

Solvers for Theories of Strings

Vijay Ganesh, Adam Kiezun
Shay Artzi, Philip Guo, Pieter Hooimeijer, Michael Ernst
MIT
Monday June 13, 2011

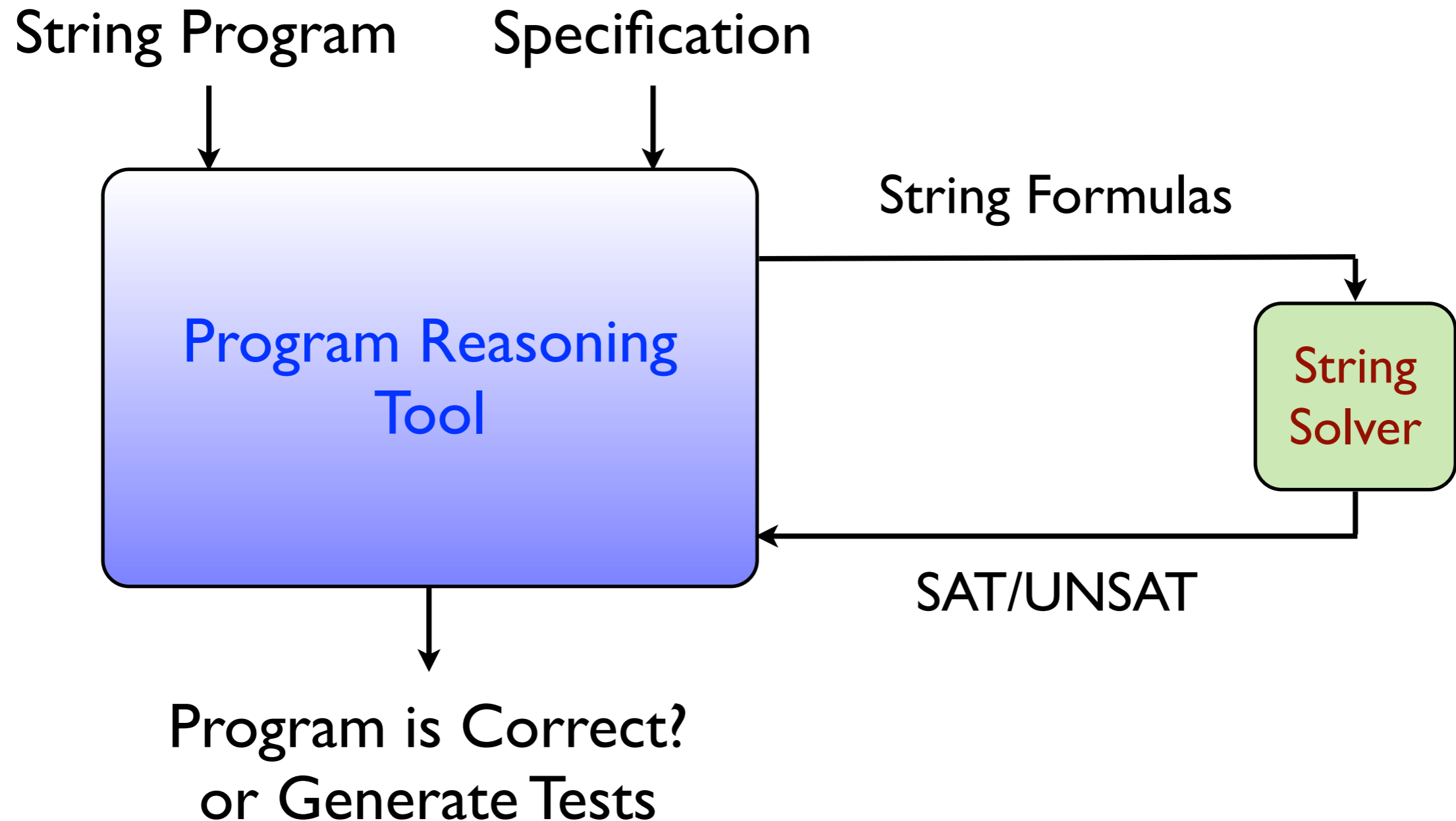
Problem Statement

Efficient Solver for Analysis of String Programs

<u>Common String Operations</u>	<u>String Programs</u>	<u>Types of Errors</u>
<p><u>Functions</u> String concatenation String extraction</p> <p><u>Predicates</u> String comparison String assignment Sanity checking of strings using RE</p>	<p><u>Traditional Apps</u> C/C++/Java Apps (Java String Library) C#/.NET</p> <p><u>Web Apps</u> Sanitization code in PHP, JavaScript Client-side and server-side Scripting code</p>	<p><u>Memory-related Errors</u> Buffer overflow Code injection</p> <p><u>Improper Sanitization</u> SQL injection XSS scripting Incomplete sanity checking</p>

Problem Statement

Efficient Solver for Analysis of String Programs



HAMPI String Solver



- $X = \text{concat}(\text{"SELECT.."}, v) \text{ AND } (X \in \text{SQL_grammar})$
- JavaScript, PHP, ... string expressions
- NP-complete

Rest of the Talk

- **HAMPI Logic**: A Theory of Strings
- Motivating Example: **HAMPI**-based Vulnerability Detection App
- How **HAMPI** works
- Experimental Results
- Related Work
- HAMPI 2.0

