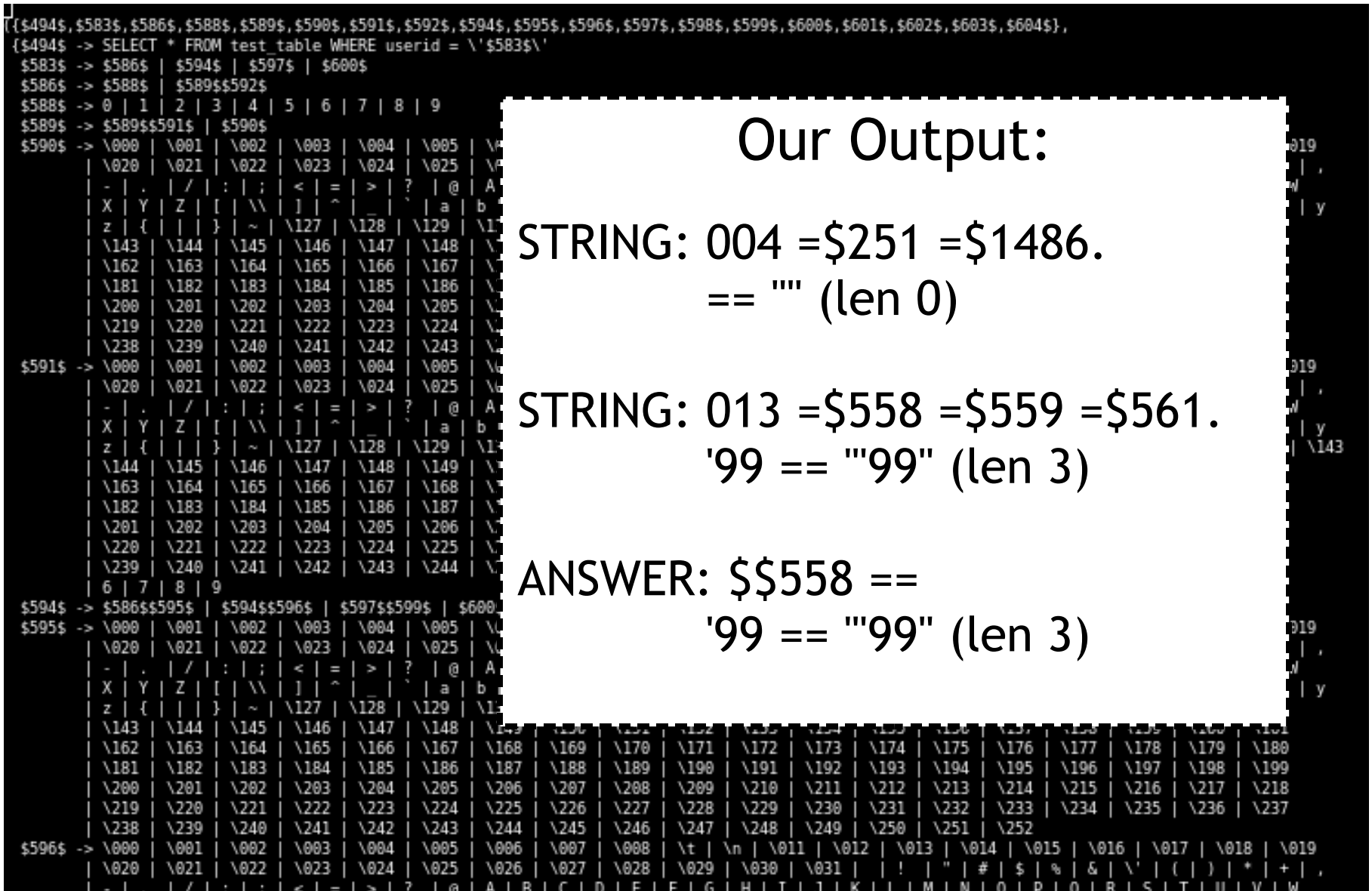
The Plan

- Wassermann and Su '07:
  - detect SQL Command Injection Vulnerabilities in real PHP code
  - Input: PHP code
    Output: Context-Free Grammar
- Plan: Extend this to generate
  actual attack inputs

Why?

```
{{$494$,$583$,$586$,$588$,$589$,$590$,$591$,$592$,$594$,$595$,$596$,$597$,$598$,$599$,$600$,$601$,$602$,$603$,$604$},
 {$494$ -> SELECT * FROM test_table WHERE userid = \'$583$\'
  $583$ -> $586$ | $594$ | $597$ | $600$
  $586$ -> $588$ | $589$$592$
  $588$ -> 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
  $589$ -> $589$$591$ | $590$
  $590$ -> \000 | \001 | \002 | \003 | \004 | \005 | \006 | \007 | \008 | \t | \n | \011 | \012 | \013 | \014 | \015 | \016 | \017 | \018 | \019
         | \020 | \021 | \022 | \023 | \024 | \025 | \026 | \027 | \028 | \029 | \030 | \031 |  | !  | " | # | $ | % | & | \' | ( | ) | * | + | ,
         | - | .  | / | : | ; | < | = | > | ?  | @ | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W
         | X | Y | Z | [ | \\ | ] | ^ | _ | `  | a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r | s | t | u | v | w | x | y
         | z | { | | | } | ~ | \127 | \128 | \129 | \130 | \131 | \132 | \133 | \134 | \135 | \136 | \137 | \138 | \139 | \140 | \141 | \142
         | \143 | \144 | \145 | \146 | \147 | \148 | \149 | \150 | \151 | \152 | \153 | \154 | \155 | \156 | \157 | \158 | \159 | \160 | \161
         | \162 | \163 | \164 | \165 | \166 | \167 | \168 | \169 | \170 | \171 | \172 | \173 | \174 | \175 | \176 | \177 | \178 | \179 | \180
         | \181 | \182 | \183 | \184 | \185 | \186 | \187 | \188 | \189 | \190 | \191 | \192 | \193 | \194 | \195 | \196 | \197 | \198 | \199
         | \200 | \201 | \202 | \203 | \204 | \205 | \206 | \207 | \208 | \209 | \210 | \211 | \212 | \213 | \214 | \215 | \216 | \217 | \218
         | \219 | \220 | \221 | \222 | \223 | \224 | \225 | \226 | \227 | \228 | \229 | \230 | \231 | \232 | \233 | \234 | \235 | \236 | \237
         | \238 | \239 | \240 | \241 | \242 | \243 | \244 | \245 | \246 | \247 | \248 | \249 | \250 | \251 | \252
  $591$ -> \000 | \001 | \002 | \003 | \004 | \005 | \006 | \007 | \008 | \t | \n | \011 | \012 | \013 | \014 | \015 | \016 | \017 | \018 | \019
         | \020 | \021 | \022 | \023 | \024 | \025 | \026 | \027 | \028 | \029 | \030 | \031 |  | !  | " | # | $ | % | & | \' | ( | ) | * | + | ,
         | - | .  | / | : | ; | < | = | > | ?  | @ | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W
         | X | Y | Z | [ | \\ | ] | ^ | _ | `  | a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r | s | t | u | v | w | x | y
         | z | { | | | } | ~ | \127 | \128 | \129 | \130 | \131 | \132 | \133 | \134 | \135 | \136 | \137 | \138 | \139 | \140 | \141 | \142 | \143
         | \144 | \145 | \146 | \147 | \148 | \149 | \150 | \151 | \152 | \153 | \154 | \155 | \156 | \157 | \158 | \159 | \160 | \161 | \162
         | \163 | \164 | \165 | \166 | \167 | \168 | \169 | \170 | \171 | \172 | \173 | \174 | \175 | \176 | \177 | \178 | \179 | \180 | \181
         | \182 | \183 | \184 | \185 | \186 | \187 | \188 | \189 | \190 | \191 | \192 | \193 | \194 | \195 | \196 | \197 | \198 | \199 | \200
         | \201 | \202 | \203 | \204 | \205 | \206 | \207 | \208 | \209 | \210 | \211 | \212 | \213 | \214 | \215 | \216 | \217 | \218 | \219
         | \220 | \221 | \222 | \223 | \224 | \225 | \226 | \227 | \228 | \229 | \230 | \231 | \232 | \233 | \234 | \235 | \236 | \237 | \238
         | \239 | \240 | \241 | \242 | \243 | \244 | \245 | \246 | \247 | \248 | \249 | \250 | \251 | \252 , $592$ -> 0 | 1 | 2 | 3 | 4 | 5
         | 6 | 7 | 8 | 9
  $594$ -> $586$$595$ | $594$$596$ | $597$$599$ | $600$$604$
  $595$ -> \000 | \001 | \002 | \003 | \004 | \005 | \006 | \007 | \008 | \t | \n | \011 | \012 | \013 | \014 | \015 | \016 | \017 | \018 | \019
         | \020 | \021 | \022 | \023 | \024 | \025 | \026 | \027 | \028 | \029 | \030 | \031 |  | !  | " | # | $ | % | & | \' | ( | ) | * | + | ,
         | - | .  | / | : | ; | < | = | > | ?  | @ | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W
         | X | Y | Z | [ | \\ | ] | ^ | _ | `  | a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r | s | t | u | v | w | x | y
         | z | { | | | } | ~ | \127 | \128 | \129 | \130 | \131 | \132 | \133 | \134 | \135 | \136 | \137 | \138 | \139 | \140 | \141 | \142
         | \143 | \144 | \145 | \146 | \147 | \148 | \149 | \150 | \151 | \152 | \153 | \154 | \155 | \156 | \157 | \158 | \159 | \160 | \161
         | \162 | \163 | \164 | \165 | \166 | \167 | \168 | \169 | \170 | \171 | \172 | \173 | \174 | \175 | \176 | \177 | \178 | \179 | \180
         | \181 | \182 | \183 | \184 | \185 | \186 | \187 | \188 | \189 | \190 | \191 | \192 | \193 | \194 | \195 | \196 | \197 | \198 | \199
         | \200 | \201 | \202 | \203 | \204 | \205 | \206 | \207 | \208 | \209 | \210 | \211 | \212 | \213 | \214 | \215 | \216 | \217 | \218
         | \219 | \220 | \221 | \222 | \223 | \224 | \225 | \226 | \227 | \228 | \229 | \230 | \231 | \232 | \233 | \234 | \235 | \236 | \237
         | \238 | \239 | \240 | \241 | \242 | \243 | \244 | \245 | \246 | \247 | \248 | \249 | \250 | \251 | \252
  $596$ -> \000 | \001 | \002 | \003 | \004 | \005 | \006 | \007 | \008 | \t | \n | \011 | \012 | \013 | \014 | \015 | \016 | \017 | \018 | \019
         | \020 | \021 | \022 | \023 | \024 | \025 | \026 | \027 | \028 | \029 | \030 | \031 |  | !  | " | # | $ | % | & | \' | ( | ) | * | + | ,
         | - | .  | / | : | ; | < | = | > | ?  | @ | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W
```

Why?



Our Output:

STRING: 004 =$251 =$1486.
== "" (len 0)

STRING: 013 =$558 =$559 =$561.
'99 == '"99" (len 3)

ANSWER: $$558 ==
'99 == '"99" (len 3)

Up Next

1. Describe Wassermann and Su '07
2. How to run PHP code backwards

WSU: An Example

Some Code:

```
x = 'z';

while(n < 5) {
  x = '(' . x . ')';
  n ++;
}
```

- We want a context-free grammar to model **x**
- Suppose we don't know anything about **n**

WSU: An Example

Some Code:

```
>x = 'z';

 while(n < 5) {
   x = '(' . x . ')';
   n ++;
 }
```

Grammar:

```
A -> z
```

WSU: An Example

Some Code:

```
 x = 'z';

>while(n < 5) {
>   x = '(' . x . ')';
>   n ++;
>}
```

Grammar:

```
A -> z
```

WSU: An Example

Some Code:

```
 x = 'z';

 while(n < 5) {
>   x = '(' . x . ')';
    n ++;
 }
```

Grammar:

```
A -> z

B -> (A)
```

The Nugget

Some Code:

```
 x = 'z';

 while(n < 5) {
    x = '(' . x . ')';
    n ++;
>>}
```

Grammar:

```
A -> z

B -> (A)
C -> A | B
```

WSU: Yet More Example

Some Code:

```
 x = 'z';

 while(n < 5) {
    x = '(' . x . ')';
    n ++;
>>}
```

Grammar:

```
A -> z
B -> (C)
C -> A | B
```

WSU: An Example

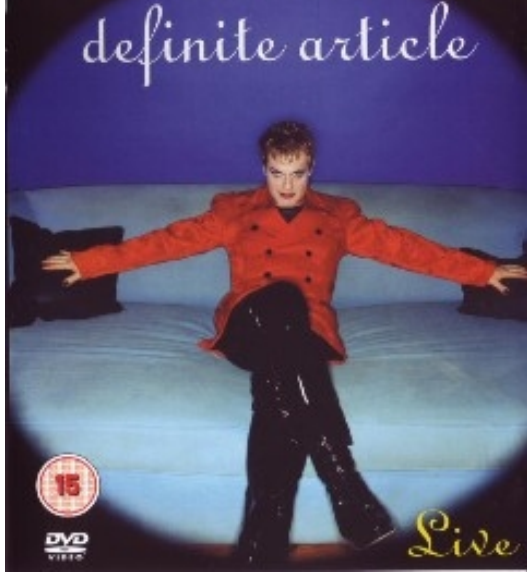
Some Code:

```
 x = 'z';

 while(n < 5) {
    x = '(' . x . ')';
    n ++;
>>}
```

Grammar:

```
X -> C


A -> z
B -> (C)
C -> A | B
```

Is that good?

