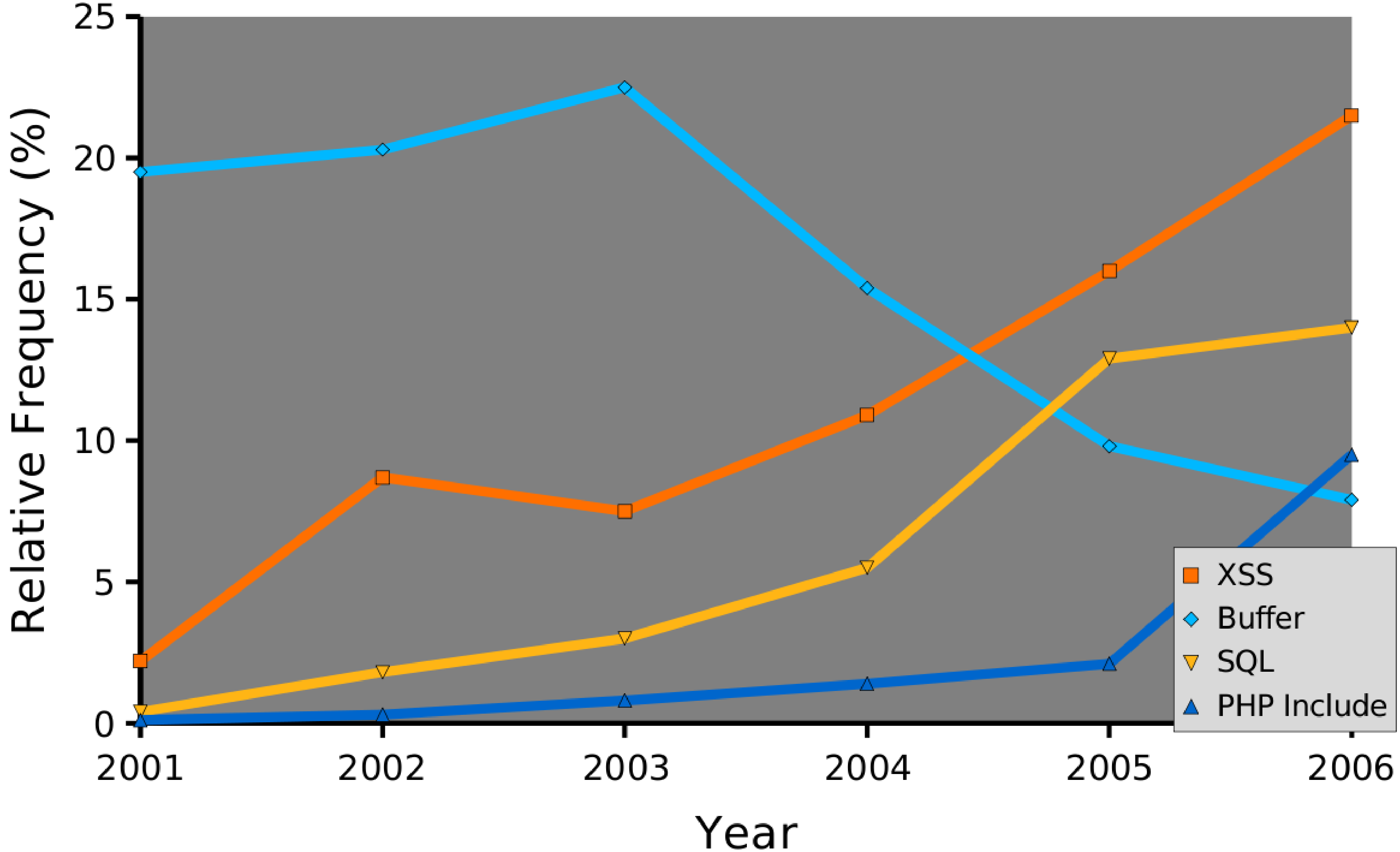
Decision Procedures for String Constraints