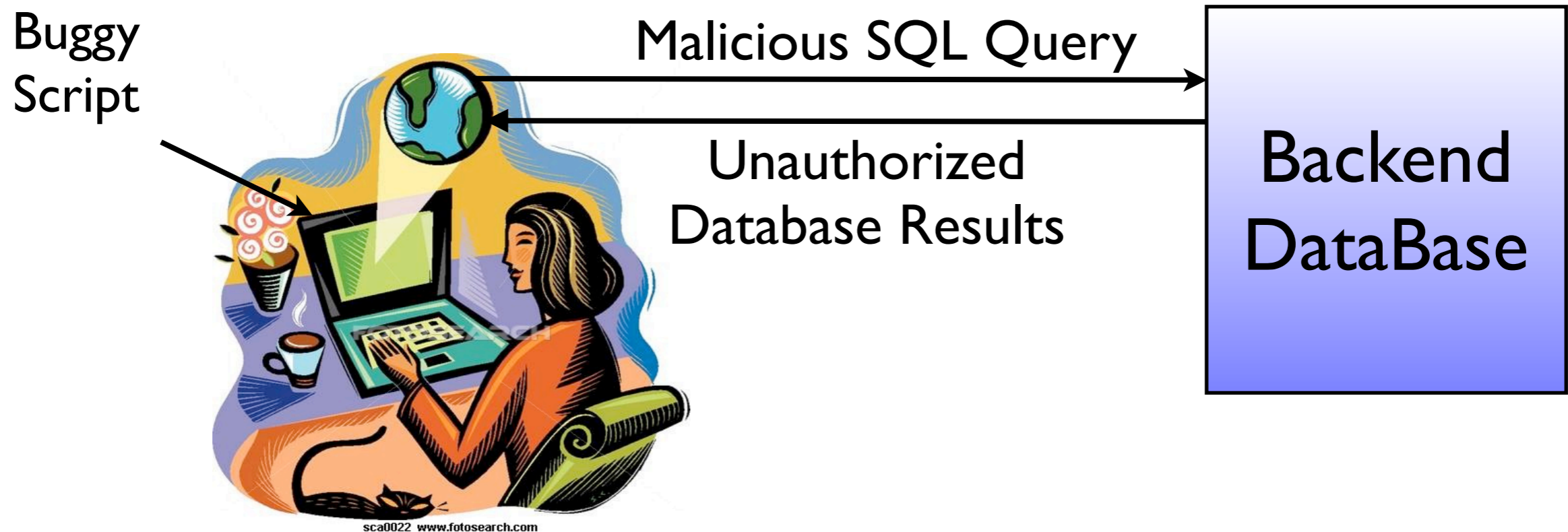
Theory of Strings

The Hampi Language

<u>PHP/JavaScript/C++...</u>	<u>HAMPI: Theory of Strings</u>	<u>Notes</u>
Var a; \$a = 'name'	Var a : 1...20; a = 'name'	Bounded String Variables String Constants
string_expr." is "	concat(string_expr," is ");	Concat Function
substr(string_expr,1,3)	string_expr[1:3]	Extract Function
assignments/strcmp a = string_expr; a /= string_expr;	equality a = string_expr; a /= string_expr;	Equality Predicate
Sanity check in regular expression RE Sanity check in context-free grammar CFG	string_expr in RE string_expr in SQL string_expr NOT in SQL	Membership Predicate
string_expr contains a sub_str string_expr does not contain a sub_str	string_expr contains sub_str string_expr NOT?contains sub_str	Contains Predicate (Substring Predicate)

HAMPI Solver Motivating Example

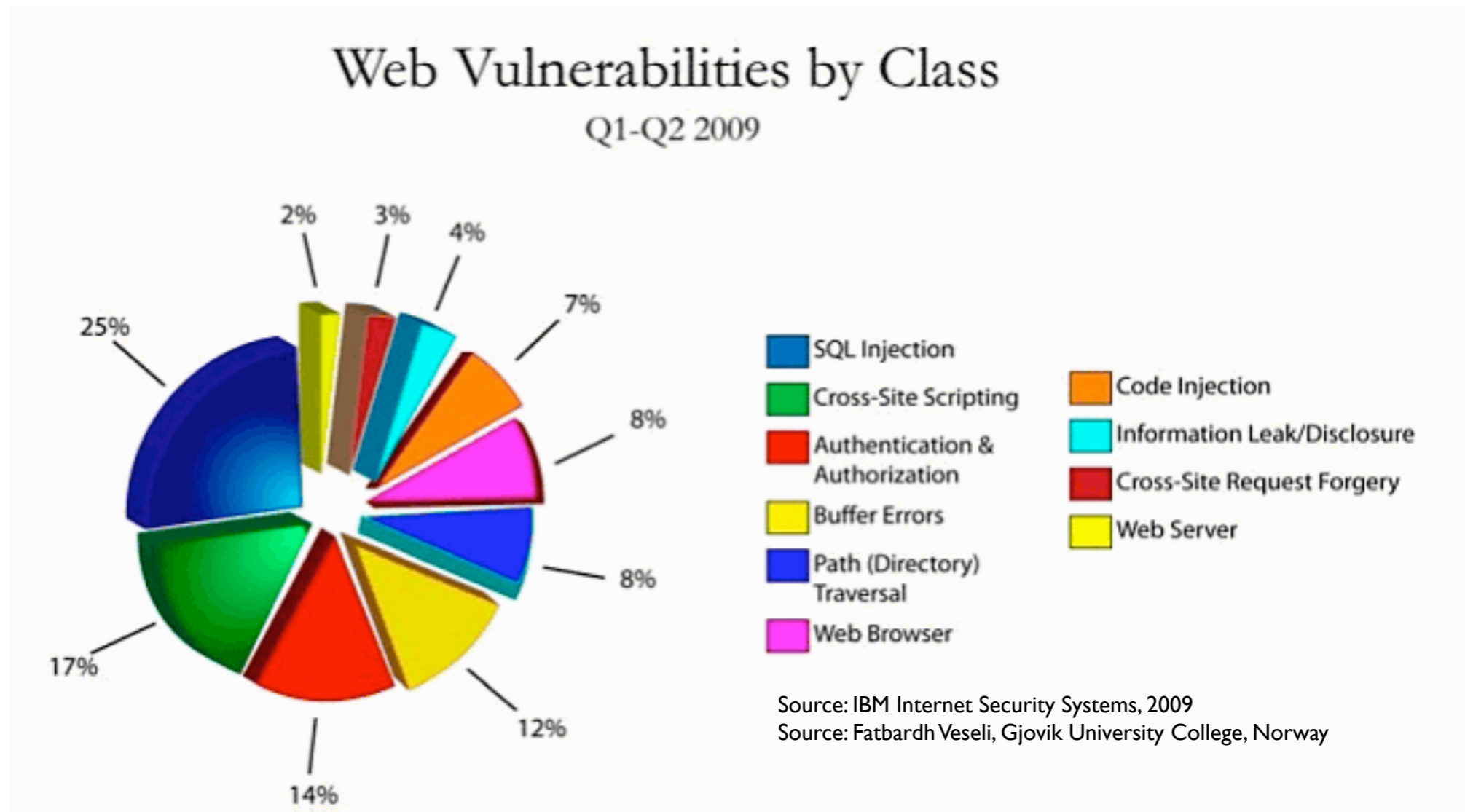
SQL Injection Vulnerabilities



```
SELECT m FROM messages WHERE id='1' OR 1 = 1
```

HAMPI Solver Motivating Example

SQL Injection Vulnerabilities



HAMPI Solver Motivating Example

SQL Injection Vulnerabilities

Buggy Script

```
if (input in regexp("[0-9]+"))  
  query := "SELECT m FROM messages WHERE id=' " + input + "'")
```

- **input** passes validation (regular expression check)
- **query** is syntactically-valid SQL
- **query** can potentially contain an attack substring (e.g., `I' OR 'I' = 'I`)

HAMPI Solver Motivating Example

SQL Injection Vulnerabilities

Should be: “^[0-9]+\$”