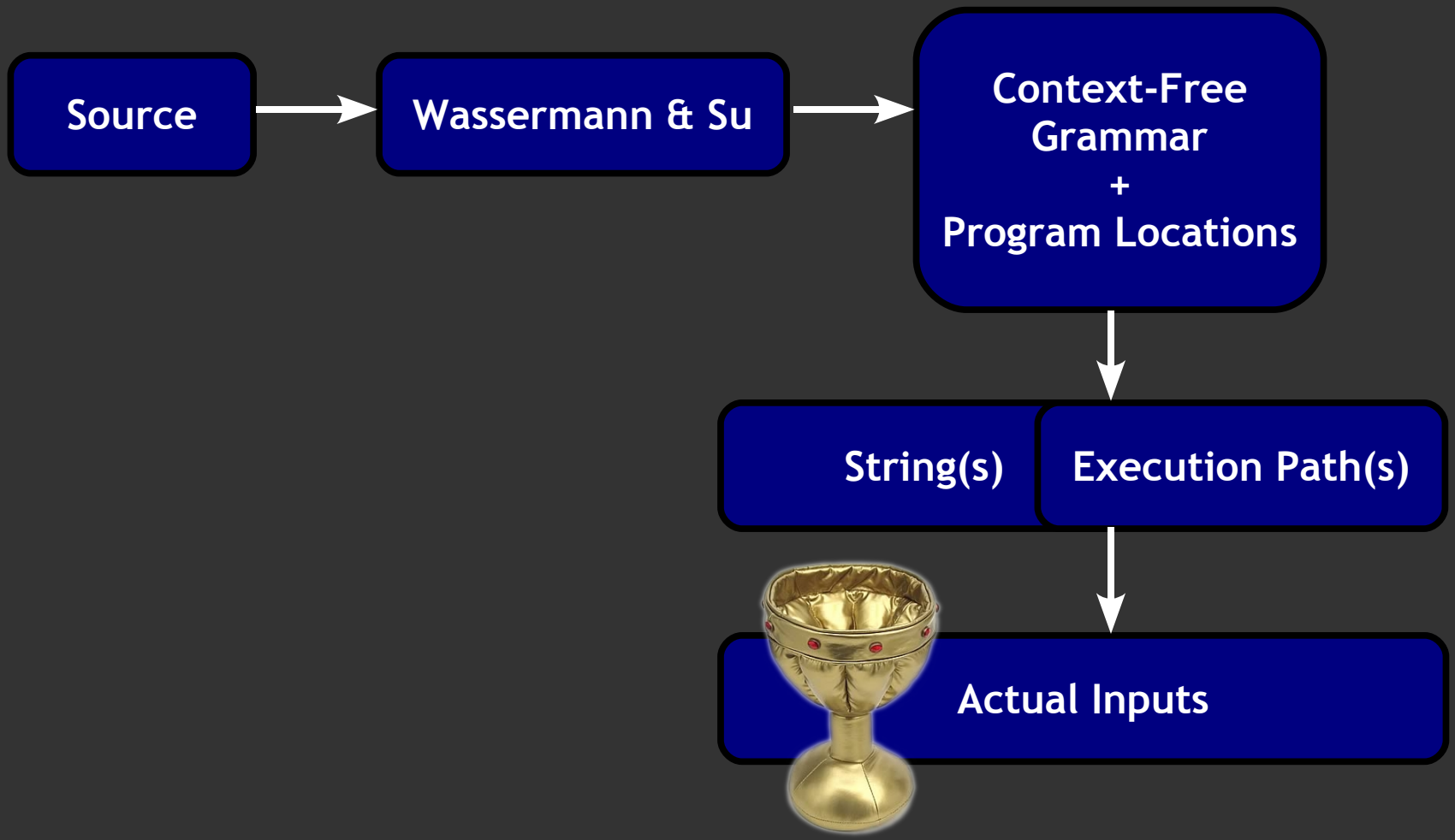


- Can model home-grown string sanitizing functions using finite state transducers [Minamide05]
- Does not require programmer assistance; always terminates
- 20% False positives; output may be difficult to interpret

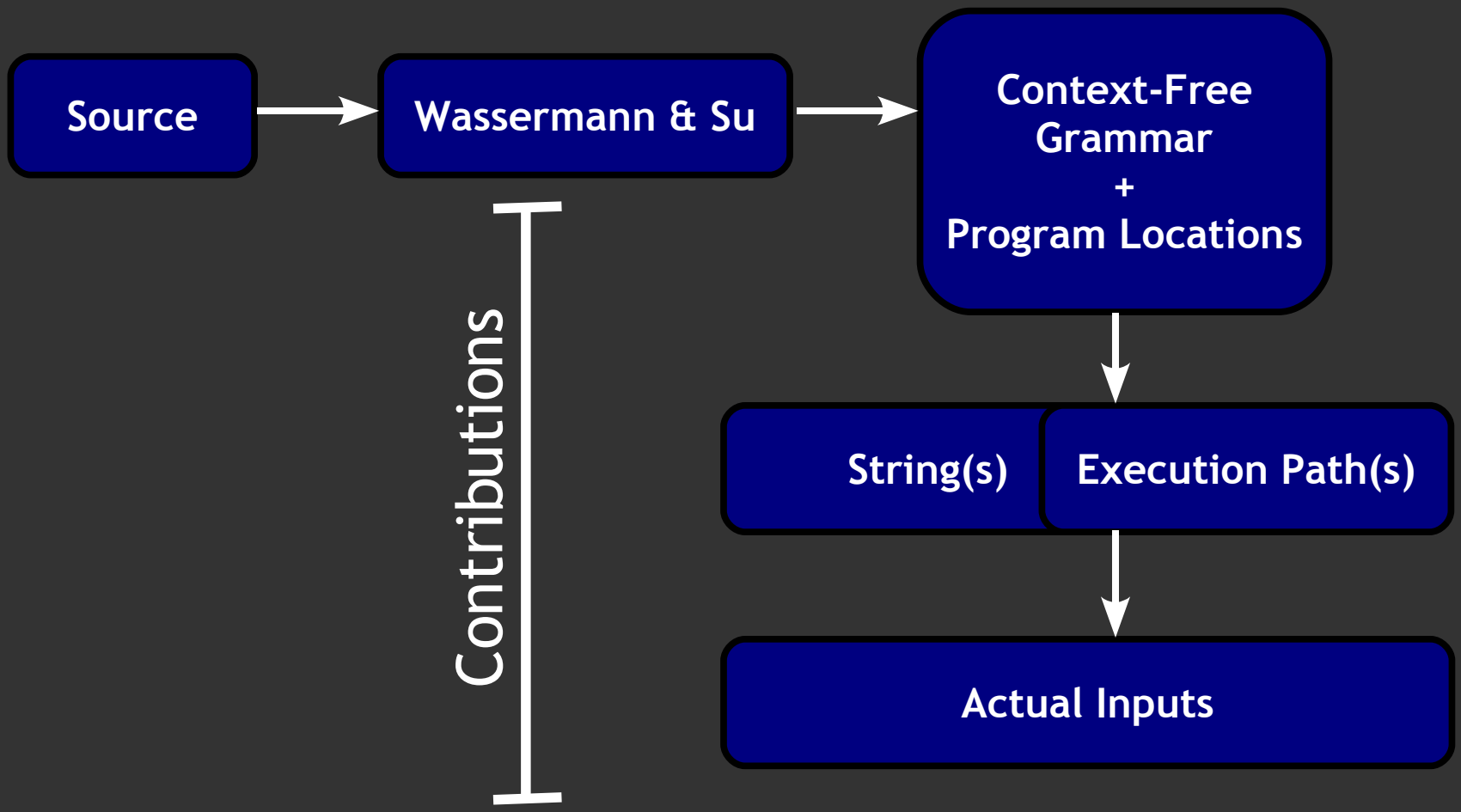

Ur instrucshuns
NOM
r unnesussary
ICANHASCHEEZBURGER.COM

Before:

After:


Source
Wassermann & Su
Context-Free Grammar + Program Locations
String(s)
Execution Path(s)
Actual Inputs

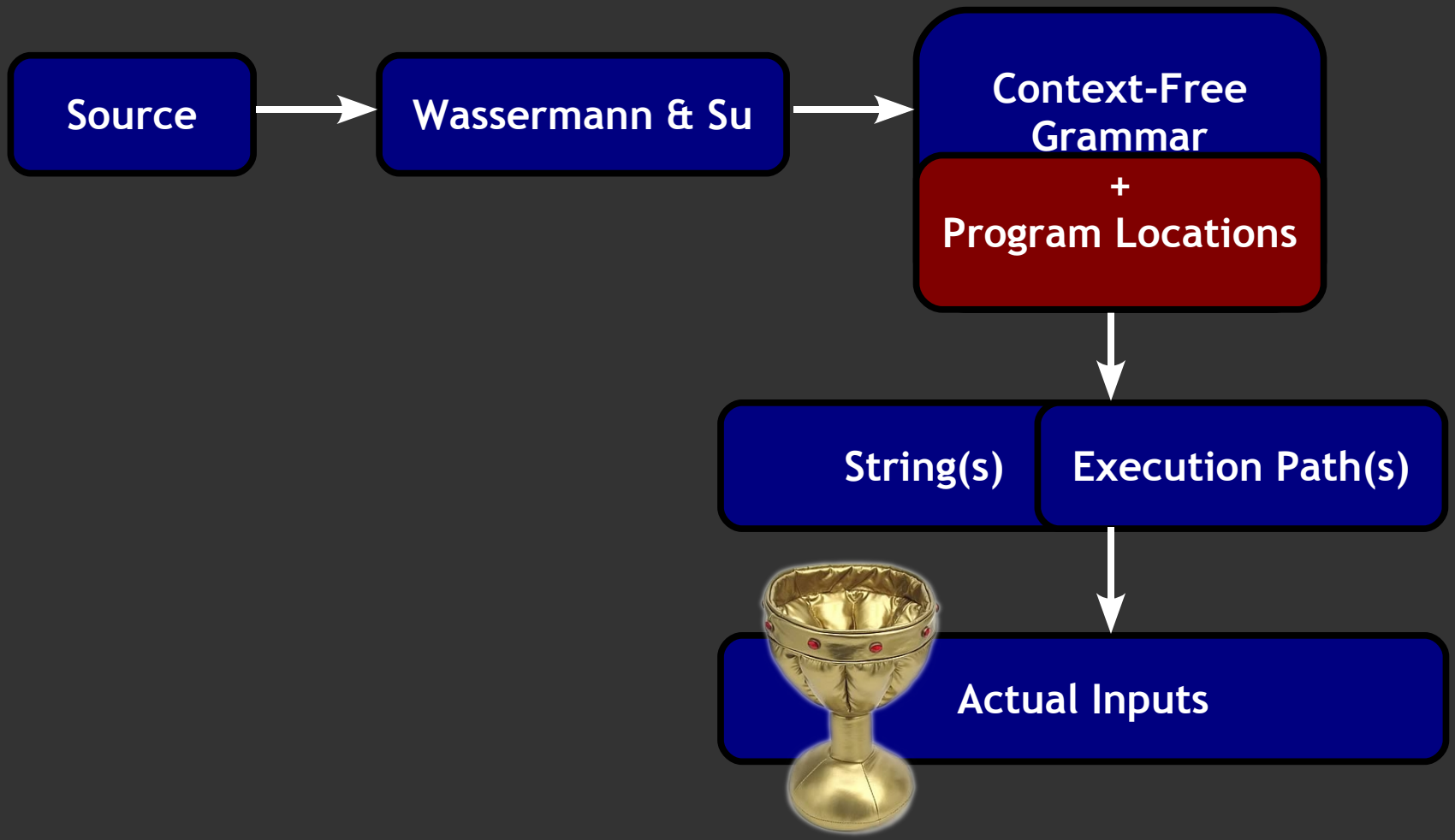
After:


Source
Wassermann & Su
Context-Free Grammar + Program Locations
Contributions
String(s)
Execution Path(s)
Actual Inputs

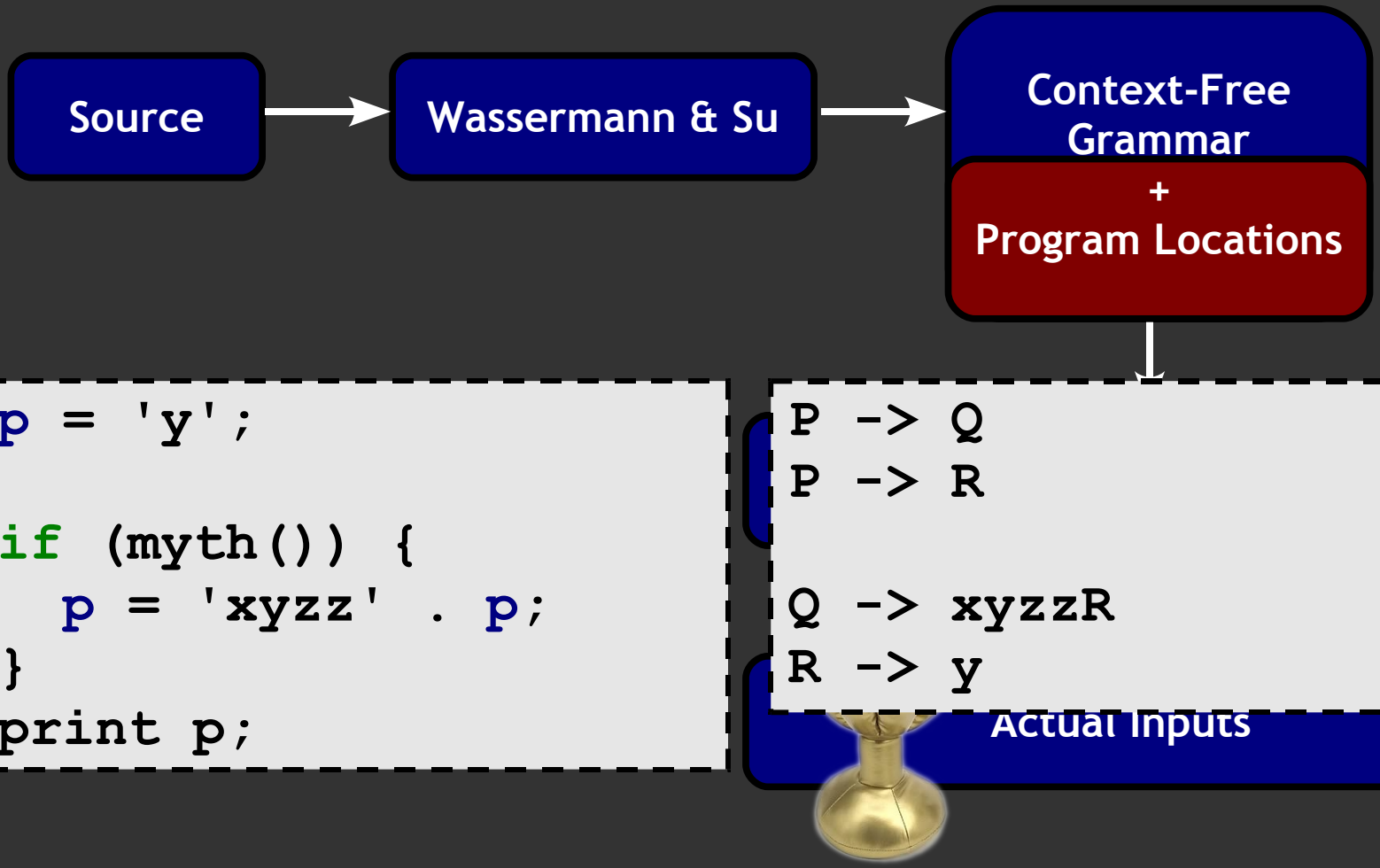
Contributions

- Add a mapping from context-free grammar back to the source
- Use mapping to find bad strings and execution paths
- Use symbolic execution to reverse string operations along a path