Ph.D. Proposal
Pieter Hooimeijer

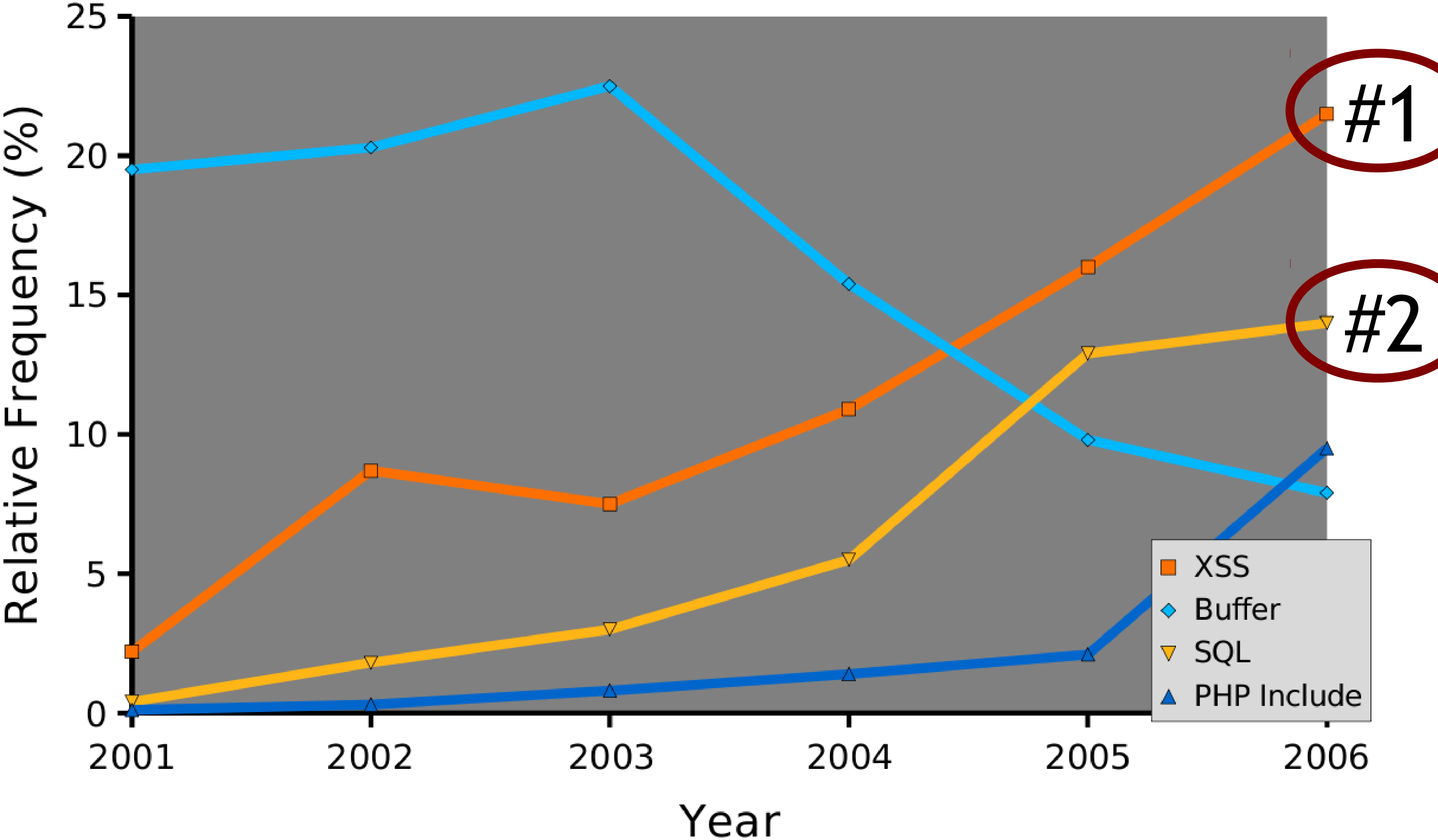
University of Virginia

Motivation



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

Motivation

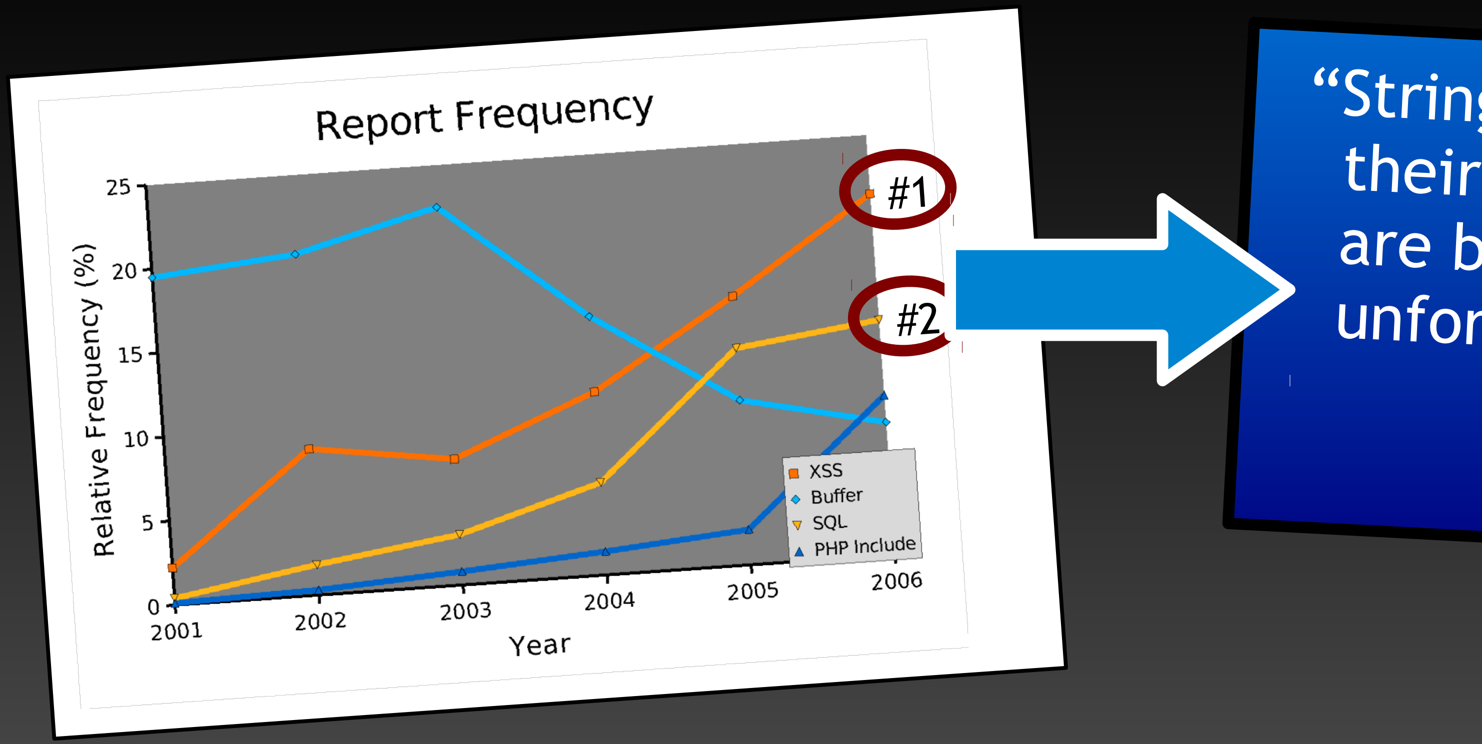


Mitre Corp. data reported on http://www.attrition.org/

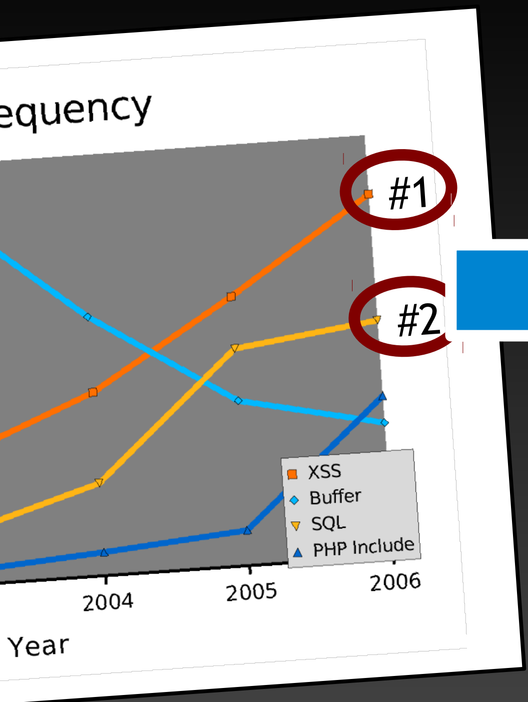
Motivation

> "String values have lost their innocence and are being used in many unforeseen contexts."
>
> [Thiemann05]

Motivation


Report Frequency
Relative Frequency (%)
Year
25
20
15
10
5
0
2001
2002
2003
2004
2005
2006
XSS
Buffer
SQL
PHP Include
#1
#2
"Strin
their
are b
unfor

Motivation



"String values have lost their innocence and are being used in many unforeseen contexts."

[Thiemann05]

Motivation

“String values have lost their innocence and are being used in many unforeseen contexts.”

[Thiemann05]

Now what?

Goal

Make string analysis available to a wider class of program analysis tools.

Outline

- String Constraint Solving
- Preliminary Results
- Proposed Research

Outline

- String Constraint Solving
  - example code
  - definitions
- Preliminary Results
- Proposed Research

Outline

- String Constraint Solving
  - example code
  - definitions
- Preliminary Results
- Proposed Research

Example

```
// v1 and v2 are user inputs

if (!ereg('o(pp)+', v1)){exit;}
if (!ereg('p*q', v2)){exit;}

v3 = v1 . v2; // concat
if (v3 != 'oppppq'){exit;}
magic();
```

Query:

Will this code ever execute *magic*?

Example

```
// v1 and v2 are user inputs

(1) if (!ereg('o(pp)+', v1)){exit;}
(2) if (!ereg('p*q', v2)){exit;}

(3) v3 = v1 . v2; // concat
if (v3 != 'opppq'){exit;}
magic();
```

Outline

- String Constraint Solving
  - example code
  - definitions
- Preliminary Results
- Proposed Research

Outline

- String Constraint Solving
  - example code
  - definitions
- Preliminary Results
- Proposed Research

Definitions

String Constraint

$$C ::= E \in R \mid E \notin R$$

$$E ::= V \mid E \circ V$$

R : regex

V : variable

Definitions

Constraint System

$S = \{ C_1, \ldots, C_n \}$

where each $C_i \in S$ is a well-formed string constraint.

Definitions

Decision Procedure

D : constraint system → { Satisfiable, Unsatisfiable }

Definitions

Soundness

$[D(S) = Sat.] \rightarrow$
S is sat.

Completeness

S is sat. $\rightarrow$
$[D(S) = Sat.]$

Definitions

Soundness

$[D(S) = Sat.] \rightarrow$ S is sat.

Completeness

S is sat. $\rightarrow$ $[D(S) = Sat.]$

Definitions

Constraint System

$S = \{ C_1, \ldots, C_n \}$
where each $C_i \in S$ is a well-formed string constraint.

Decision Procedure

D : constraint system → { Satisfiable, Unsatisfiable }

String Constraint

$C ::= E \in R \mid E \notin R$

$E ::= V \mid E \circ V$

R : regex

V : variable

Soundness

$[D(S) = Sat.] \rightarrow S$ is sat.

Completeness

S is sat. $\rightarrow [D(S) = Sat.]$

Definitions

Constraint System

$S = \{ C_1, \ldots, C_n \}$
where each $C_i \in S$ is a well-formed string constraint.

Decision Procedure

D : constraint system → { Satisfiable, Unsatisfiable }

String Constraint

$C ::= E \in R \mid E \notin R$

$E ::= V \mid E \circ V$

R : regex

V : variable

Soundness

$[D(S) = Sat.] \rightarrow S$ is sat.

Completeness

S is sat. $\rightarrow [D(S) = Sat.]$

Definitions

Constraint System

$S = \{ C_1, \ldots, C_n \}$
where each $C_i \in S$ is a well-formed string constraint.

Decision Procedure

D : constraint system → { Satisfiable, Unsatisfiable }

Soundness

$[D(S) = Sat.] \rightarrow S$ is sat.

String Constraint

$C ::= E \in R \mid E \notin R$

$E ::= V \mid E \circ V$

R : regex

V : variable

Completeness

S is sat. $\rightarrow [D(S) = Sat.]$

Outline

- String Constraint Solving
  - example code
  - definitions
- Preliminary Results
- Proposed Research

Existing Tools

DPRLE [PLDI09]	Automata
Hampi [ISSTA09]	Encode to STP
Rex [ICST10]	Encode to Z3
Kaluza [Oakland10]	Encode to Hampi & STP
Our Prototype	Lazy Automata

Questions

Make string analysis available to a wider class of program analysis tools.

Questions

- What is acceptable performance?
- What type of constraints should we allow?