Buggy Script

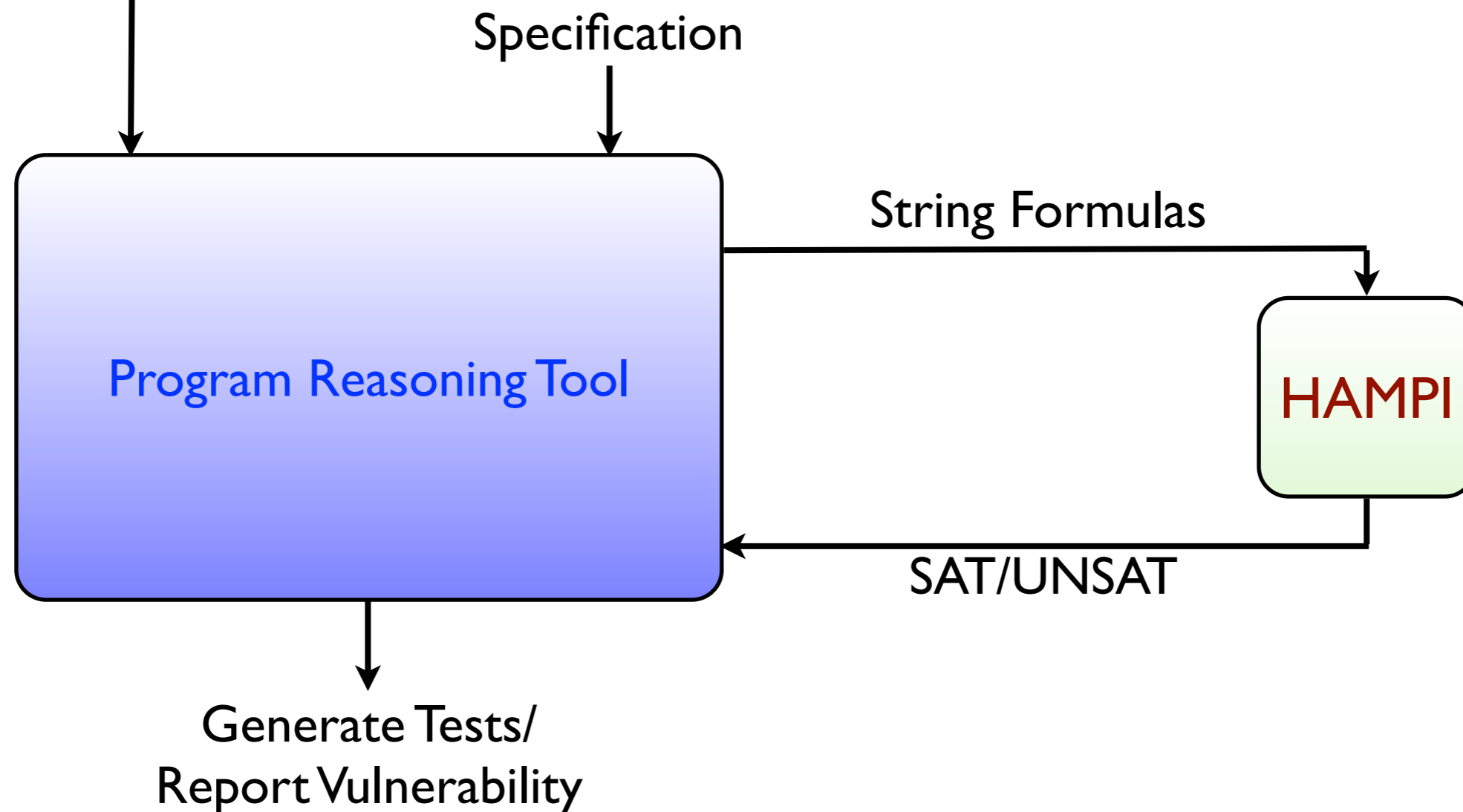
```
if (input in regexp(“[0-9]+”))  
  query := “SELECT m FROM messages WHERE id=’ ” + input + “ ’”
```

- **input** passes validation (regular expression check)
- **query** is syntactically-valid SQL
- **query** can potentially contain an attack substring (e.g., `’ OR ’ = ’`)

HAMPI Solver Motivating Example

SQL Injection Vulnerabilities

```
if (input in regexp("[0-9]+"))  
  query := "SELECT m FROM messages WHERE id=' " + input + "'")
```



Rest of the Talk

- HAMPI Logic: A Theory of Strings
- HAMPI-based Vulnerability Detection App
- How **HAMPI** works
- Experimental Results
- Related Work
- HAMPI 2.0

HAMPI Solver Motivating Example

SQL Injection Vulnerabilities

Input String → `Var v : 12;`

SQL Grammar

`cfg SqlSmall := "SELECT " [a-z]+ " FROM " [a-z]+ " WHERE " Cond;`

`cfg Cond := Val "=" Val | Cond " OR " Cond;`

`cfg Val := [a-z]+ | "" [a-z0-9]* "" | [0-9]+;`

SQL Query

`val q := concat("SELECT msg FROM messages WHERE topicid=", v, "");`

`assert v in [0-9]+;`

“q is a valid SQL query”

`assert q in SqlSmall;`

SQLI attack conditions

`assert q contains "OR '1'='1';`

“q contains an attack vector”

Hampi Key Conceptual Idea

Bounding, expressiveness and efficiency

L_i	Complexity of $\emptyset = L_1 \cap \dots \cap L_n$	Current Solvers
Context-free	Undecidable	n/a
Regular	PSPACE-complete	Quantified Boolean Logic
Bounded	NP-complete	SAT Efficient in practice

Hampi Key Idea: Bounded Logics

Testing, Vulnerability Detection,...

- Finding satisfying assignment is key
- Short assignments are sufficient
- Hence, bounding strings is sufficient
- Furthermore, bounded logics are easier to decide
- HAMPI bounds sets defined by Context-free Grammars (CFG) and Regular Expressions (RE)

Hampi Key Idea: Bounded Logics

Bounding vs. Completeness

- Bounding leads to incompleteness
- Testing (Bounded MC) vs. Verification (MC)
- Bounding allows trade-off (Scalability vs. Completeness)
- Completeness (also, soundness) as resources

HAMPI Solver Motivating Example

SQL Injection Vulnerabilities

Input String → Var v : 12;

SQL Grammar

cfg *SqISmall* := "SELECT " [a-z]+ " FROM " [a-z]+ " WHERE " *Cond*;

cfg *Cond* := *Val* "=" *Val* | *Cond* " OR " *Cond*;

cfg *Val* := [a-z]+ | "'" [a-z0-9]* "'" | [0-9]+;

SQL Query

val q := concat("SELECT msg FROM messages WHERE topicid=", v, "");

assert v in [0-9]+;

“q is a valid SQL query”

assert q in *SqISmall*;

SQLI attack conditions

assert q contains "OR '1'='1'";

“q contains an attack vector”

How Hampi Works

Bird's Eye View: Strings into Bit-vectors

```
var v : 4;
```

```
cfg E := “()” | E E | “(“ E “)”;
```

```
val q := concat(“(“ , v , “)”);
```

```
assert q in E;
```

```
assert q contains “()”;
```