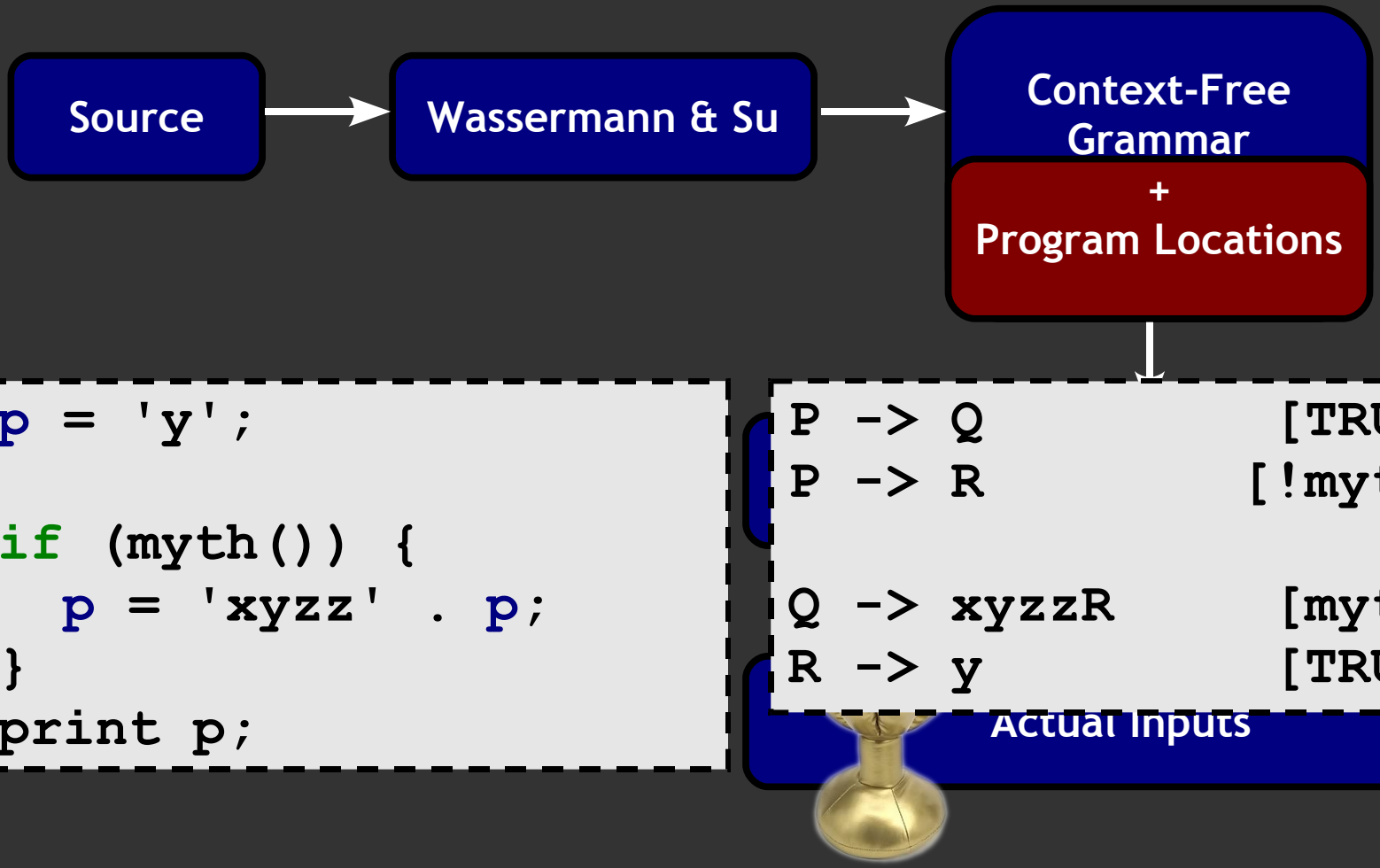
Up Next:

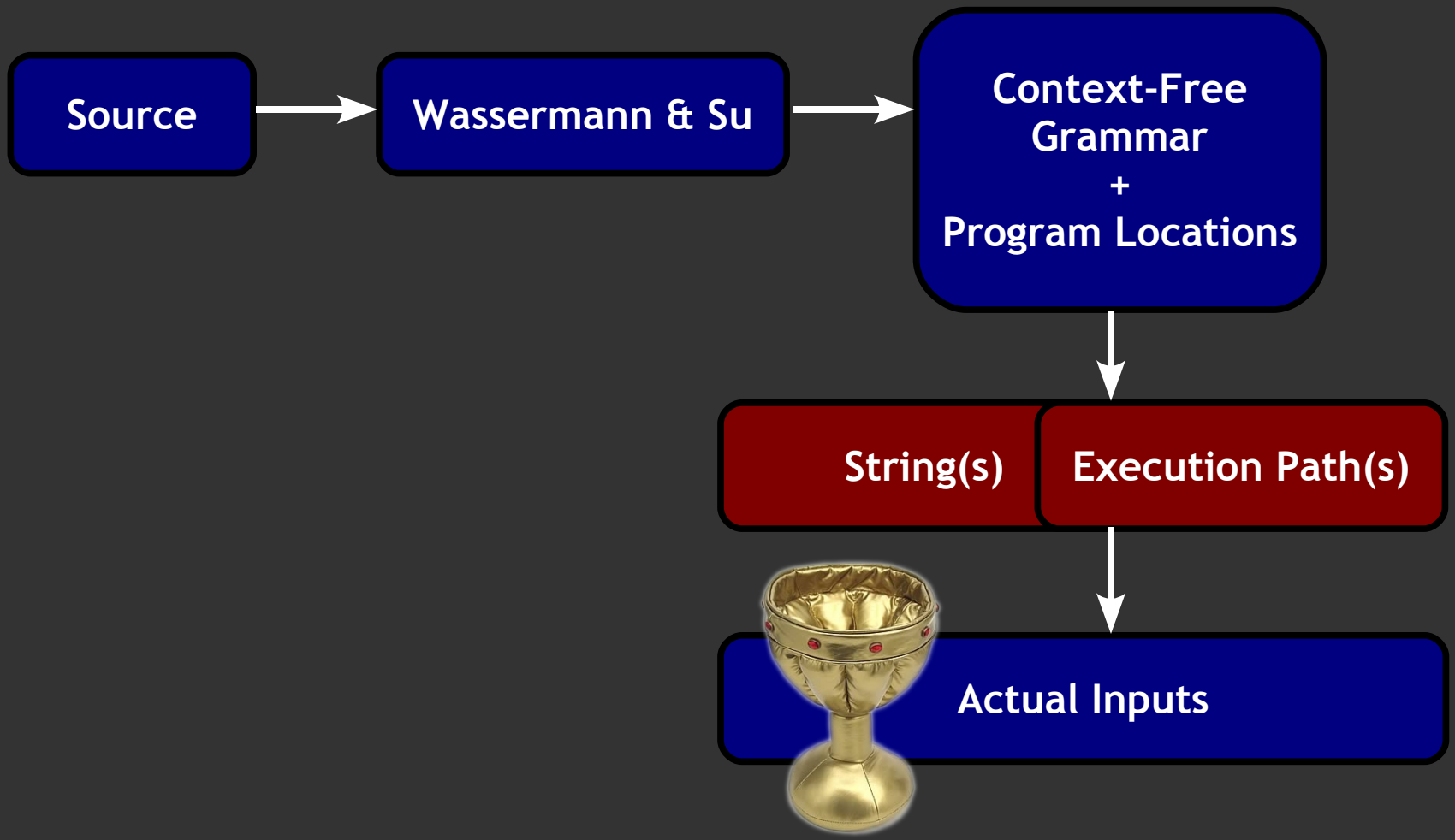

Source
Wassermann & Su
Context-Free Grammar
+
Program Locations
String(s)
Execution Path(s)
Actual Inputs

Grammar Annotations

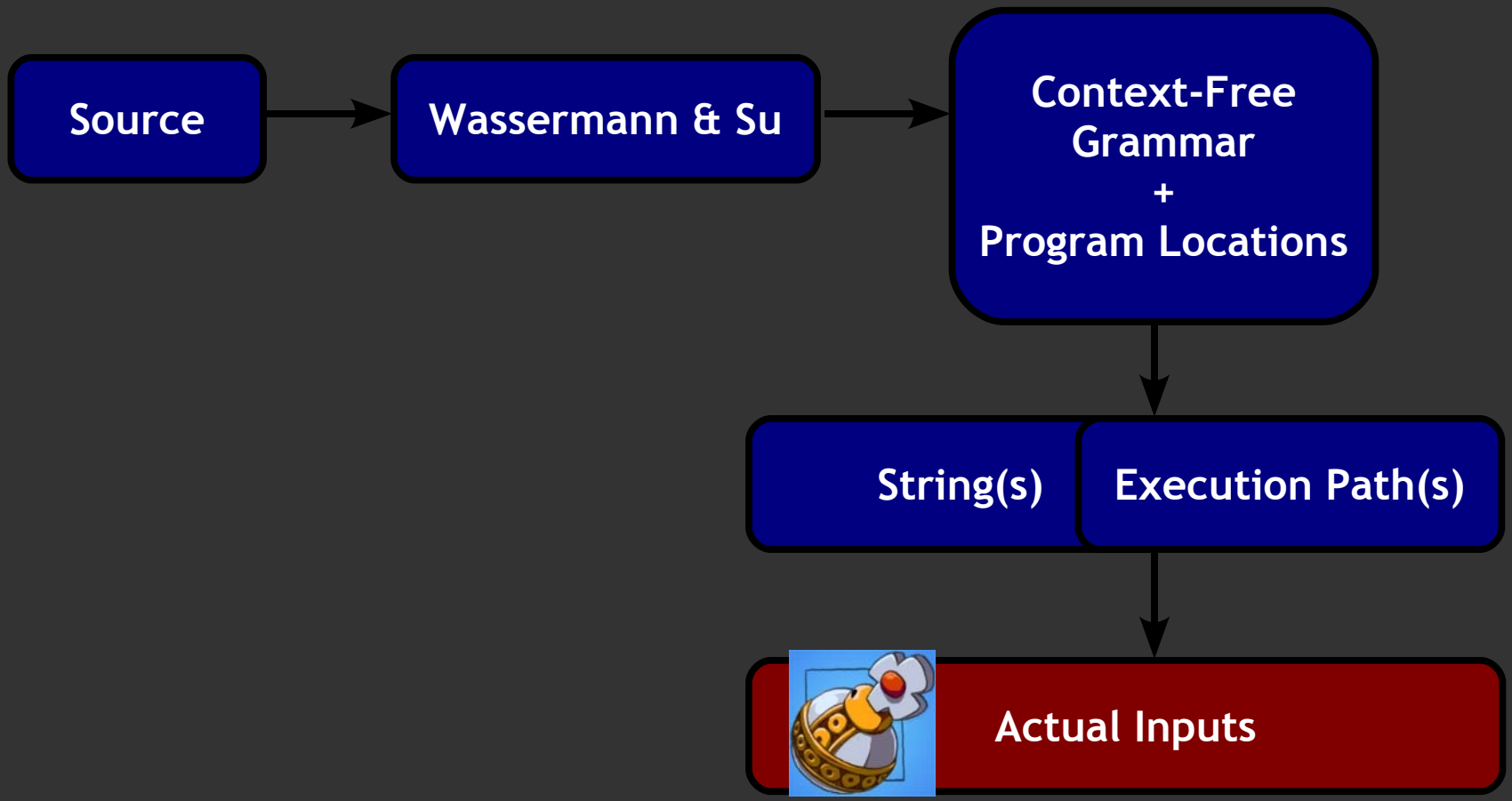


Grammar Annotations




Source
Wassermann & Su
Context-Free Grammar + Program Locations
String(s)
Execution Path(s)
Actual Inputs

Up Next:


Source
Wassermann & Su
Context-Free Grammar + Program Locations
String(s)
Execution Path(s)
Actual Inputs

How to Find Inputs

1. Create a dependency graph
2. Detect implicit cycles
3. Solve constraints

Creating a Dependency Graph

```
1    $userid = $_POST['uid'];
2    if (!eregi('[0-9]+', $userid)) {
3        exit;
4    }
5
6    query("SELECT * FROM `unp_user`".
7          "WHERE userid=" . $userid);
```

Path: 1-2-4-6

Creating a Dependency Graph

```
1    $userid = $_POST['uid'];
2    if (!eregi('[0-9]+', $userid)) {
3        exit;
4    }
5
6    query("SELECT * FROM `unp_user` .
7          "WHERE userid=" . $userid);
```

t0

t1

t2

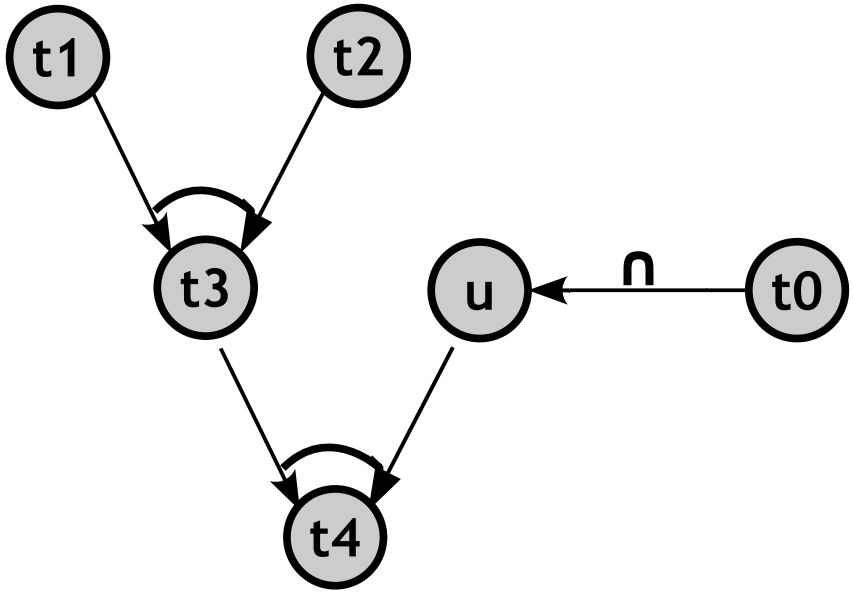
Path: 1-2-4-6

```
1    $userid = $_POST['uid'];
2    if (!eregi('[0-9]+', $userid)) {
3        exit;
4    }
5
6    query("SELECT * FROM `unp_user` .
7          "WHERE userid=" . $userid);
```