Outline

- String Constraint Solving
- Preliminary Results
  - scalability
  - expressive utility
- Proposed Research

Scalability

Subjects:

- Decision Procedure for Regular
  Language Equations [PLDI09]
- Hampi [ISSTA09]
- Lazy Prototype

Scalability

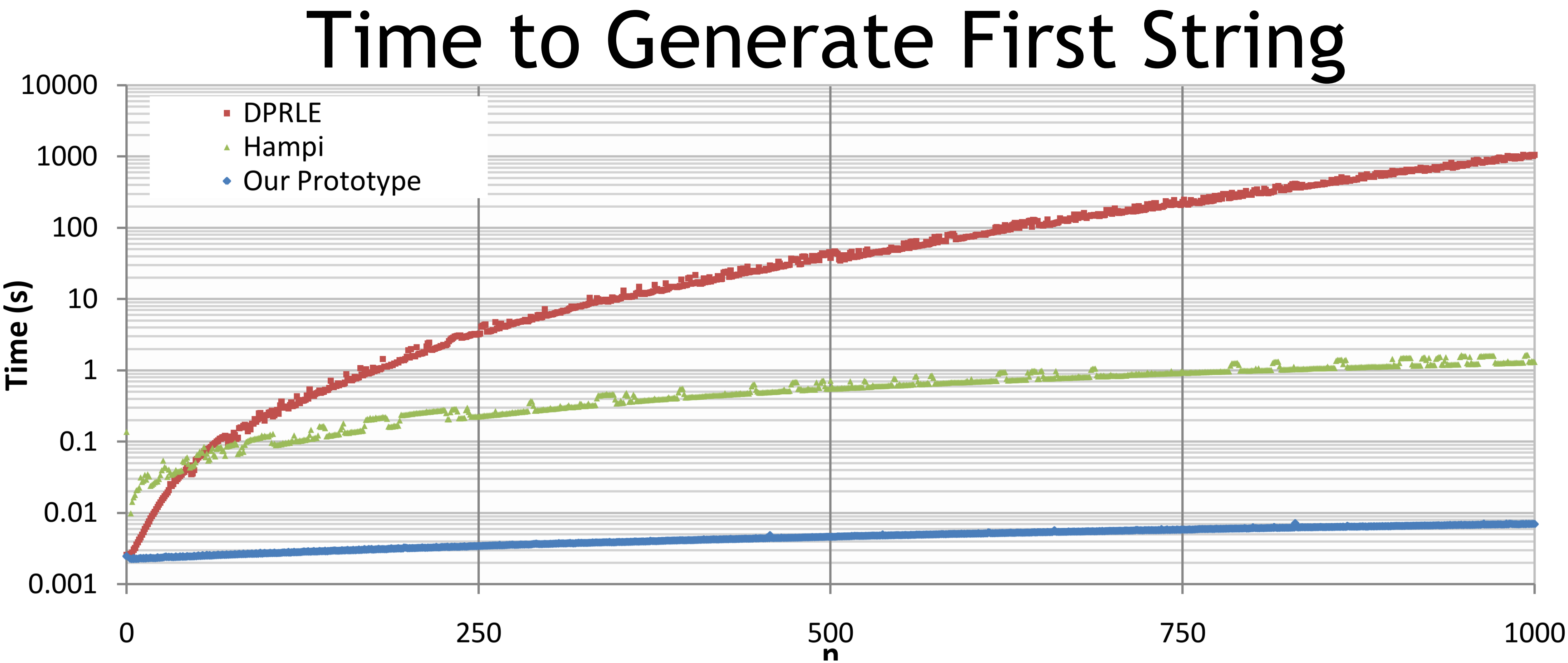
Task: find a string that is in both

`[a-c]*a[a-c]{n+1}`

and

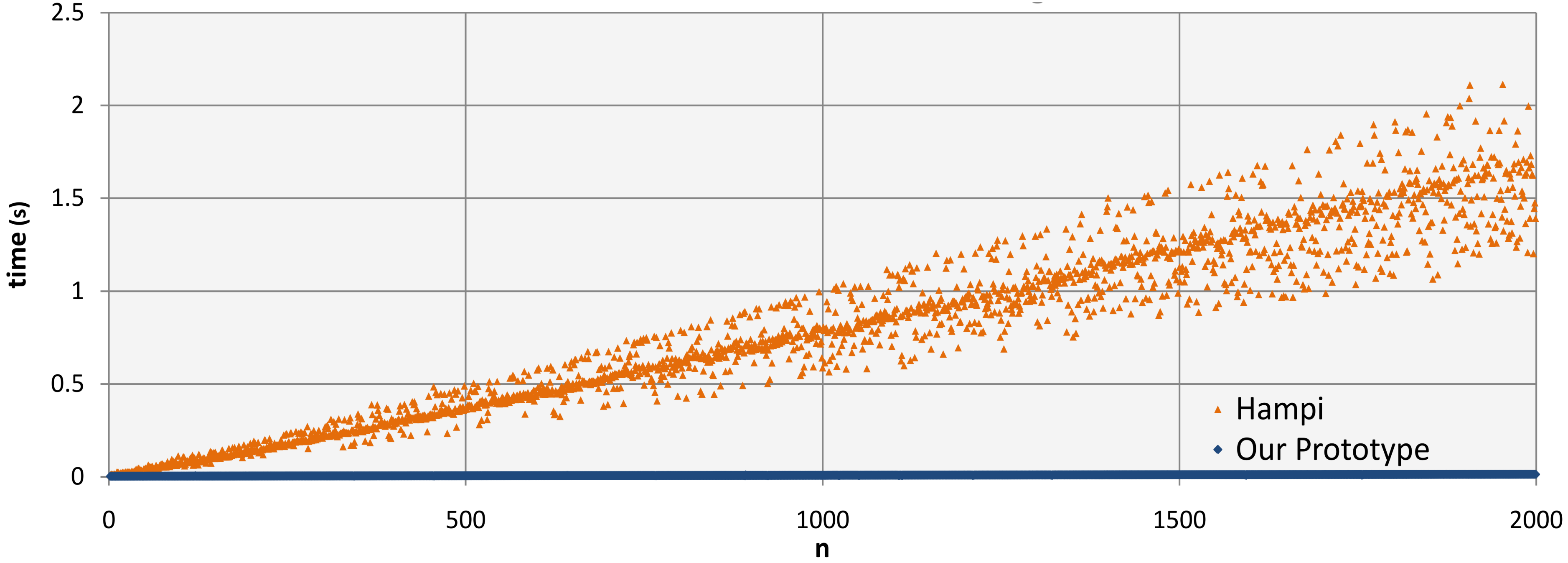
`[a-c]*b[a-c]{n}`

Scalability



Scalability

Time to Generate First String



Scalability

- Existing approaches are less scalable than they could be on the tested benchmarks
- Interaction with an underlying solver introduces performance artifacts

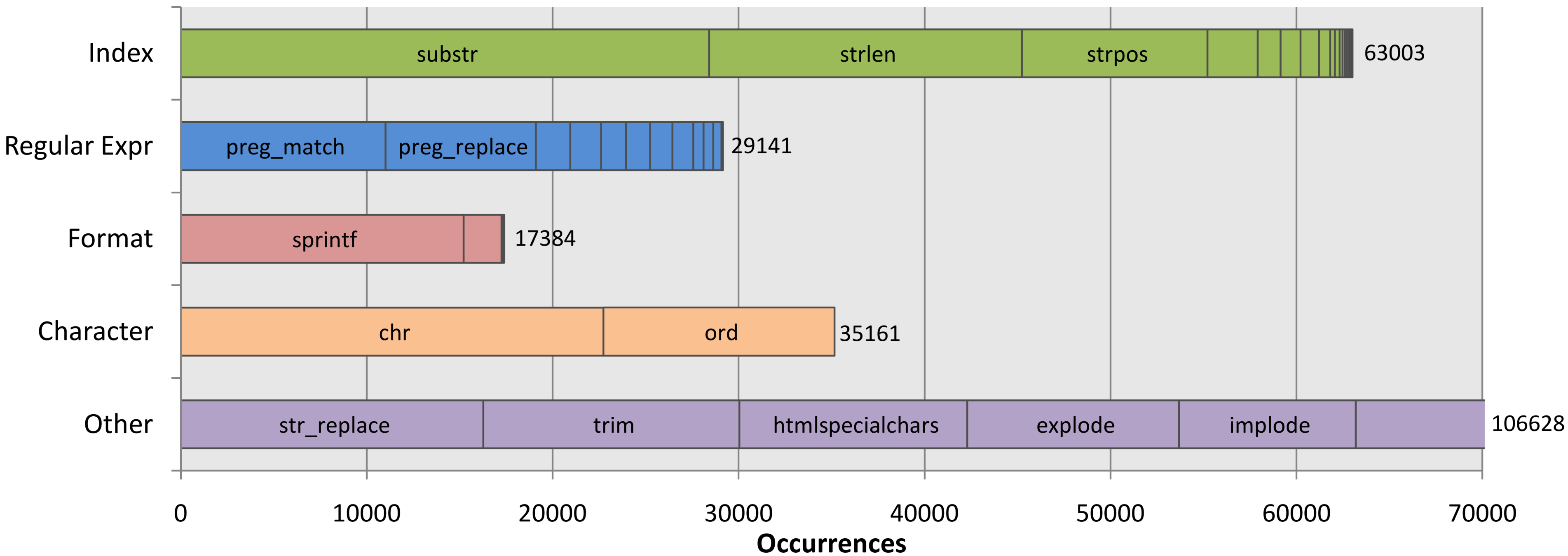
Outline

- String Constraint Solving
- Preliminary Results
  - scalability
  - expressive utility
- Proposed Research