Hampi

Normalizer

STP Encoder

STP Decoder

Bit-vector
Constraints

STP

Bit-vector
Solution

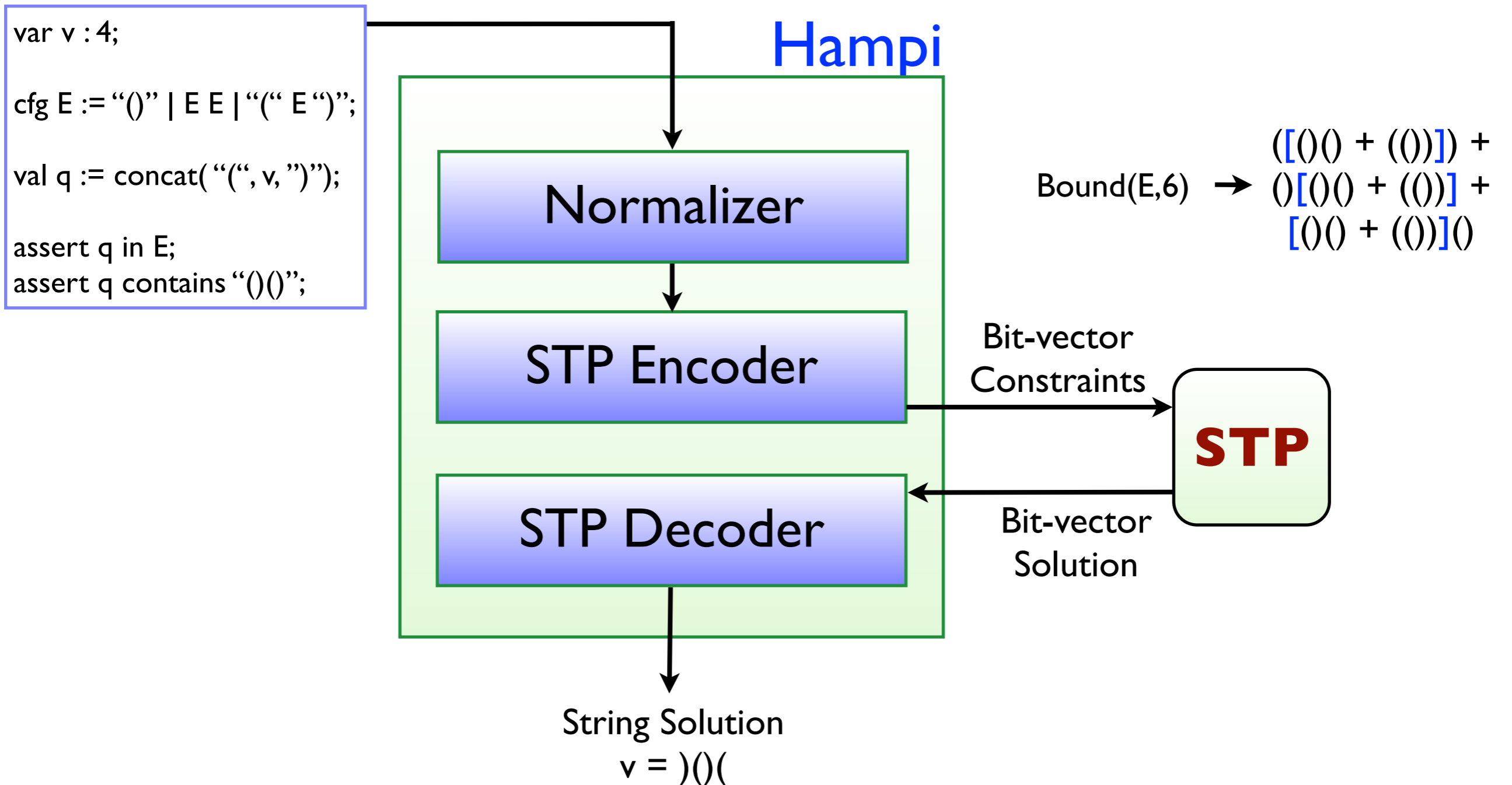
String Solution
v =)()(

Find a 4-char string v:

- (v) is in E
- (v) contains ()()

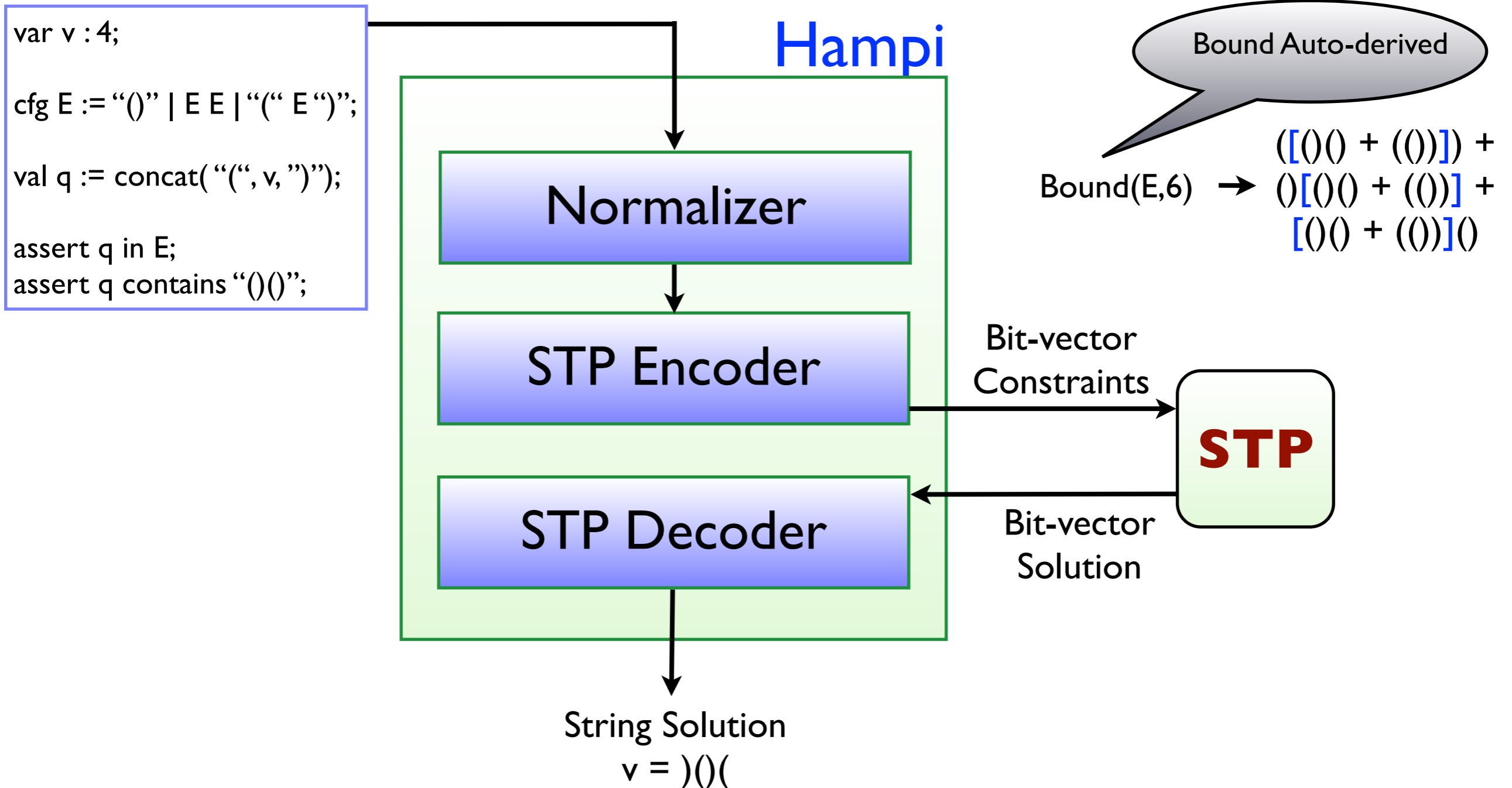
How Hampi Works

Unroll Bounded CFGs into Regular Exp.