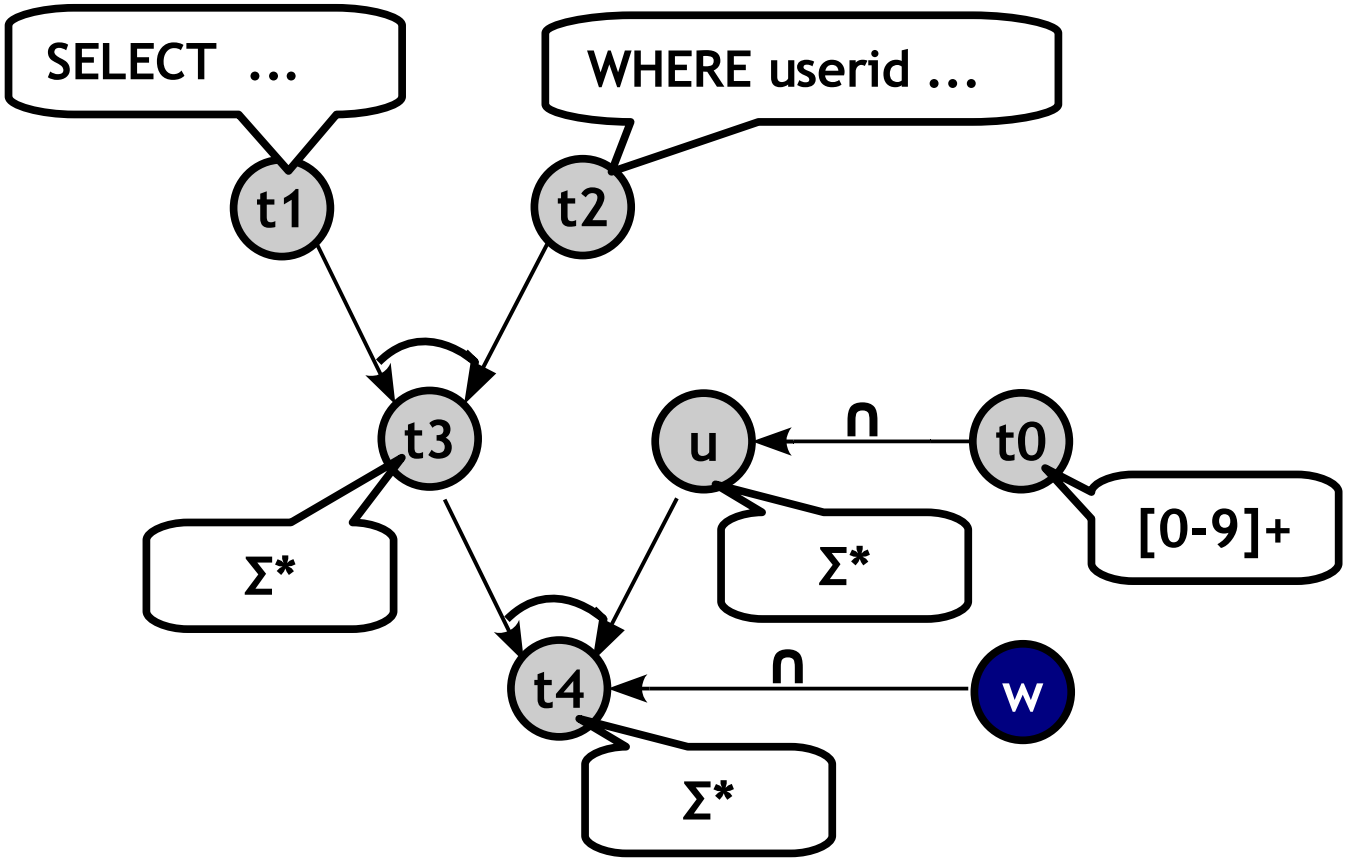
t0

t1

t2

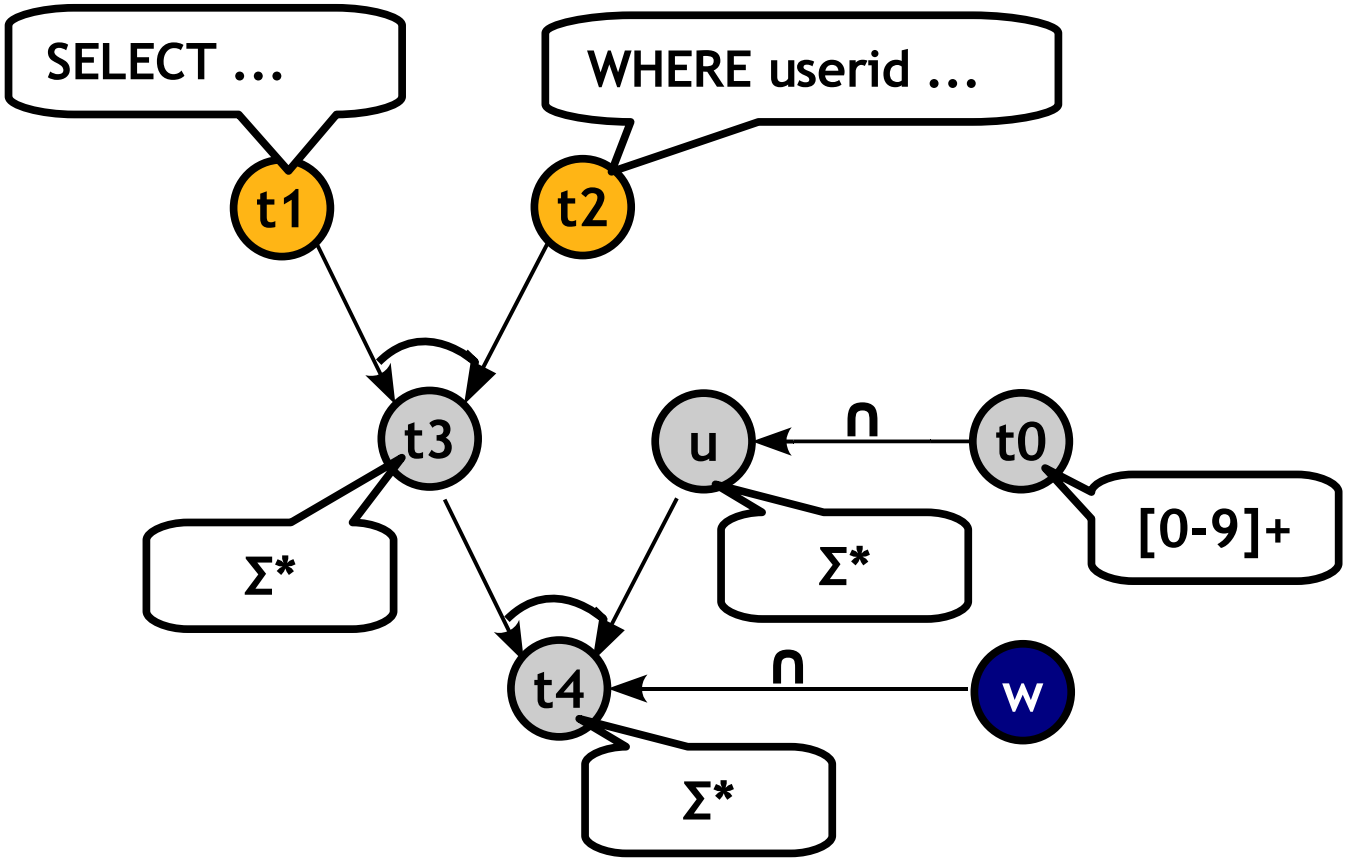


Solving Constraints



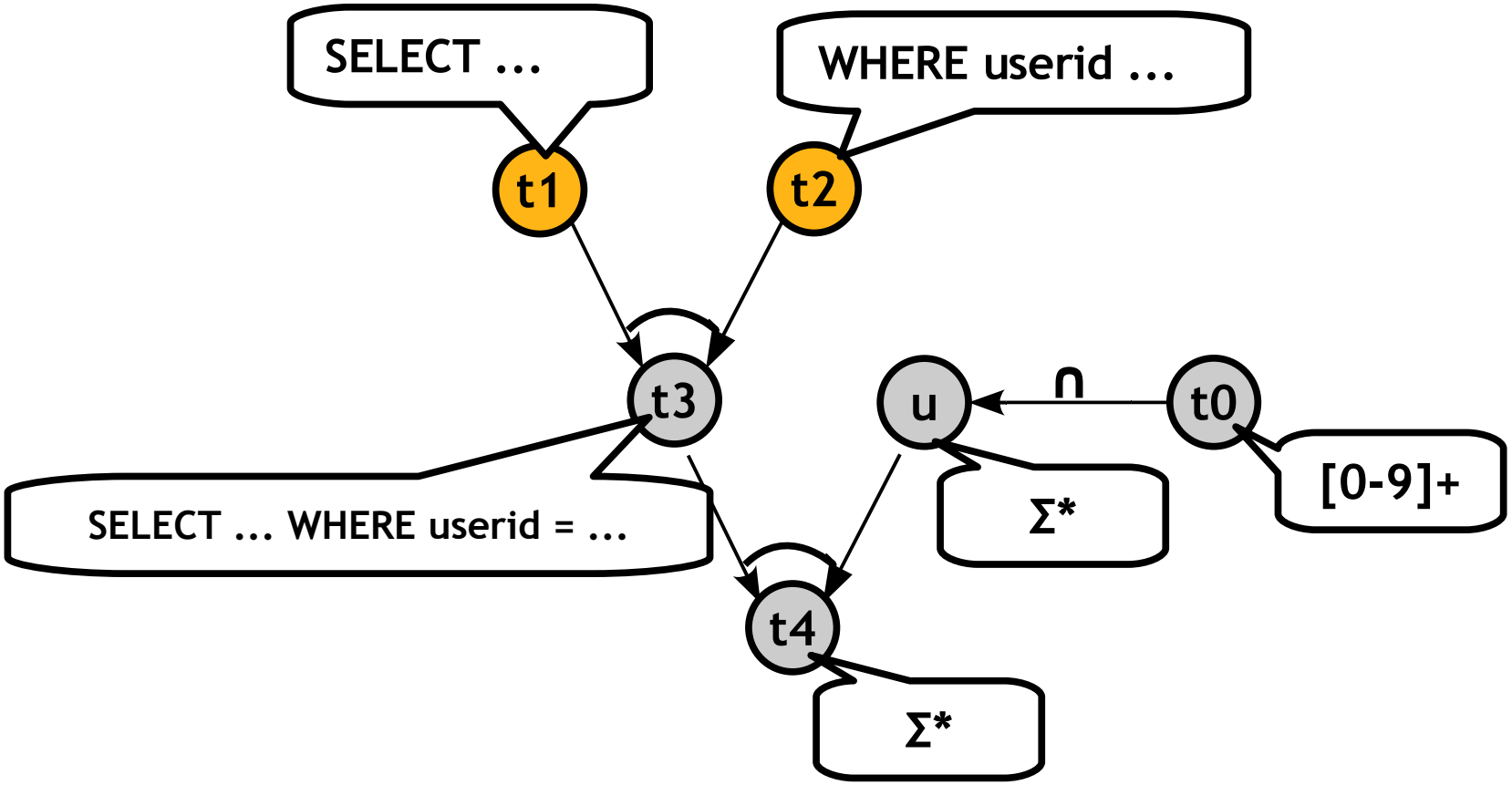

```
SELECT * FROM `unp_user` WHERE userid=1 OR 1=1
```

Solving Constraints



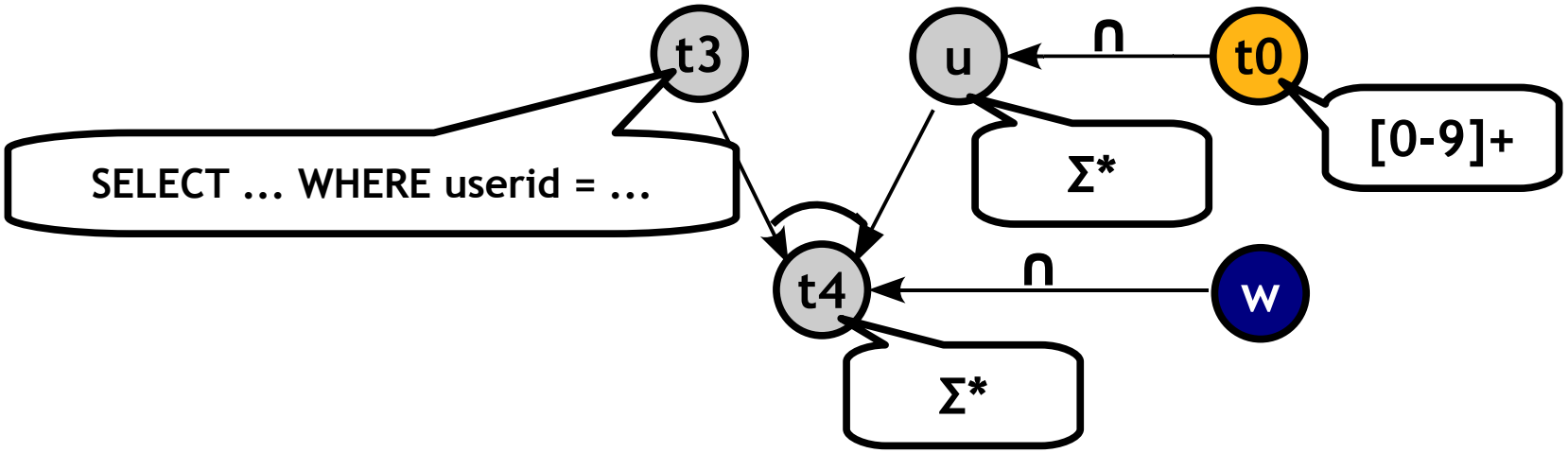

```
SELECT * FROM `unp_user` WHERE userid=1 OR 1=1
```