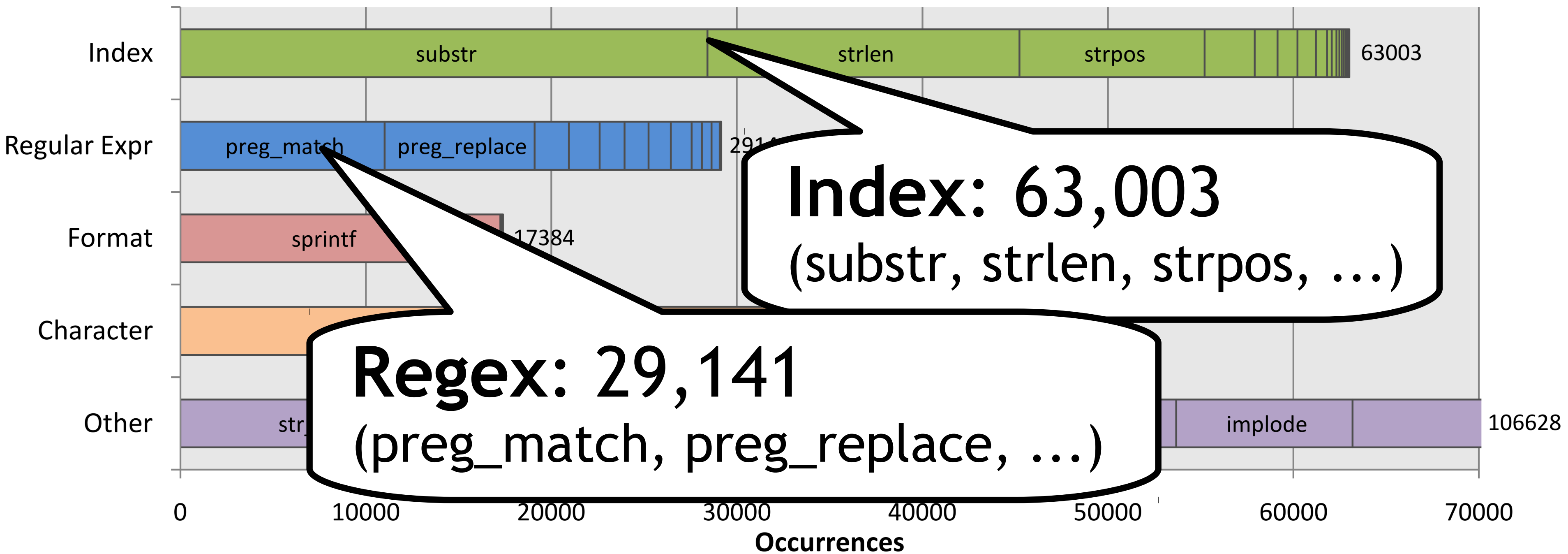
Expressive Utility

- Picked 88 PHP projects on SourceForge = 9.6 million LOC
- Tally:111 distinct string functions

Expressive Utility



Expressive Utility



Expressive Utility

- Existing approaches typically support 'Regex,' but not 'Index' operations

- 'Index' operations were 2x as common in the sample under study

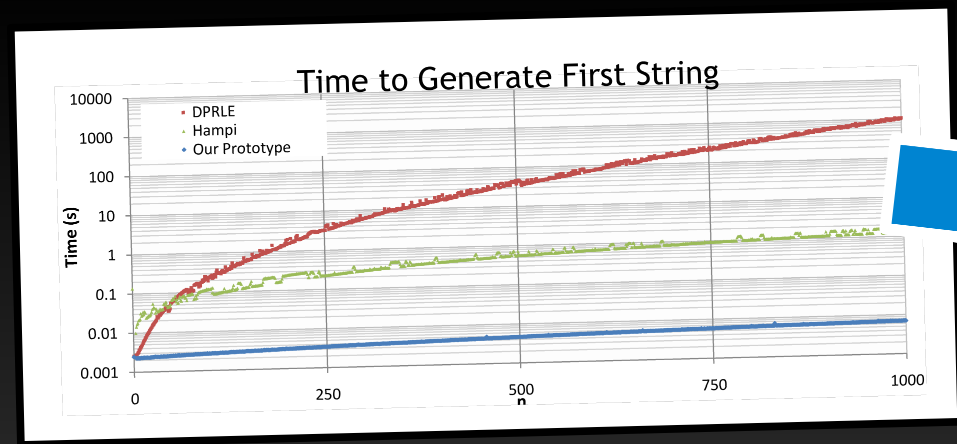
Outline

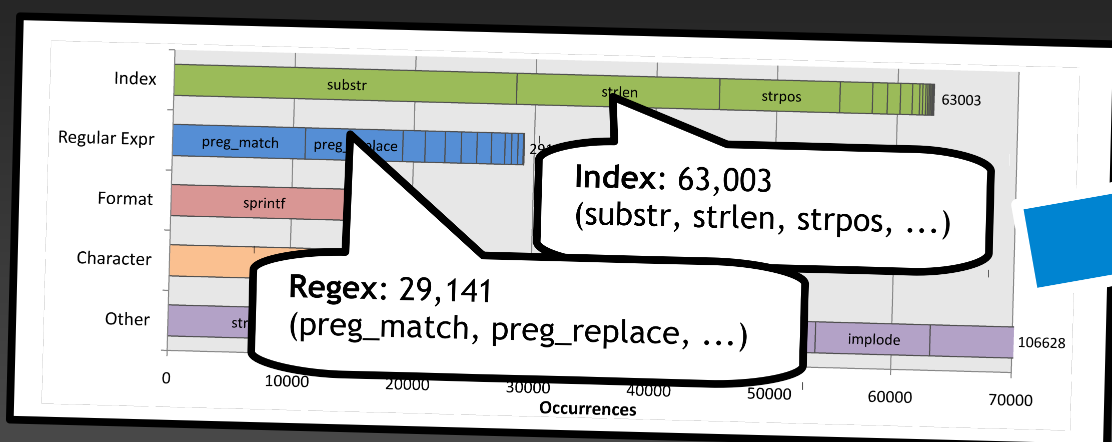
- String Constraint Solving
- Preliminary Results
  - scalability
  - expressive utility
- Proposed Research

Outline

- String Constraint Solving
- Preliminary Results
- Proposed Research
  - subset constraints
  - scalability through laziness
  - integer index operations
  - proof strategies

Thesis Statement





It is possib
practical a
the satisfia
cover both
operations
program ar
admits a m
of correctn

Thesis Statement

It is possible to construct a practical algorithm that decides the satisfiability of constraints that cover both string and integer index operations, scales up to real-world program analysis problems, and admits a machine-checkable proof of correctness.

Outline

- String Constraint Solving
- Preliminary Results
- Proposed Research
  - subset constraints
  - scalability through laziness
  - integer index operations
  - proof strategies

Subset Constraints [PLDI'09]

$$
\begin{aligned}
S &::= E \subseteq C \\
E &::= E \circ E \\
&\quad\mid\ C \\
&\quad\mid\ V
\end{aligned}
$$