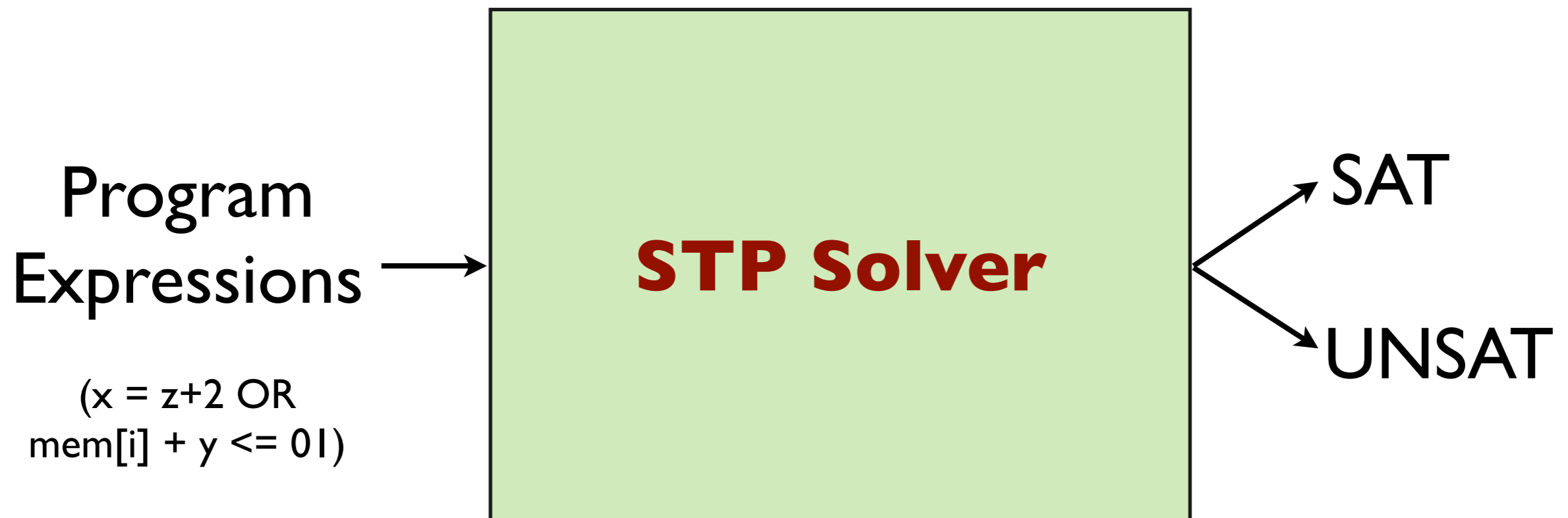


How Hampi Works

Unroll Bounded CFGs into Regular Exp.



STP Bit-vector & Array Solver



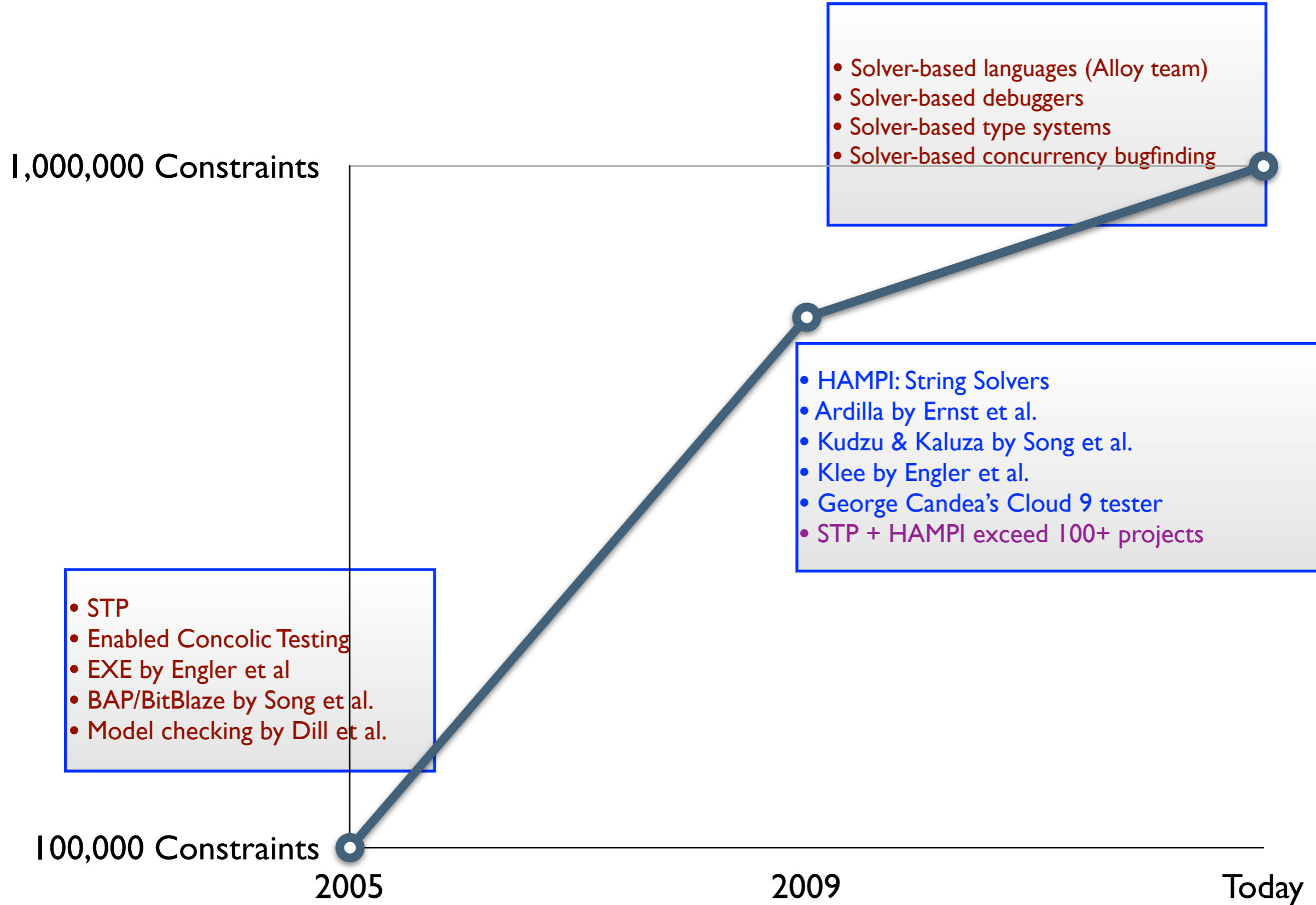
- Bit-vector or machine arithmetic
- Arrays for memory
- C/C++/Java expressions
- NP-complete

Impact of STP: Notable Projects

- Enabled Concolic Testing
- 100+ reliability and security projects

<u>Category</u>	<u>Research Project</u>	<u>Project Leader/Institution</u>
Formal Methods	ACL2 Theorem Prover + STP Verification-aware Design Checker Java PathFinder Model Checker	Eric Smith & David Dill/ Stanford Jacob Chang & David Dill/ Stanford Mehlitz & Pasareanu/ NASA
Program Analysis	BitBlaze & WebBlaze BAP	Dawn Song et al./ Berkeley David Brumley/ CMU
Automatic Testing Security	Klee, EXE SmartFuzz Kudzu S2E & Cloud9	Engler & Cadar/ Stanford Molnar & Wagner/ Berkeley Saxena & Song/ Berkeley Bucur & Candea/ EPFL
Hardware Bounded Model-checking (BMC)	Blue-spec BMC BMC	Katelman & Dave/ MIT Haimed/ NVIDIA

The History of STP



Key Contributions

<u>Name</u>	<u>Key Concept</u>	<u>Impact</u>	<u>Pubs</u>
STP Bit-vector & Array Solver ^{1,2}	Abstraction-refinement for Solving	Concolic Testing	CAV 2007 CCS 2006 TISSEC 2008
HAMPI String Solver ¹	App-driven Bounding for Solving	Analysis of Web Apps	ISSTA 2009 ³ TOSEM 2011 (Invited/in submission)
(Un)Decidability results for Strings	Insights from Practical Applications	First results for strings+length	In submission

1. 100+ research projects use STP and HAMPI
2. STP won the SMTCOMP 2006 and 2010 competitions for bit-vector solvers
3. ACM Best Paper Award 2009

How Hampi Works

Bird's Eye View: Strings into Bit-vectors

```
var v : 4;
```

```
cfg E := “()” | E E | “(“ E “)”;
```

```
val q := concat(“(“ , v , “)”);
```

```
assert q in E;
```

```
assert q contains “()”;
```

Hampi

Normalizer

STP Encoder

STP Decoder

Bit-vector
Constraints

STP

Bit-vector
Solution

String Solution
v =)()(

Find a 4-char string v:

- (v) is in E
- (v) contains ()()

How Hampi Works

Unroll Bounded CFGs into Regular Exp.