Solving Constraints



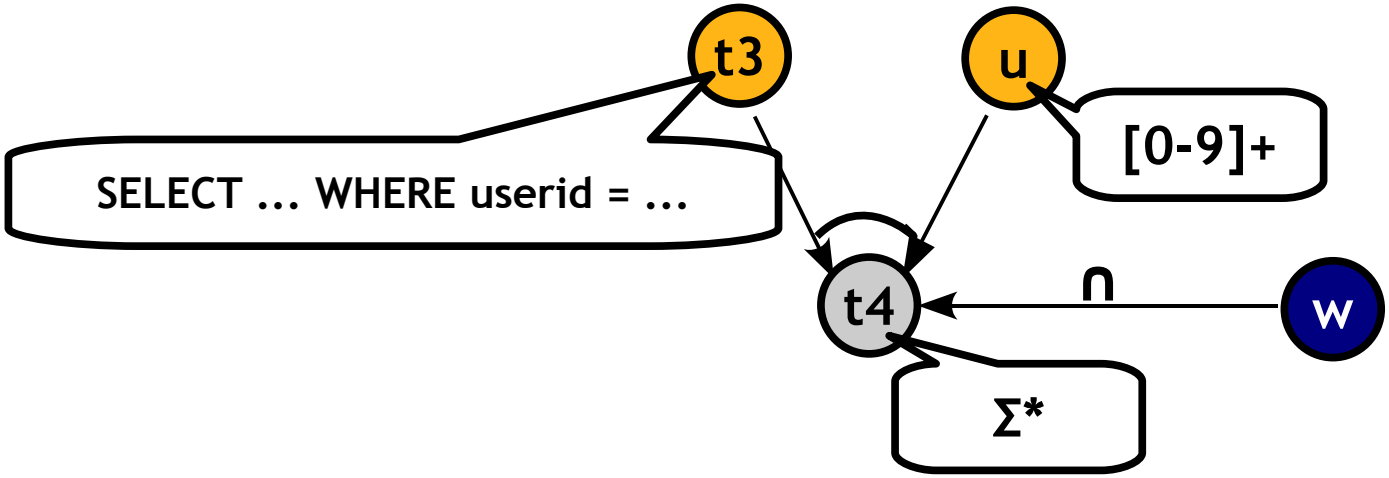

```
SELECT * FROM `unp_user` WHERE userid=1 OR 1=1
```

Solving Constraints



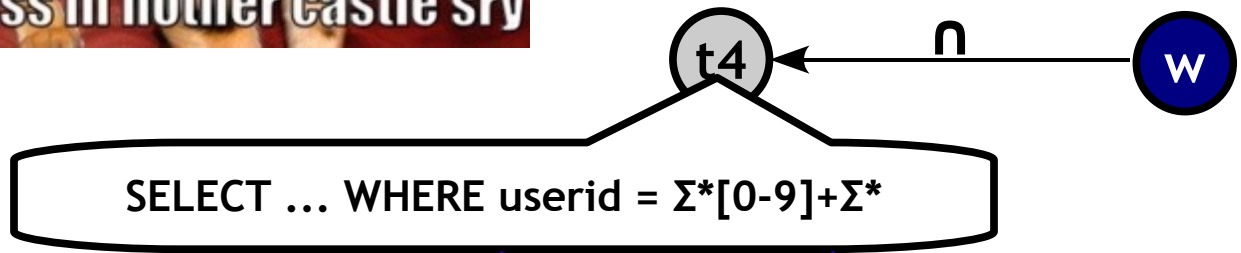

```
SELECT * FROM `unp_user` WHERE userid=1 OR 1=1
```

Solving Constraints




```
SELECT * FROM `unp_user` WHERE userid=1 OR 1=1
```

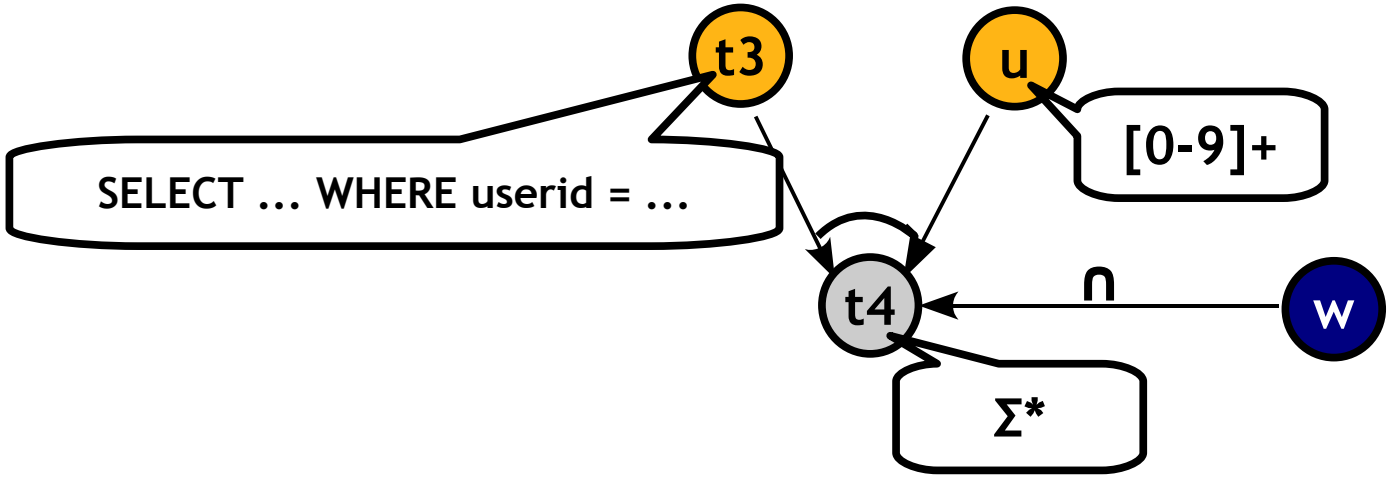
Solving Basic Constraints




```
SELECT * FROM `unp_user` WHERE userid=1 OR 1=1
```

Problem: Backward Propagation

How do we map `1 OR 1=1` back onto *u*?



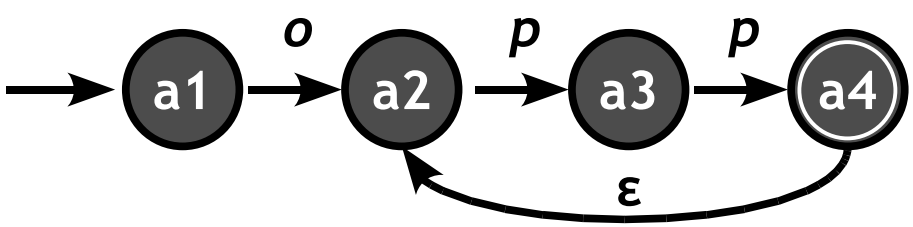

```
SELECT * FROM `unp_user` WHERE userid=1 OR 1=1
```

Concat-Intersect Example I

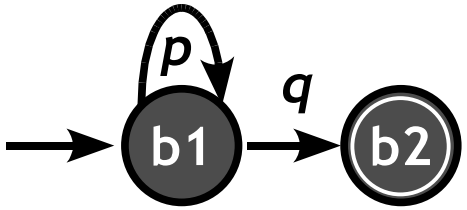
```
// a and b are user inputs

if (!ereg('o(pp)+', a)){ exit; }
if (!ereg('p*q', b)){ exit; }

d = a . b; // concat
if (!ereg('oppppq', d) { exit; }
```