concatenation

constants

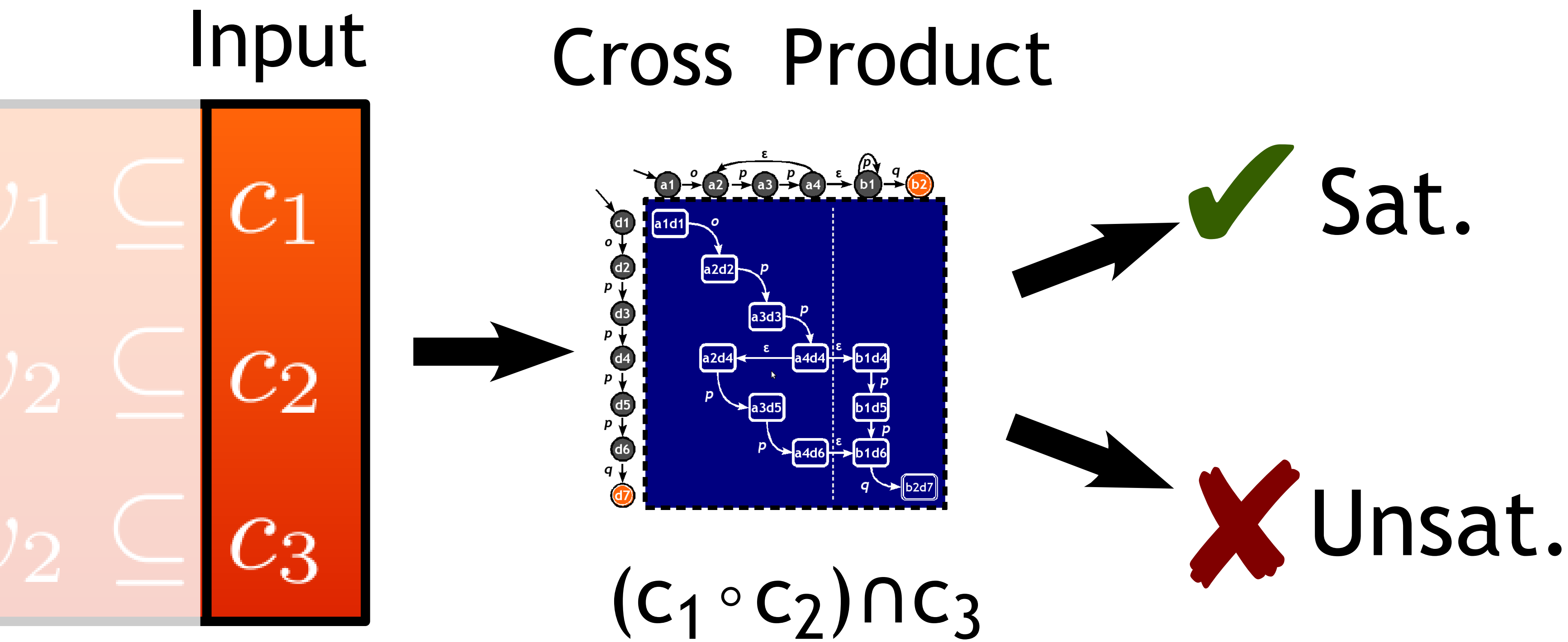
variables

Approach

Input

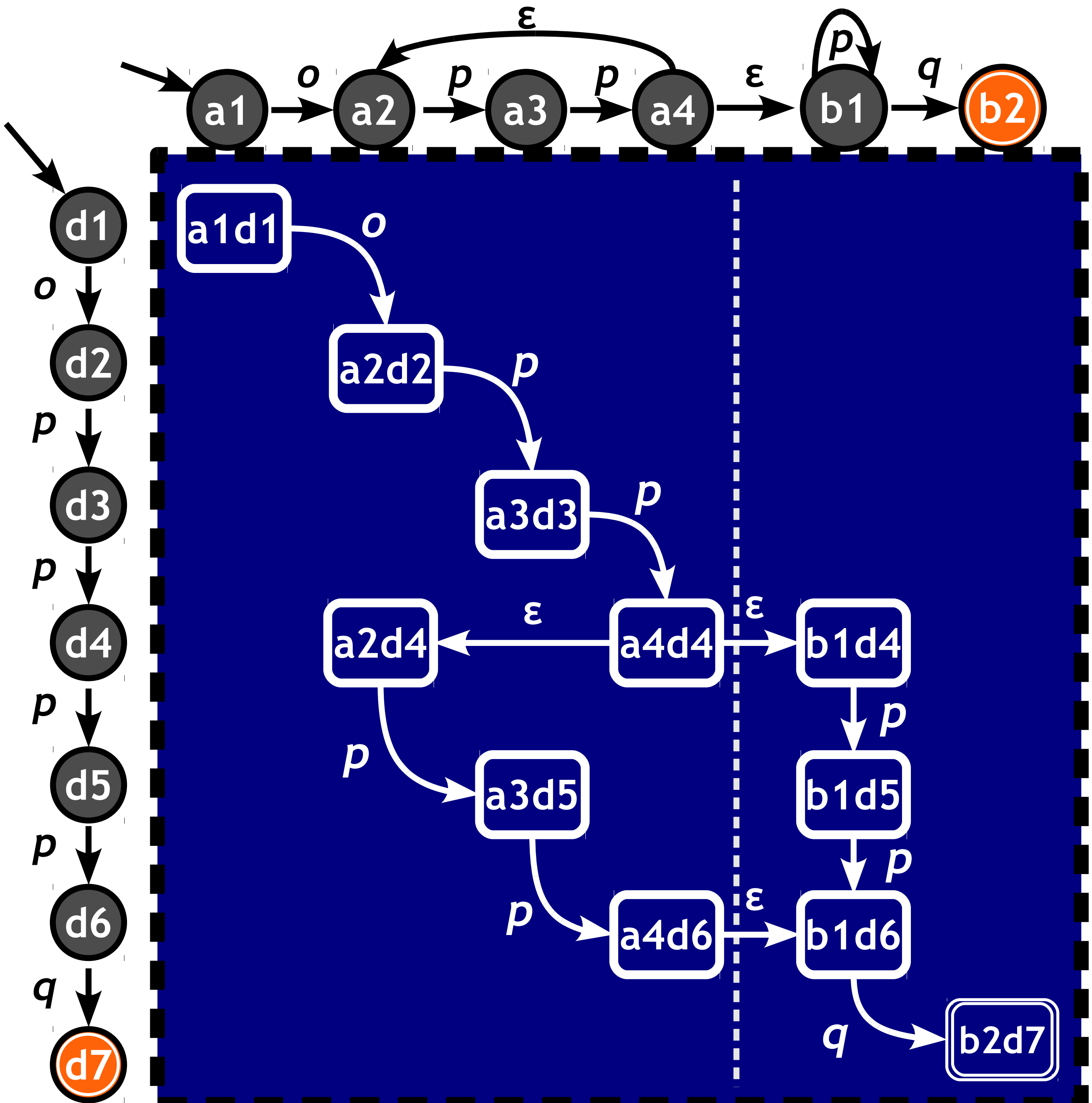
1	$v_1 \subseteq$	c_1
2	$v_2 \subseteq$	c_2
3	$v_1 \circ v_2 \subseteq$	c_3