Step 1:

```
var v : 4;  
cfg E := “()” | E E | “(“ E “)”;  
val q := concat(“(“ , v , “)”);  
assert q in E;  
assert q contains “()()”;
```

Auto-derive
lower/upper bounds
[L,B]
on CFG

[6,6]

Step 2:

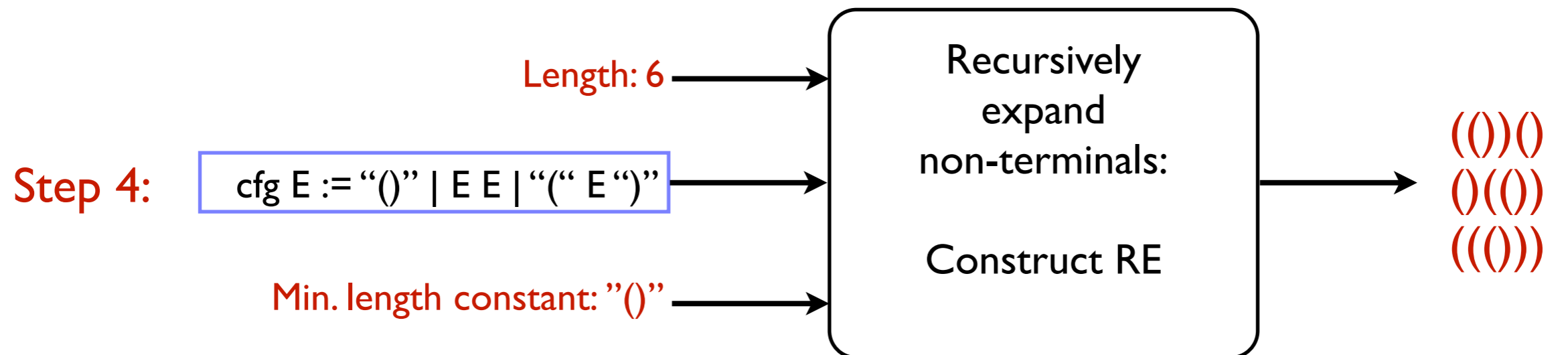
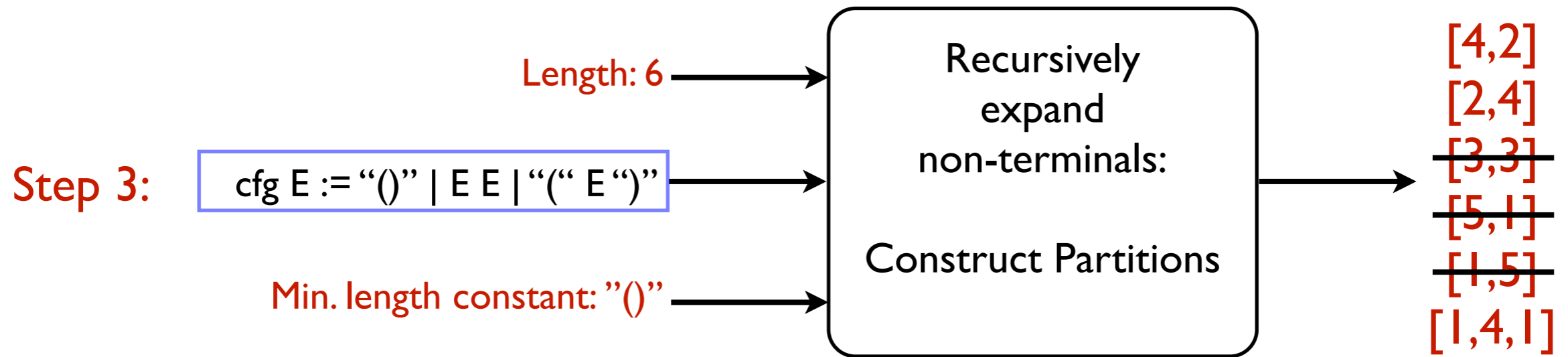
```
cfg E := “()” | E E | “(“ E “)”
```

Look for
minimal length
string

“()”

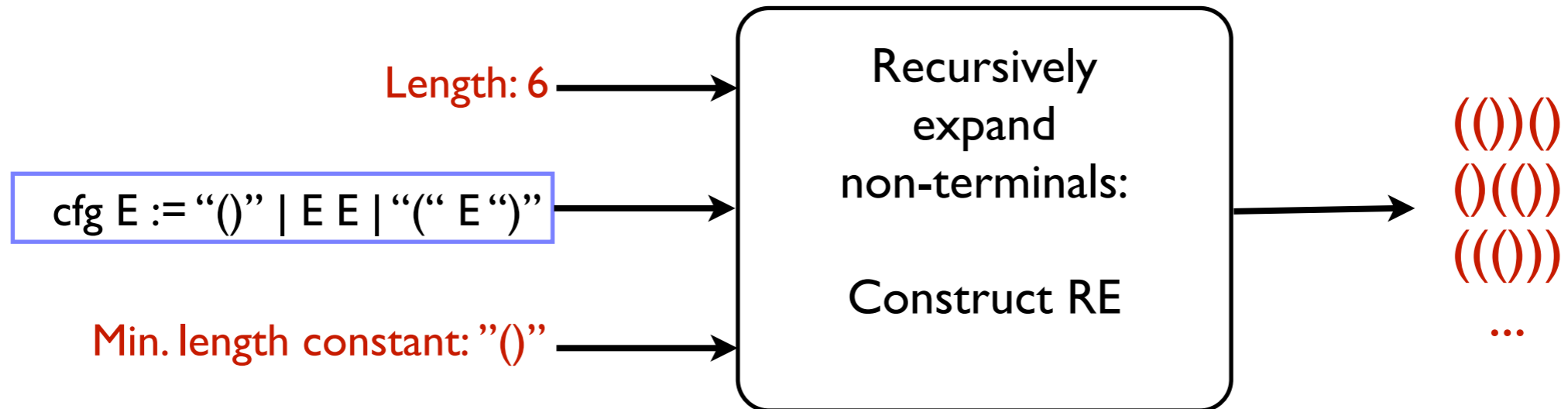
How Hampi Works

Unroll Bounded CFGs into Regular Exp.



Unroll Bounded CFGs into Regular Exp.

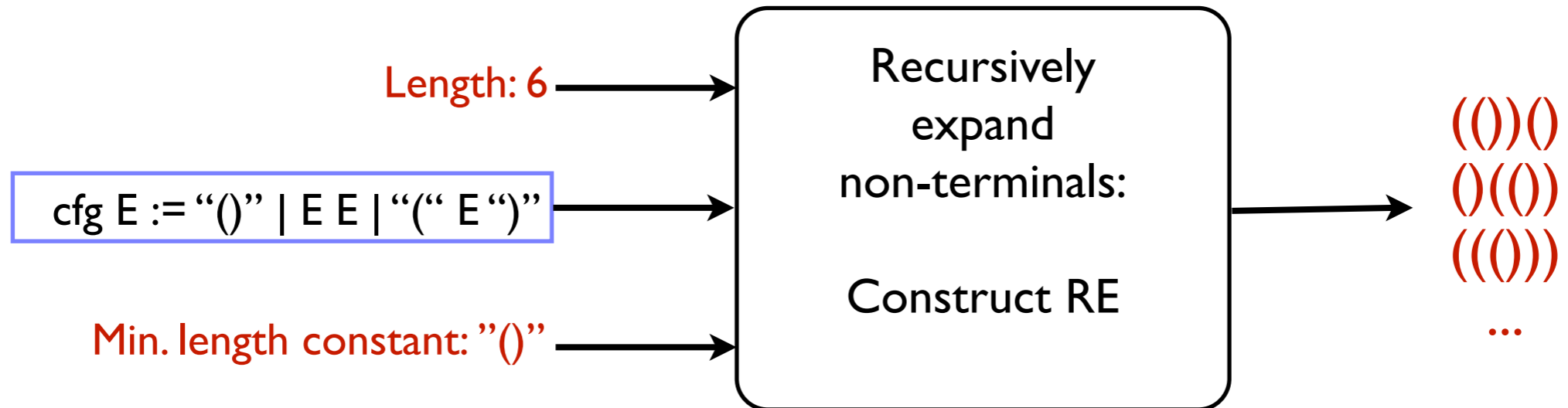
Managing Exponential Blow-up



- Dynamic programming style
- Works well in practice

Unroll Bounded CFGs into Regular Exp.