Concat-Intersect Example I

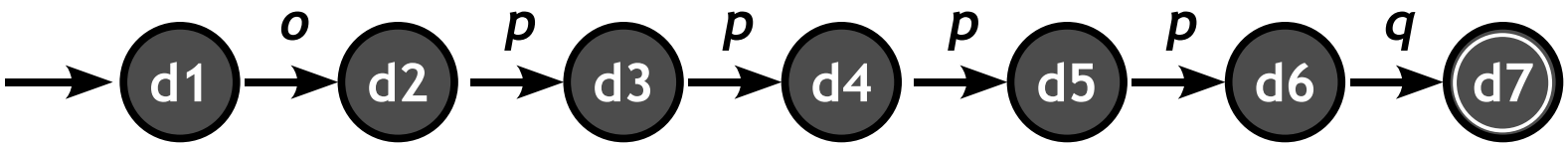
$L_1 = o(pp)+$

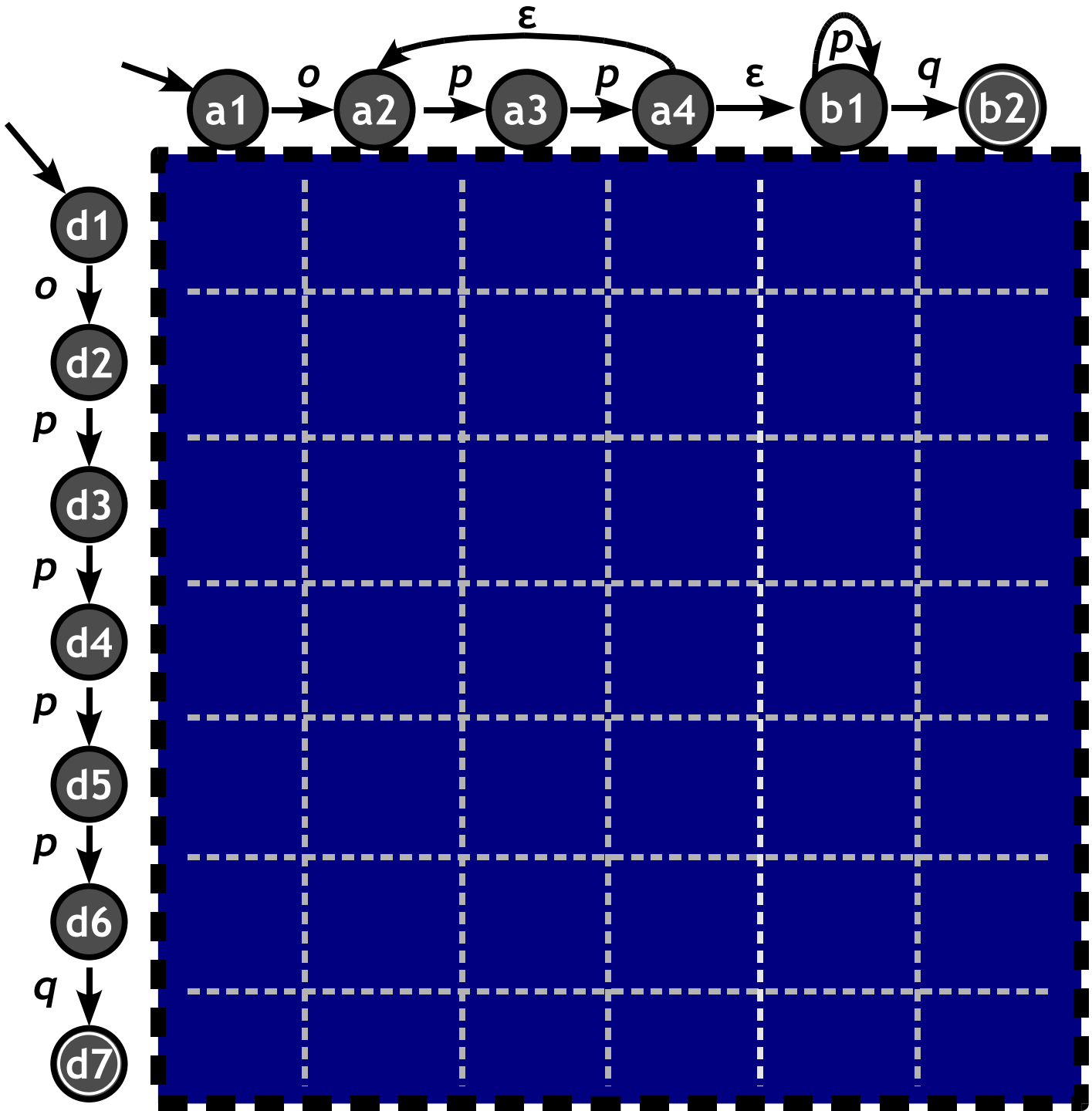


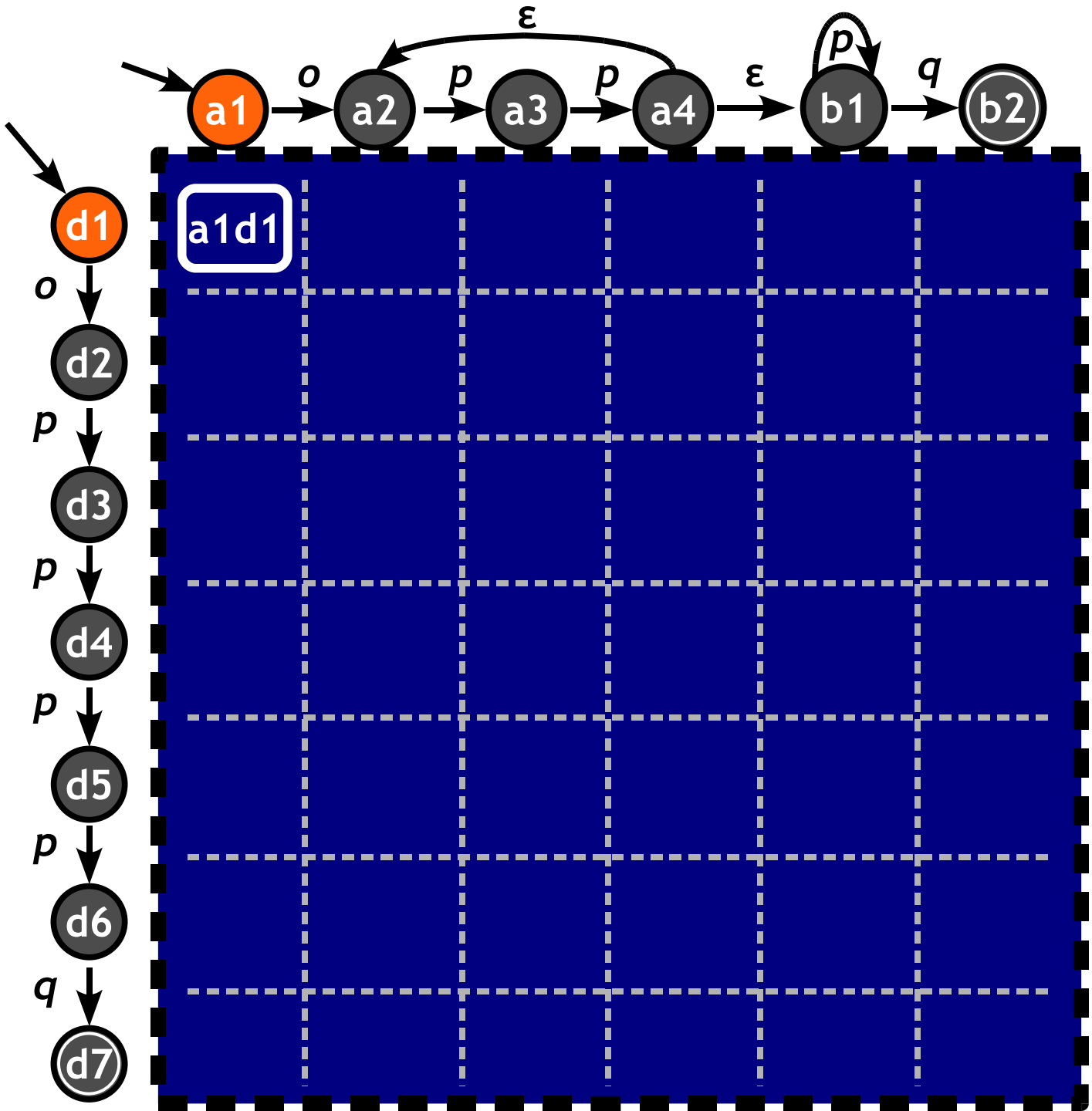
$L_2 = p^*q$

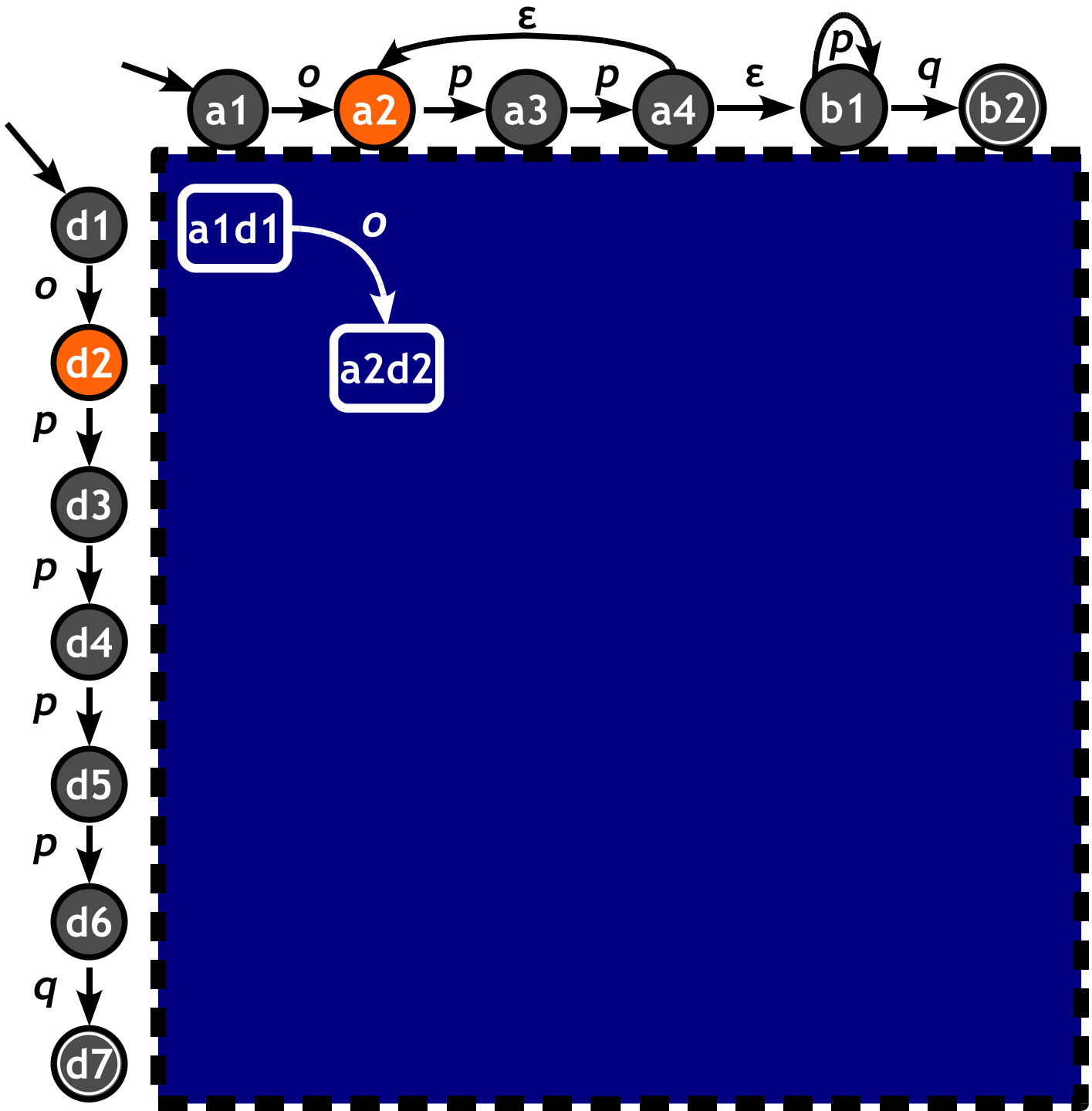


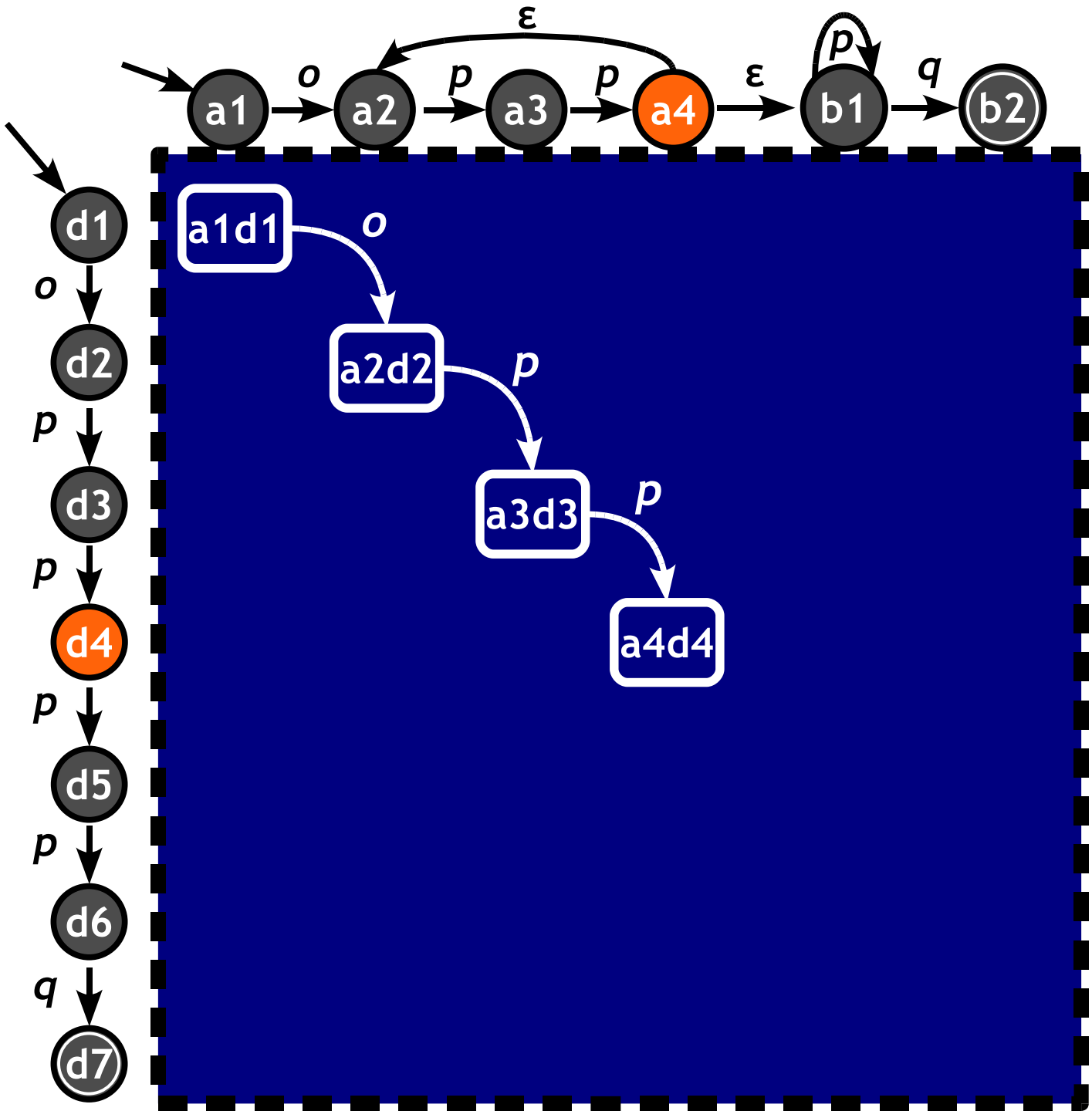
$L_3 = op^4q$

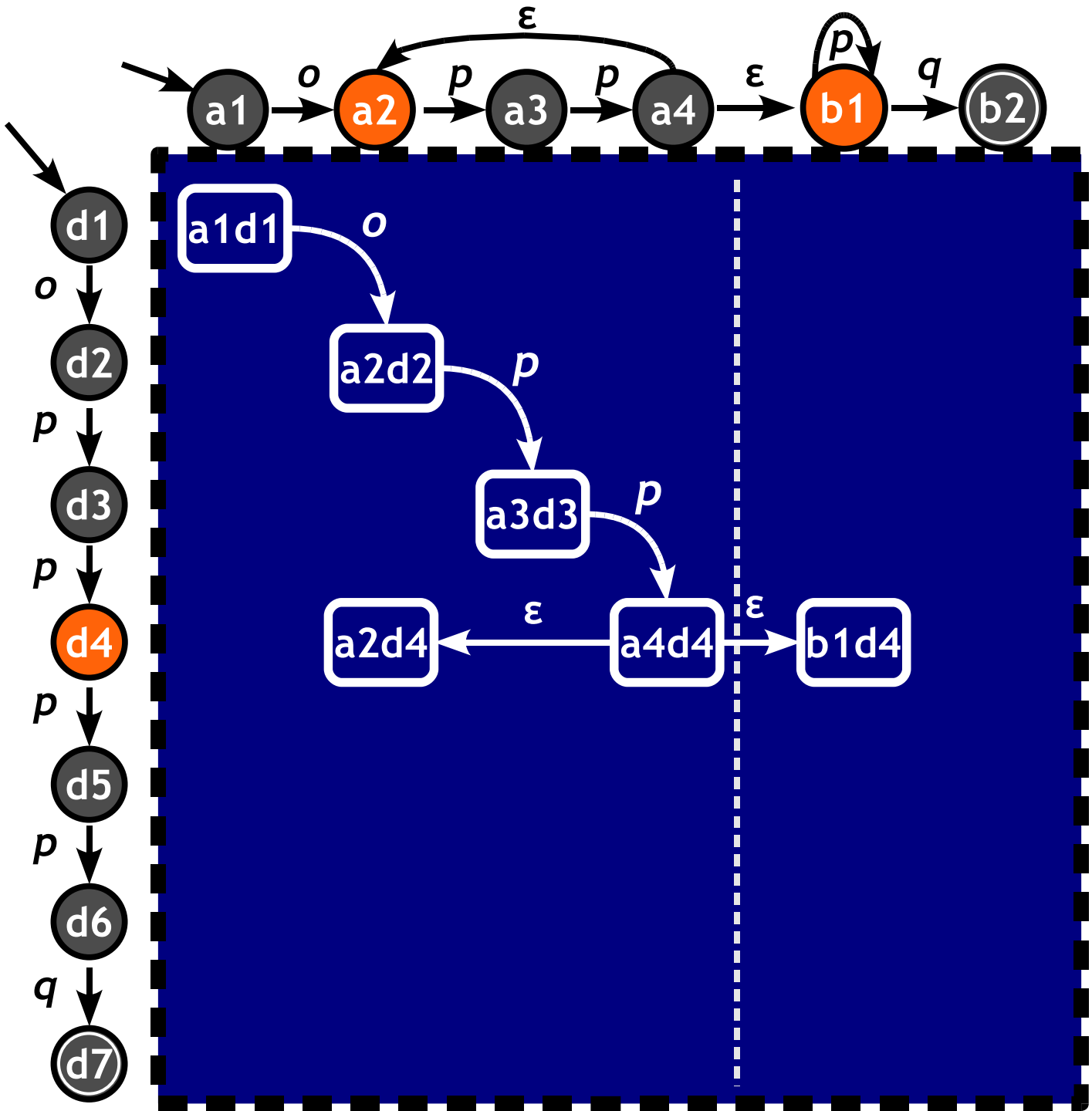


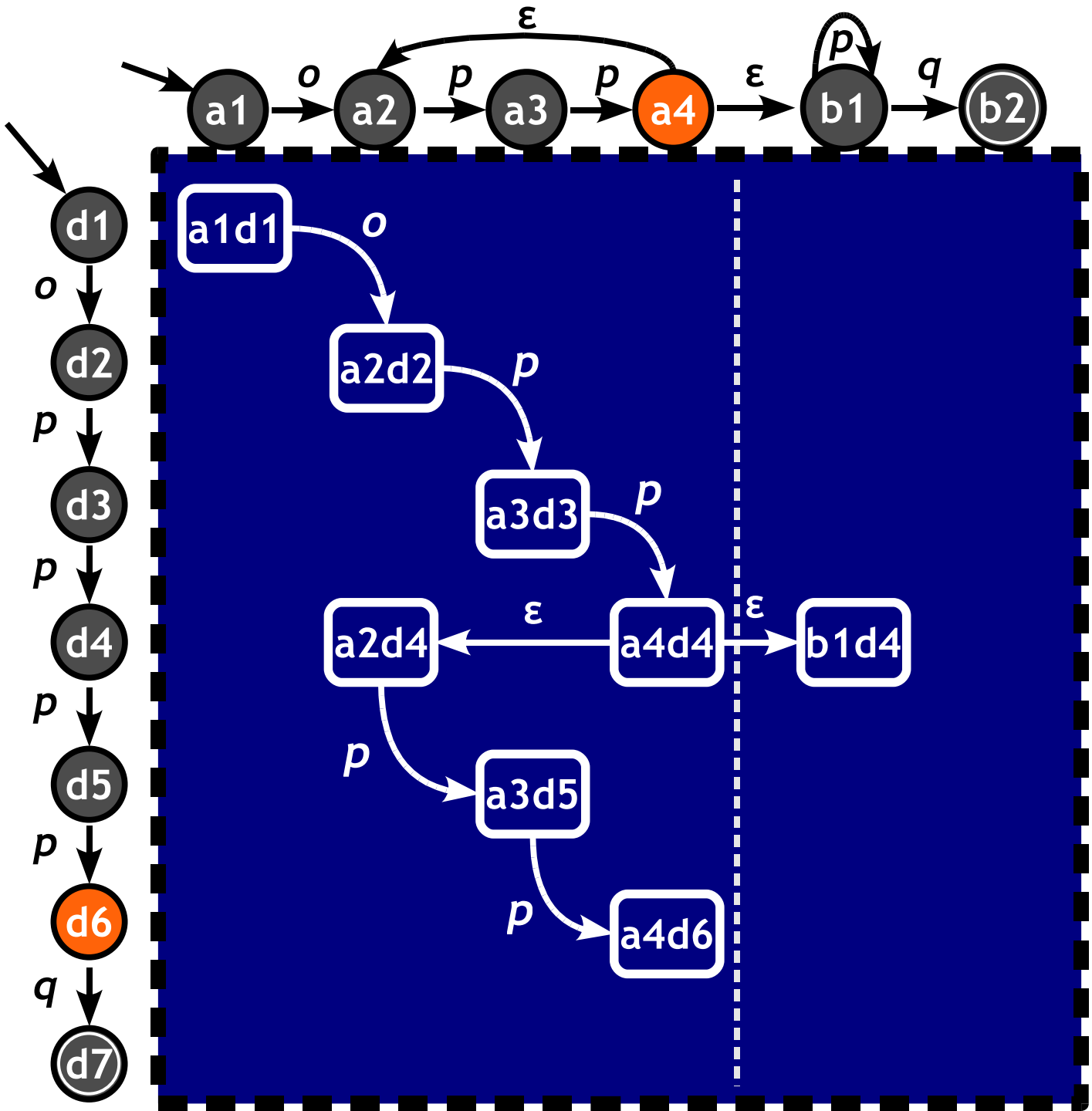

ε
o
p
p
ε
p
q
a1
a2
a3
a4
b1
b2