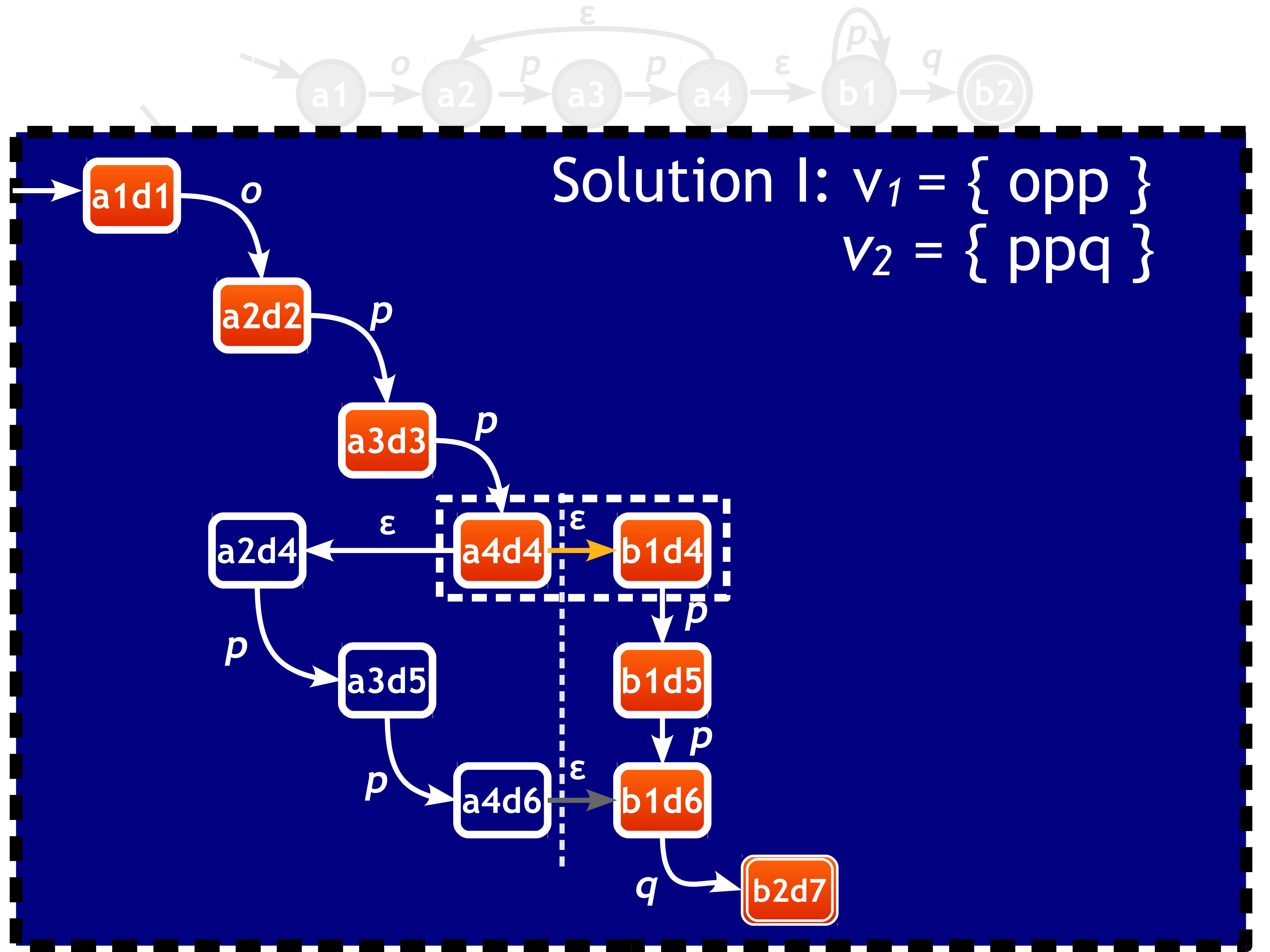
Approach



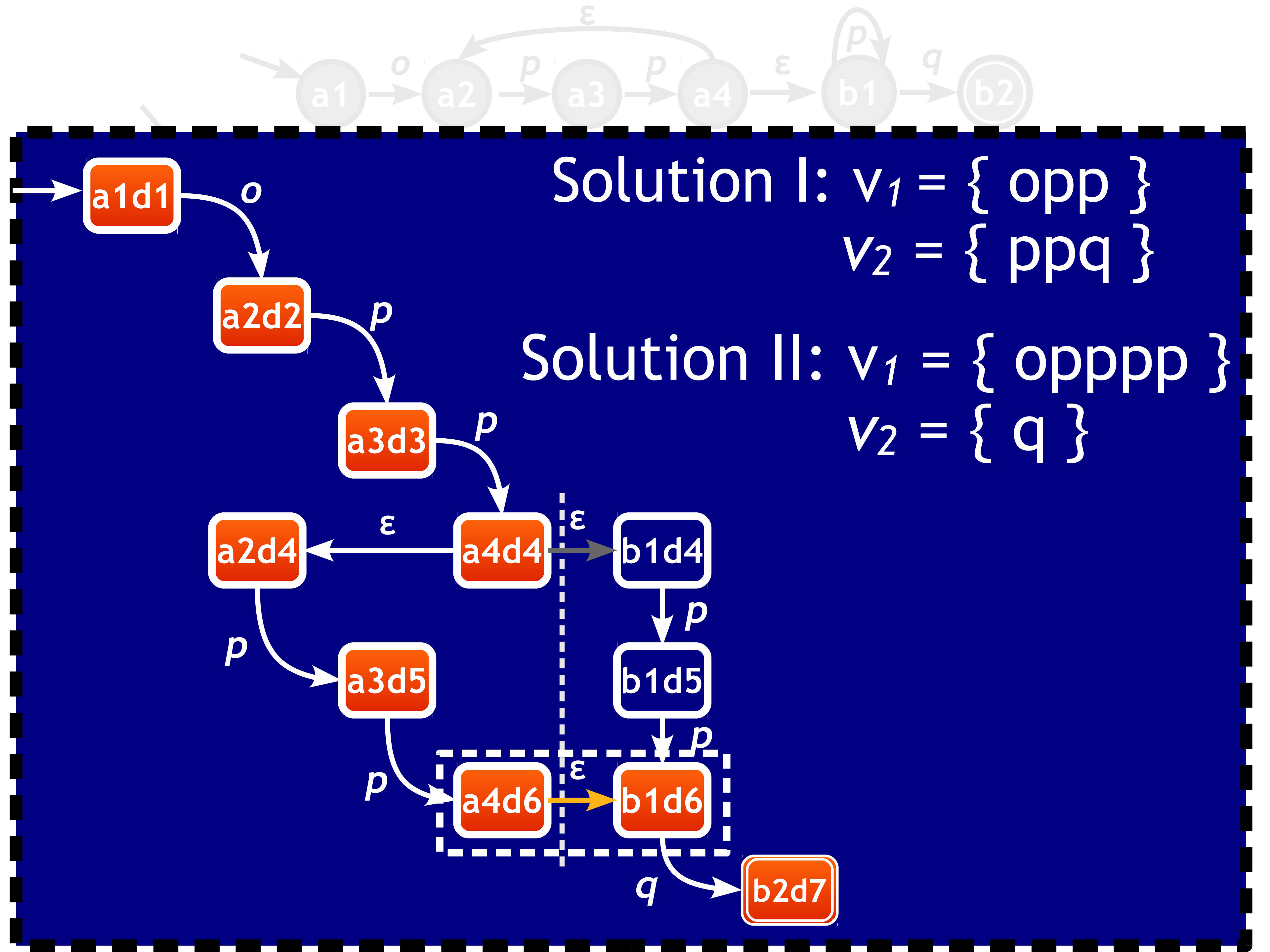
Example

```
// v1 and v2 are user inputs

if (!ereg('o(pp)+', v1)){exit;}
if (!ereg('p*q', v2)){exit;}

v3 = v1 . v2; // concat
if (v3 != 'opppq'){exit;}
magic();
```


ε
p
a1
o
a2
p
a3
p
a4
ε
b1
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
ε
a4d6
b1d6
q
b2d7



Solution I: v_1 = { opp }
v_2 = { ppq }
a1
a2
a3
a4
b1
b2
o
p
p
ε
ε
p
q
a1d1
a2d2
a3d3
a4d4
b1d4
a2d4
a3d5
a4d6
b1d5
b1d6
b2d7
o
p
p
ε
ε
p
p
p
p
ε
q


a1
o
a2
p
a3
p
a4
ε
ε
b1
p
q
b2
Solution I: v_1 = { opp }
v_2 = { ppq }
Solution II: v_1 = { opppp }
v_2 = { q }
a1d1
o
a2d2
p
a3d3
p
a2d4
ε
a4d4
ε
b1d4
p
p
a3d5
b1d5
p
p
a4d6
ε
b1d6
q
b2d7

Algorithms and a Proof

- Concat-Intersect (CI) algorithm:
  - two variables, three constants; fixed form
  - mechanically verified proof in Coq 8.1pl3
  - proof size is ~1300 lines
- Regular Matching Assignments (RMA):
  - implemented in a tool, DPRPLE
  - applies *CI* procedure inductively

Evaluation

- Find SQL injection vulnerabilities [Wassermann and Su; PLDI07]
- For each vulnerability:
  - generate SQL + program path
  - check path consistency (Simplify)
  - solve string constraints (DPRLE)

Outline

- String Constraint Solving
- Preliminary Results
- Proposed Research
  - subset constraints
  - scalability through laziness
  - integer index operations
  - proof strategies

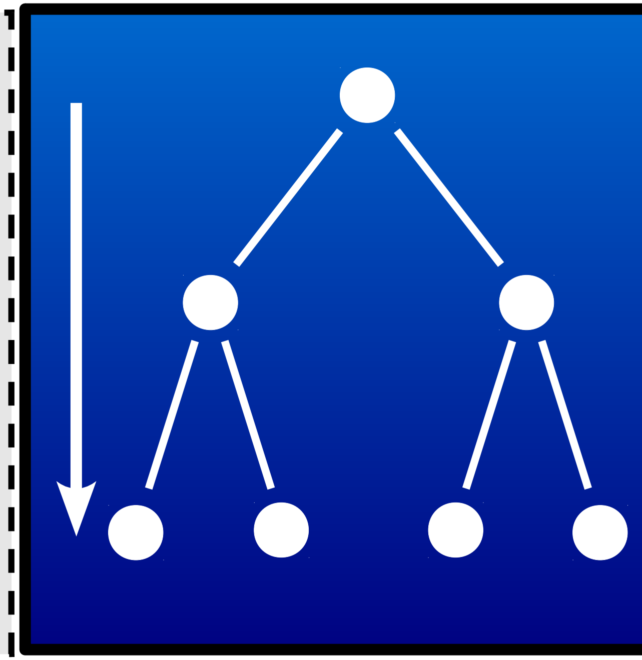
Scalability through Laziness

Idea:

Cast constraint solving as a search problem.
Traverse as little of the search space as possible.

Proposed Approach

```
datatype searchstate =
 { next   : variable;
   states : variable→pos→status}
datatype status =
 | Unknown  of status
 | StartsAt of nfastate→status
 | Path     of nfapath→status
```



Proposed Evaluation

- Within-domain performance comparison:
  - DPRLE
  - Hampi
  - CFG Analyzer
  - Rex

- Use previously-published benchmarks:
  - long strings task [Veanes *et al.*]
  - set difference task [Veanes *et al.*]
  - grammar intersection task [Kiezun *et al.*]

Outline

- String Constraint Solving
- Preliminary Results
- Proposed Research
  - subset constraints
  - scalability through laziness
  - integer index operations
  - proof strategies

Integer Index Operations

Idea:

Extend the lazy search-based approach to support integer index operations. Make use (if possible) of existing integer arithmetic models that support incremental solving.