Managing Exponential Blow-up



$$\text{Bound}(E,6) \rightarrow \begin{aligned} &([() + ()]) + \\ &()[() + ()] + \\ &[() + ()]() \end{aligned}$$

How Hampi Works

Converting Regular Exp. into Bit-vectors

Encode regular expressions recursively

- Alphabet { (,) } \rightarrow 0, 1
- constant \rightarrow bit-vector constant
- union + \rightarrow disjunction \vee
- concatenation \rightarrow conjunction \wedge
- Kleene star * \rightarrow conjunction \wedge
- Membership, equality \rightarrow equality

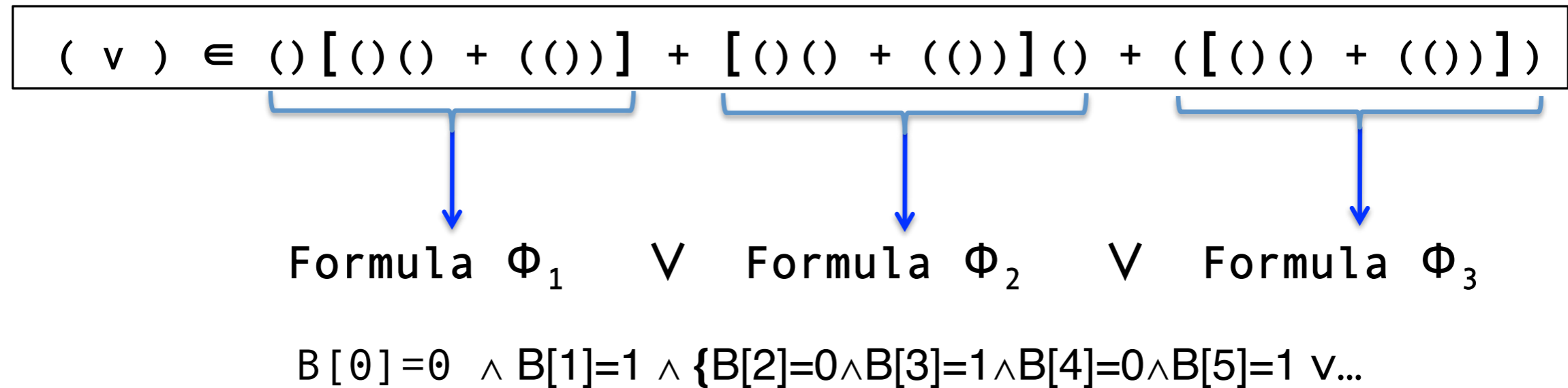
(v) \in () [() () + (())] + [() () + (())] () + ([() () + (())])

Formula Φ_1 \vee Formula Φ_2 \vee Formula Φ_3

$B[0]=0 \wedge B[1]=1 \wedge \{ B[2]=0 \wedge B[3]=1 \wedge B[4]=0 \wedge B[5]=1 \vee \dots$

How Hampi Works

Converting Regular Exp. into Bit-vectors



- Constraint Templates
- Encode once, and reuse
- On-demand formula generation

How Hampi Works

Decoder converts Bit-vectors to Strings

```
var v : 4;
```

```
cfg E := “()” | E E | “(“ E “)”;
```

```
val q := concat(“(“ , v , “)”);
```

```
assert q in E;
```

```
assert q contains “()”;
```

Hampi

Normalizer

STP Encoder

STP Decoder

Bit-vector
Constraints

STP

Bit-vector
Solution

String Solution
v =)()(

Find a 4-char string v:

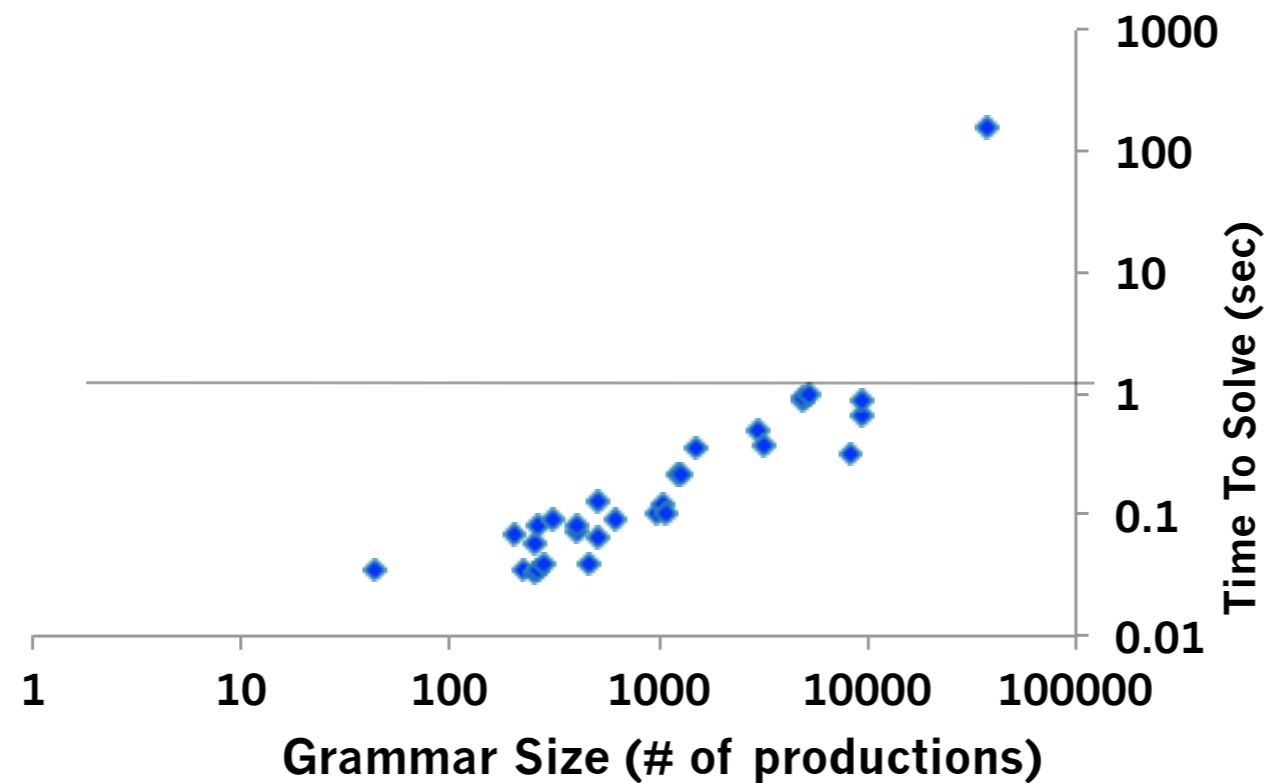
- (v) is in E
- (v) contains ()()