d1
o
d2
p
d3
p
d4
p
d5
p
d6
q
d7

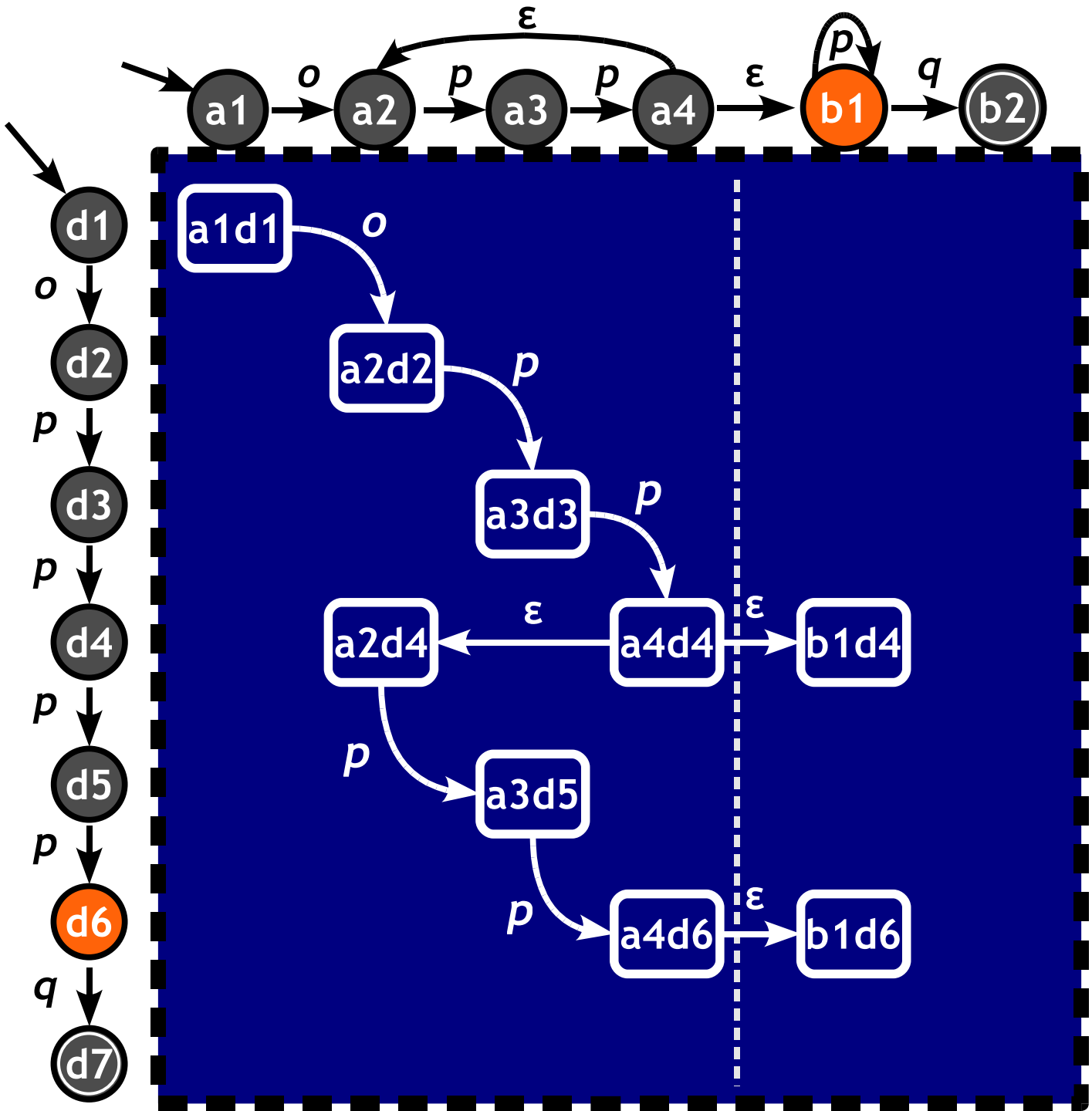

ε
o
p
p
ε
p
q
a1
a2
a3
a4
b1
b2
a1d1
d1
o
d2
p
d3
p
d4
p
d5
p
d6
q
d7

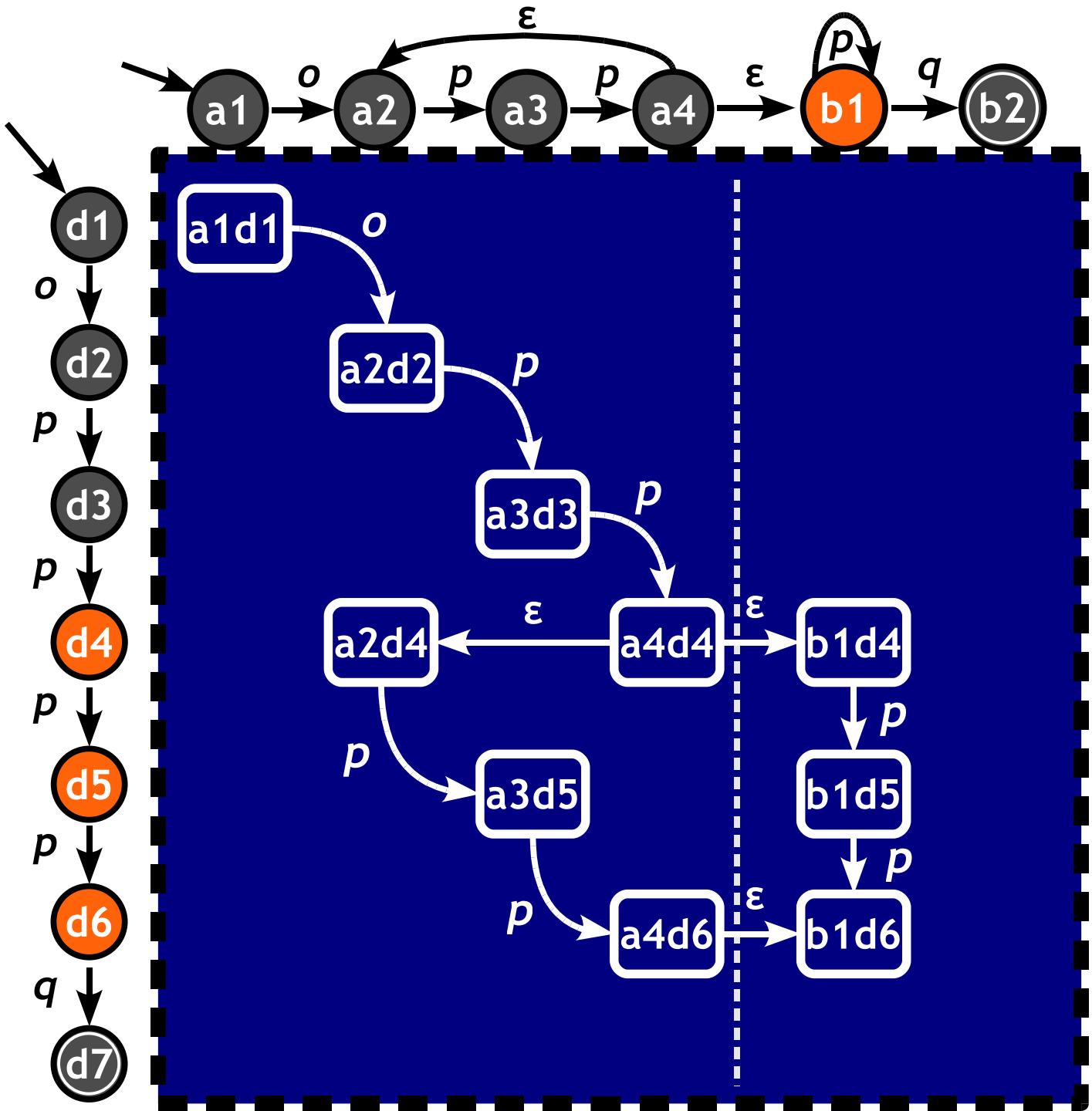

ε
o
p
p
ε
p
q
a1
a2
a3
a4
b1
b2
a1d1
o
a2d2
d1
o
d2
p
d3
p
d4
p
d5
p
d6
q
d7

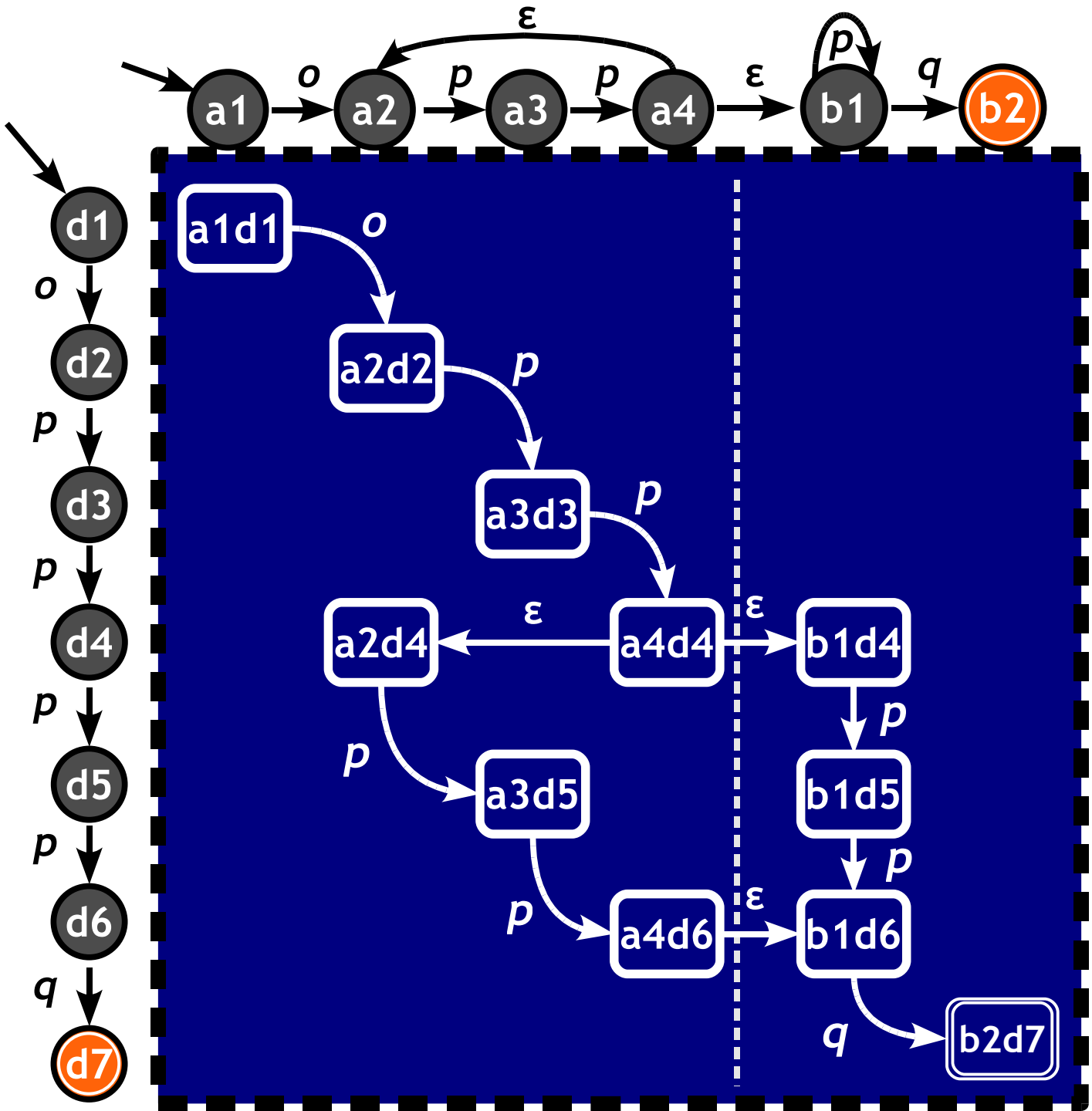

ε
o
p
p
ε
p
q
a1
a2
a3
a4
b1
b2
d1
o
d2
p
d3
p
d4
p
d5
p
d6
q
d7
a1d1
o
a2d2
p
a3d3
p
a4d4