Proposed Approach

- Explicitly-typed constraint language for strings and integer indices
- Support integer arithmetic on indices using an existing approach

Proposed Evaluation

- Compare to existing approach [Saxena *et al.*] where features overlap
- Develop PHP benchmark based on preliminary results
- Metrics: running time, proportion of testcases fully expressible

Outline

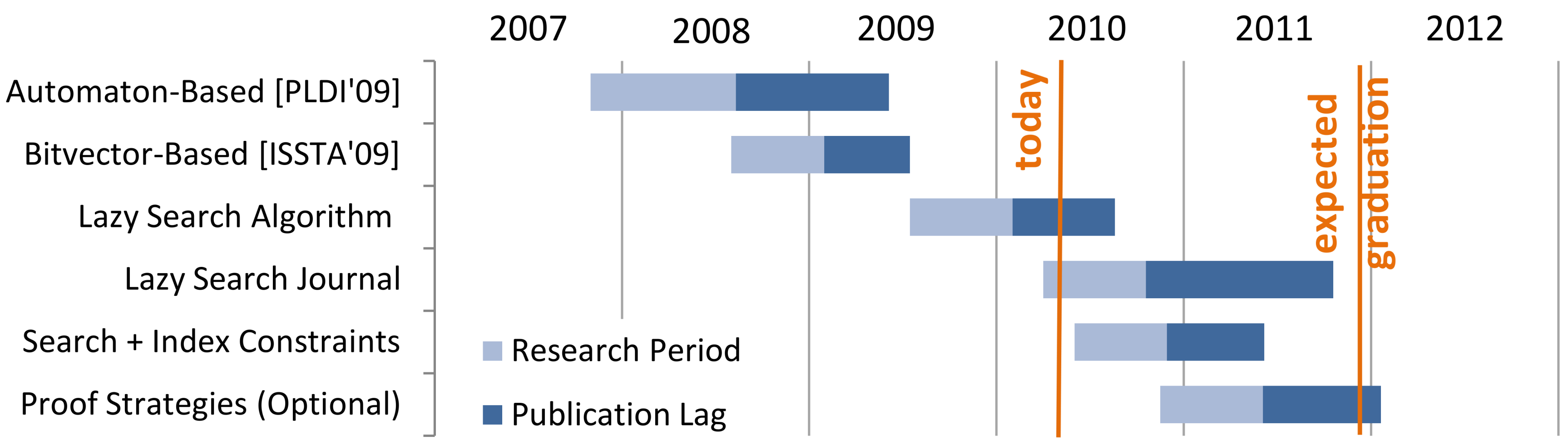
- String Constraint Solving
- Preliminary Results
- Proposed Research
  - subset constraints
  - scalability through laziness
  - integer index operations
  - proof strategies

Proof Strategies

Idea:

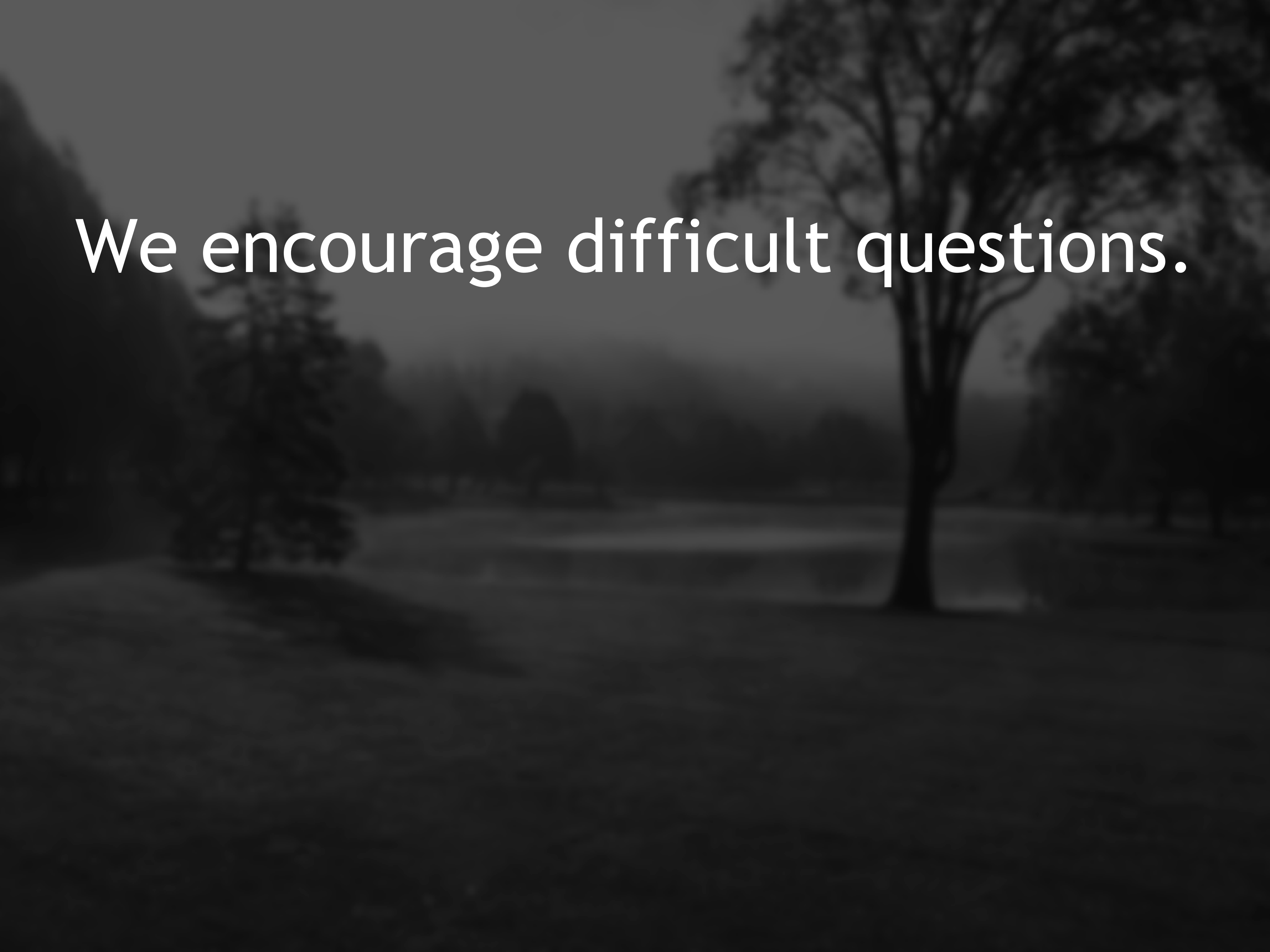
Develop a more general approach for formally verifying string decision procedures so that proof and algorithm can co-evolve.

Schedule


2007
2008
2009
2010
2011
2012
Automaton-Based [PLDI'09]
Bitvector-Based [ISSTA'09]
Lazy Search Algorithm
Lazy Search Journal
Search + Index Constraints
Proof Strategies (Optional)
today
expected graduation
Research Period
Publication Lag

Conclusion

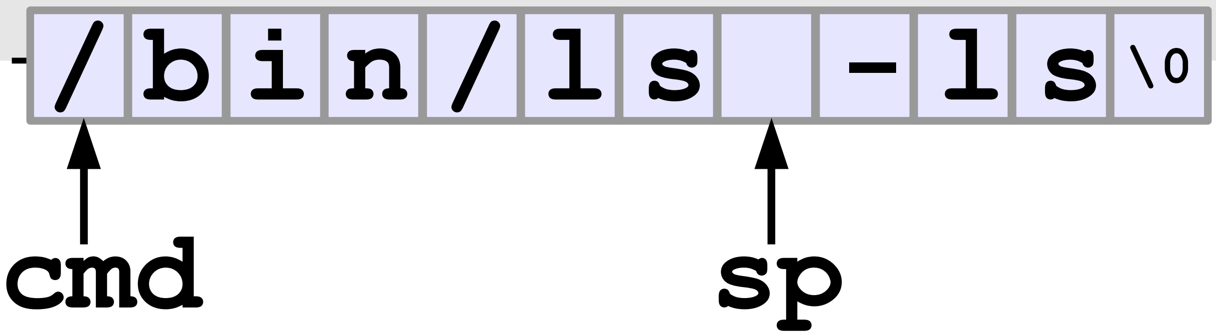
- Presented proposed research on decision procedures, focusing on:
  - expressive utility
  - scalability
  - correctness
- Research thrusts:
  - subset constraints
  - lazy search
  - integer index operations
  - proof strategies


We encourage difficult questions.

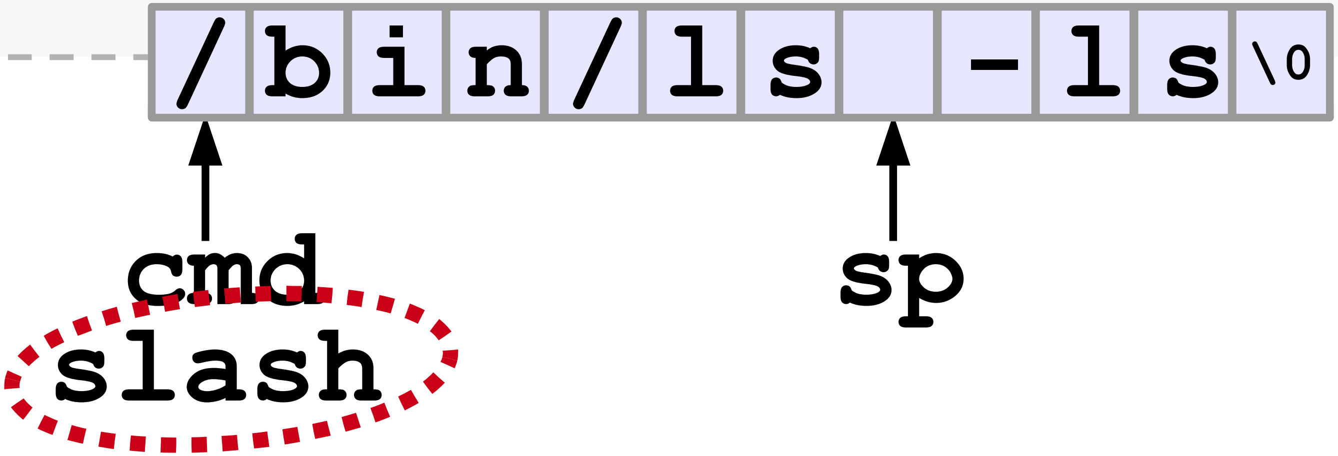
An Example

```
void site_exec(char *cmd){
char *slash;
char *sp = (char*)strchr(cmd,' ');
/* sanitize the command-string */
 while (sp &&
         (slash=strchr(cmd,'/')) &&
          (slash < sp))
    cmd = slash + 1;
```