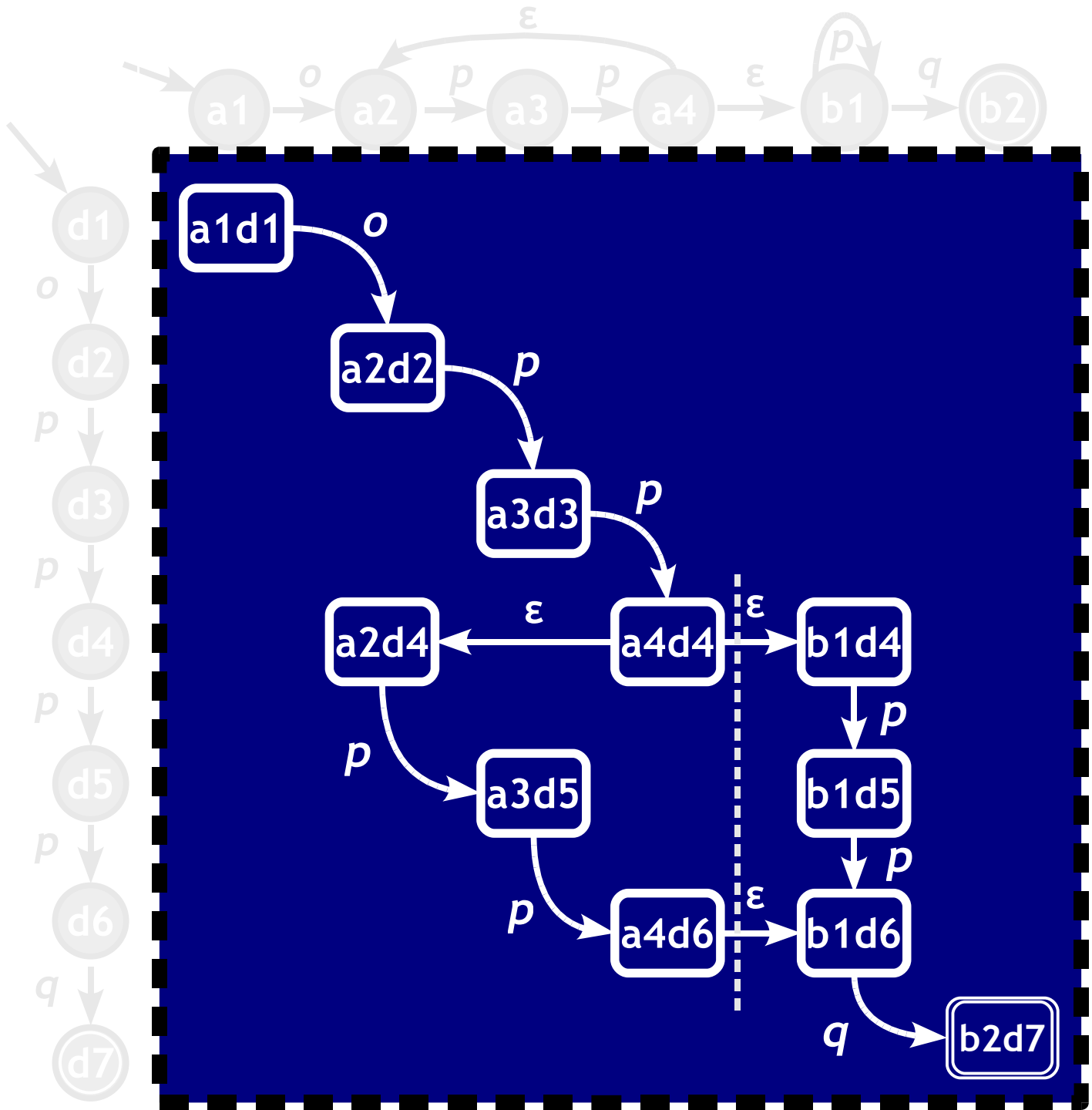

ε
o
p
p
ε
p
q
a1
a2
a3
a4
b1
b2
d1
o
d2
p
d3
p
d4
p
d5
p
d6
q
d7
a1d1
o
a2d2
p
a3d3
p
a2d4
ε
a4d4
ε
b1d4

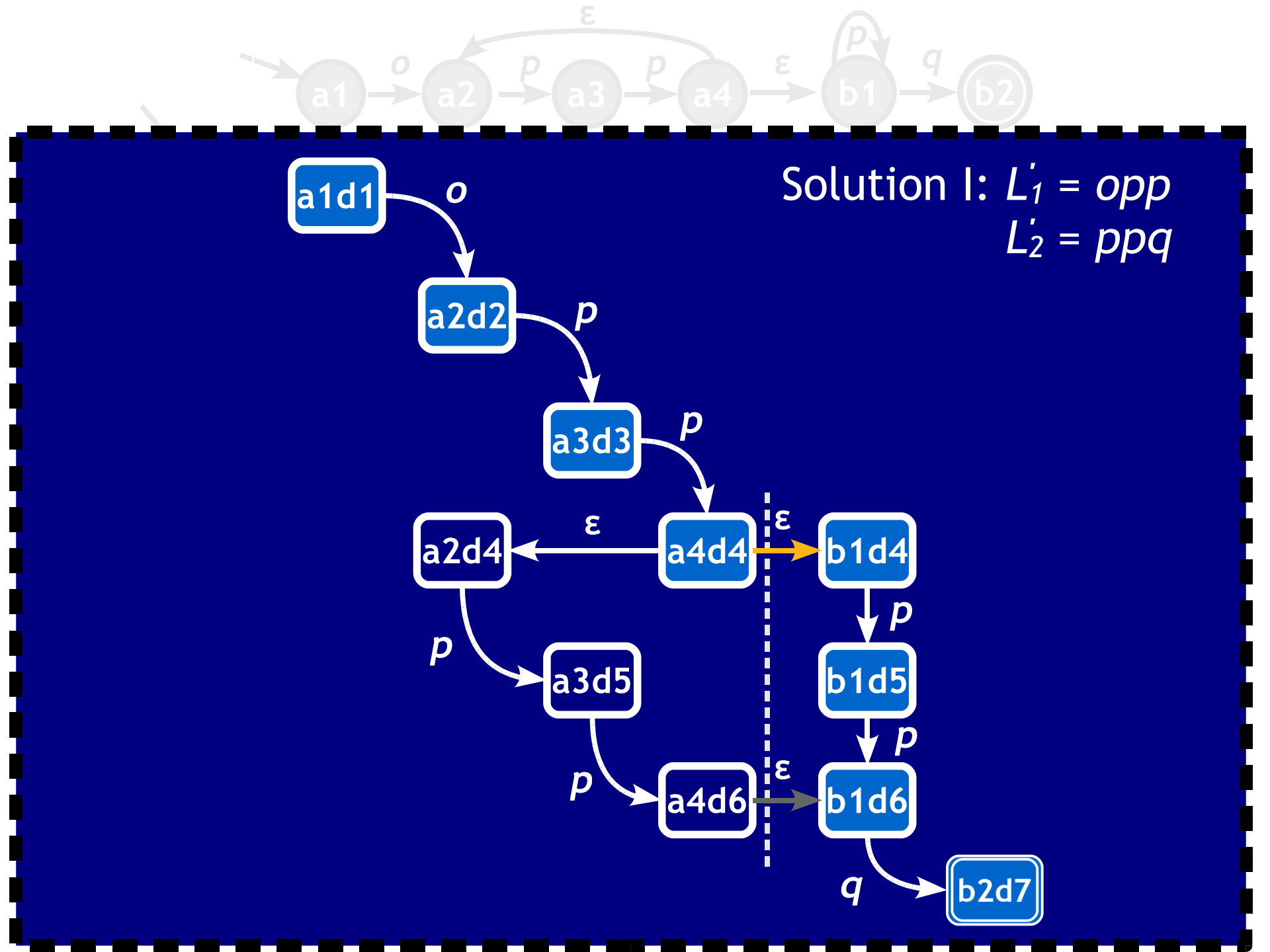

ε
o
p
p
ε
p
q
a1
a2
a3
a4
b1
b2
d1
o
d2
p
d3
p
d4
p
d5
p
d6
q
d7
a1d1
o
a2d2
p
a3d3
p
a2d4
ε
a4d4
ε
b1d4
p
a3d5
p
a4d6

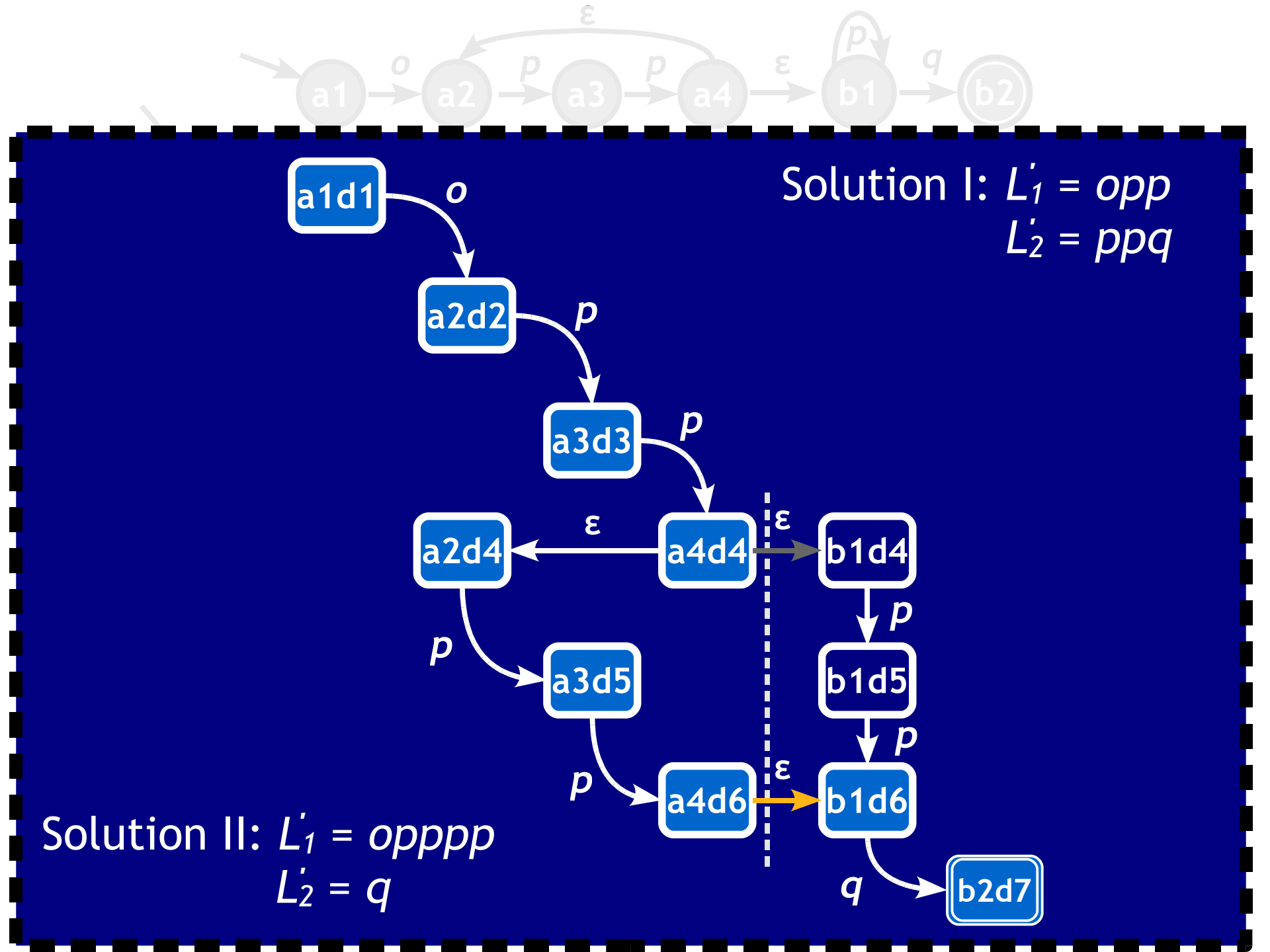

ε
o
p
p
ε
p
q
a1
a2
a3
a4
b1
b2
d1
o
d2
p
d3
p
d4
p
d5
p
d6
q
d7
a1d1
o
a2d2
p
a3d3
p
a2d4
ε
a4d4
ε
b1d4
p
a3d5
p
a4d6
ε
b1d6


ε
o
p
p
ε
p
q
a1
a2
a3
a4
b1
b2
d1
o
d2
p
d3
p
d4
p
d5
p
d6
q
d7
a1d1
o
a2d2
p
a3d3
p
ε
ε
a2d4
a4d4
b1d4
p
p
a3d5
b1d5
p
p
ε
a4d6
b1d6


ε
o
p
p
ε
p
q
a1
a2
a3
a4
b1
b2
d1
o
d2
p
d3
p
d4
p
d5
p
d6
q
d7
a1d1
o
a2d2
p
a3d3
p
ε
ε
a2d4
a4d4
b1d4
p
p
a3d5
b1d5
p
p
ε
a4d6
b1d6
q
b2d7


a1
o
a2
p
a3
p
a4
ε
b1
p
q
b2
d1
o
d2
p
d3
p
d4
p
d5
p
d6
q
d7
a1d1
o
a2d2
p
a3d3
p
a2d4
ε
a4d4
ε
b1d4
p
p
a3d5
b1d5
p
p
a4d6
ε
b1d6
q
b2d7


a1
a2
a3
a4
b1
b2
o
p
p
ε
p
q
Solution I: $L_1' = opp$
$L_2' = ppq$
a1d1
o
a2d2
p
a3d3
p
a2d4
ε
a4d4
ε
b1d4
p
b1d5
p
b1d6
q
b2d7
p
a3d5
p
a4d6
ε


a1
o
a2
p
a3
p
a4
ε
b1
p
q
b2
a1d1
o
a2d2
p
a3d3
p
a2d4
ε
a4d4
ε
b1d4
p
p
a3d5
b1d5
p
p
a4d6
ε
b1d6
q
b2d7
Solution I: $L_1' = opp$
$L_2' = ppq$
Solution II: $L_1' = opppp$
$L_2' = q$

Evaluation

Name	Version	Description	files Total	loc Total	Vulnerable
e107	0.7.5	content management	741	132,862	N/A
eve	1.0	activity tracker	8	905	3
tiger	1.0 beta 39	news management	30	6,701	0
utopia	1.3.0	news management	24	5,438	5
warp	1.2.1	content management	44	24,365	14

- Found inputs for 17/22 vulnerabilities

Min:	1s
Avg:	86s
Med:	36s
Max:	697s

Conclusion

- We presented a general constraint-solving approach for string variables
- It can find inputs for SQL injections within a reasonable time
- We used a three-stage algorithm:
    1. Generate annotated grammar
    2. Search for strings and associated paths
    3. Solving (cyclic) constraints over strings