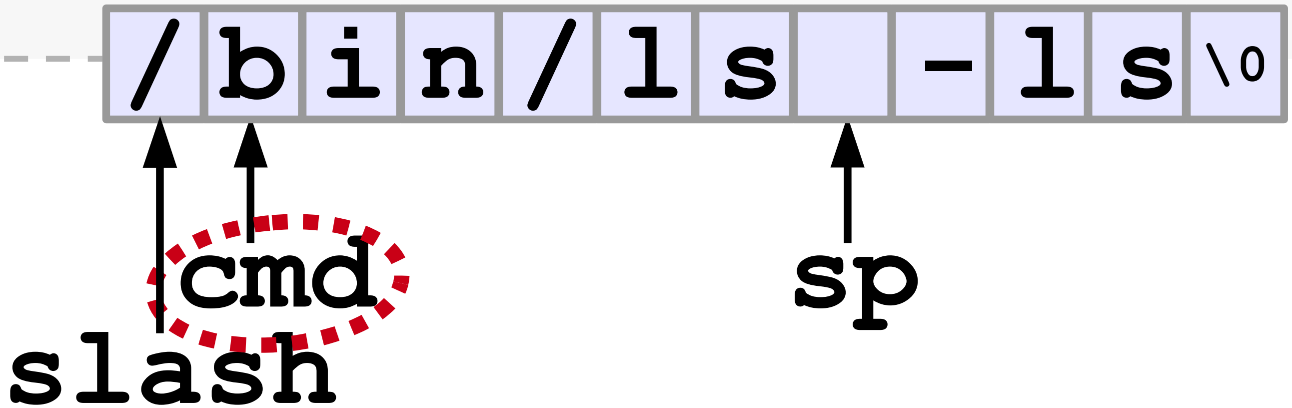
```
/* sanitize the command-string */
 while (sp &&
        (slash=strchr(cmd,'/')) &&
         (slash < sp))
    cmd = slash + 1;
```



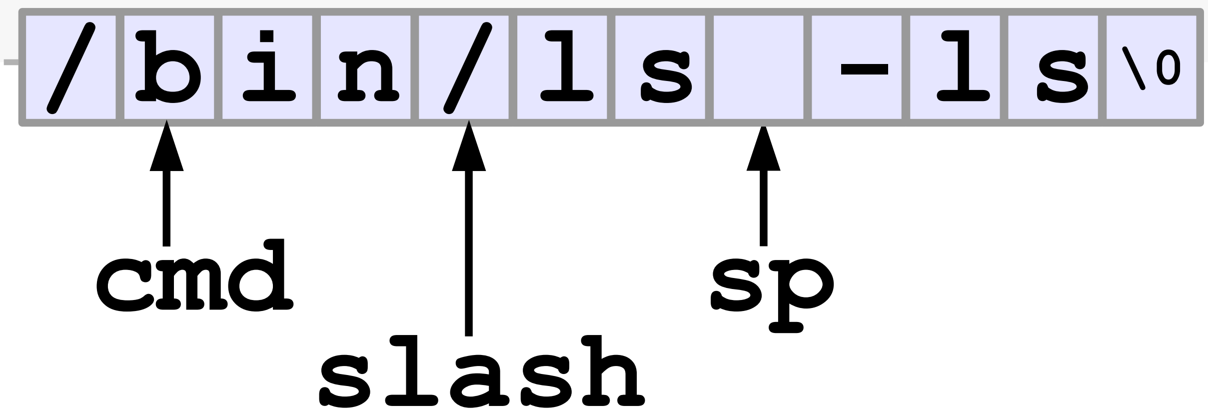
```
/* sanitize the command-string */
 while (sp &&
         (slash=strchr(cmd,'/')) &&
          (slash < sp))
    cmd = slash + 1;
```



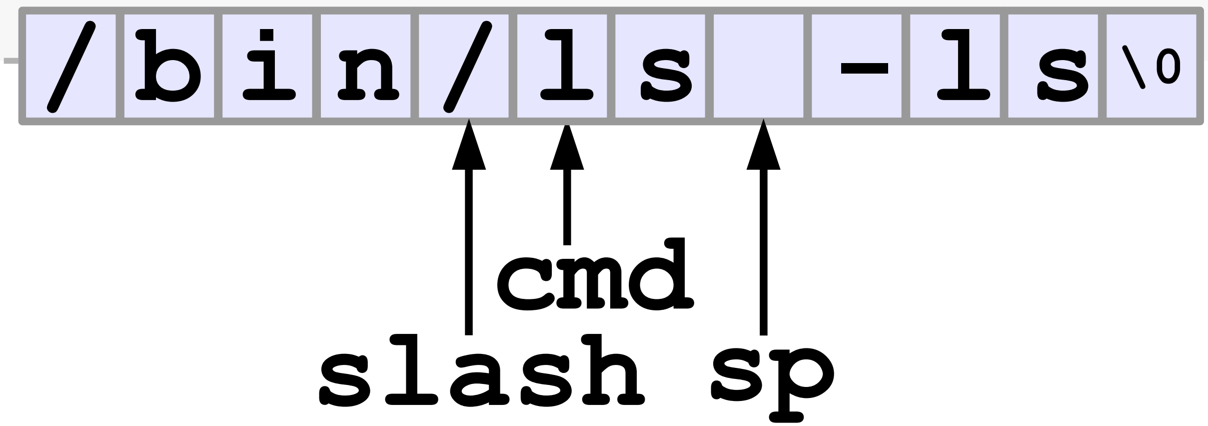
```
/* sanitize the command-string */
 while (sp &&
         (slash=strchr(cmd,'/')) &&
          (slash < sp))
    cmd = slash + 1;
```



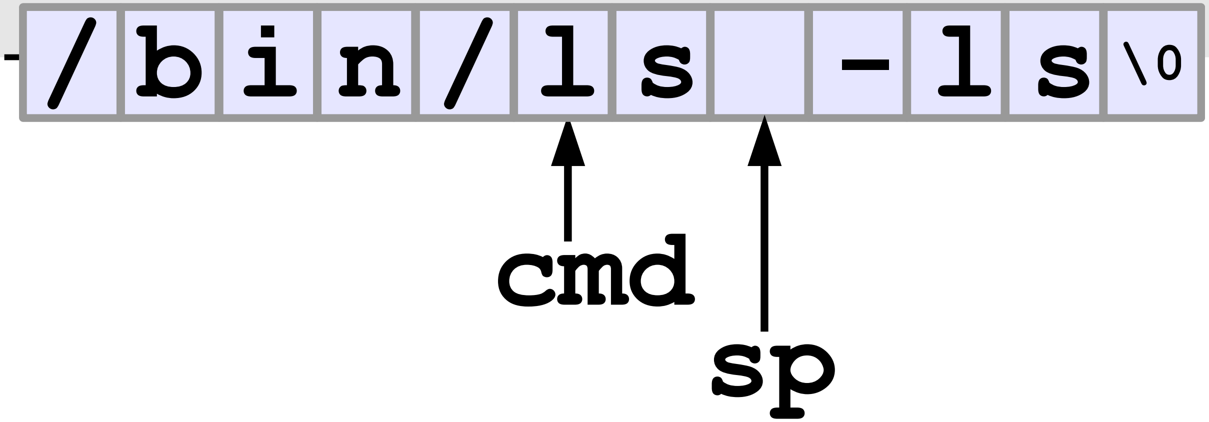
```
/* sanitize the command-string */
 while (sp &&
         (slash=strchr(cmd,'/')) &&
          (slash < sp))
    cmd = slash + 1;
```



```
/* sanitize the command-string */
 while (sp &&
         (slash=strchr(cmd,'/')) &&
          (slash < sp))
    cmd = slash + 1;
```



```
/* sanitize the command-string */
 while (sp &&
         (slash=strchr(cmd,'/')) &&
          (slash < sp))
    cmd = slash + 1;
```



slash=0

```
char *sp = (char*)strchr(cmd,' ');
/* sanitize the command-string */
 while (sp &&
         (slash=strchr(cmd,'/')) &&
          (slash < sp))
    cmd = slash + 1;
```

string c	$\in \Sigma^*$
index sp	:= findfirst(cmd, ' ');
string c2	:= cmd[:sp]
index slsh	:= findlast(cmd2, '/')
string c3	:= cmd[slash + 1:]

Example: Some Queries

Can *cmd* contain '/' ? ✓

Can the substring between *cmd* and *sp* contain '/bin/rm' ? ✗