Rest of the Talk

- HAMPI Logic: A Theory of Strings
- HAMPI-based Vulnerability Detection App
- How HAMPI works
- Experimental Results
- Related Work
- Future Work

HAMPI: Result I

Static SQL Injection Analysis



- 1367 string constraints from Wasserman & Su [PLDI'07]
- Hampi scales to **large grammars**
- Hampi solved 99.7% of constraints in < 1sec
- All solvable constraints had short solutions

HAMPI: Result 2

Security Testing

- Hampi used to build Ardilla security tester [Kiezun et al., ICSE'09]
 - 60 new vulnerabilities on 5 PHP applications (300+ kLOC)
 - 23 SQL injection
 - 37 cross-site scripting (XSS)
- 5 added to
US National Vulnerability DB
- 46% of constraints solved in < 1 second per constraint
 - 100% of constraints solved in < 10 seconds per constraint

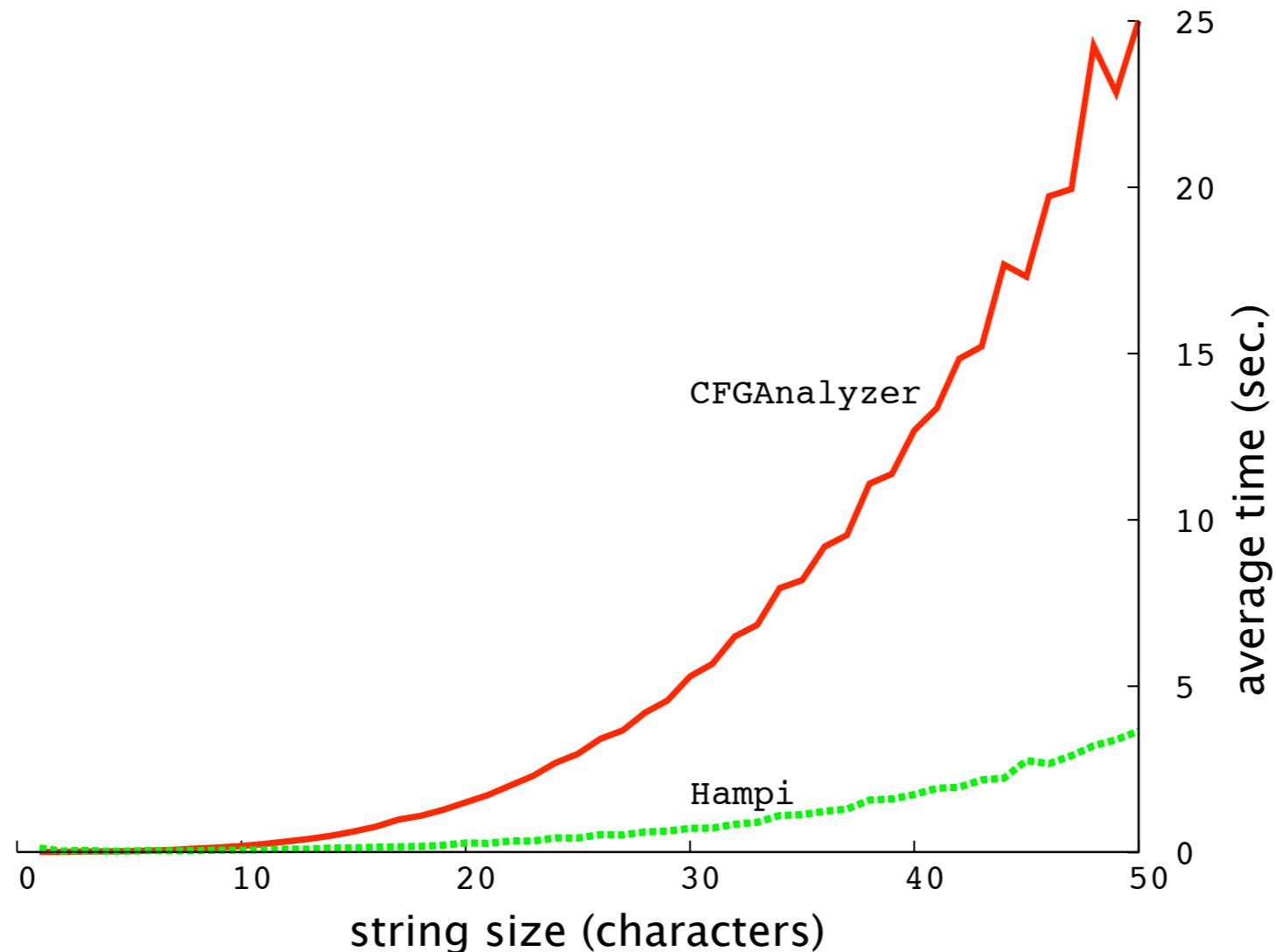
HAMPI: Result 2

Security Testing and XSS

- Attackers inject client-side script into web pages
- Somehow circumvent same-origin policy in websites
- echo “Thank you \$my_poster for using the message board”;
- Unsanitized \$my_poster
- Can be JavaScript
- Execution can be bad

HAMPI: Result 3

Comparison with Competing Tools



- [HAMPI vs. CFGAnalyzer \(U. Munich\)](#): HAMPI ~7x faster for strings of size 50+

HAMPI: Result 3

Comparison with Competing Tools

RE intersection problems

- HAMPI 100x faster than Rex (MSR)
- HAMPI 1000x faster than DPRLE (U.Virginia)
- Pieter Hooimeijer 2010 paper titled 'Solving String Constraints Lazily'

HAMPI: Result 4

Helping KLEE Pierce Parsers

HAMPI for Klee

- Klee provides API to place constraints on symbolic inputs
- Particularly useful for programs with highly-structured inputs
- Manually writing constraints is hard
- Specify grammar using HAMPI, compile to C code
- 2-5X improvement in line coverage

Impact of Hampi: Notable Projects

<u>Category</u>	<u>Research Project</u>	<u>Project Leader/Institution</u>
Static Analysis	SQL-injection vulnerabilities	Wasserman & Su/UC, Davis
Security Testing	Ardilla for PHP (SQL injections, cross-site scripting)	Kiezun & Ernst/MIT
Concolic Testing	Klee SAGE Kudzu NoTamper	Engler & Cadar/Stanford Godefroid/Microsoft Research Saxena & Song/Berkeley Bisht & Venkatakrishnan/U Chicago
New Solvers	Kaluza	Saxena & Song/Berkeley

Impact of Hampi: Notable Projects

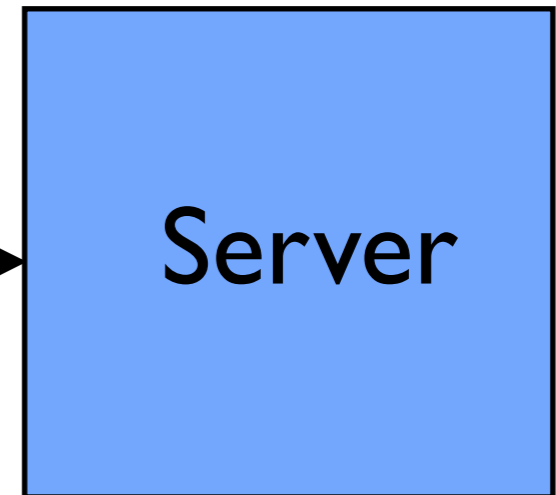
<u>Tool Name</u>	<u>Description</u>	<u>Project Leader/ Institution</u>
Kudzu	JavaScript Bug Finder & Vulnerability Detector	Saxena Akhawe Hanna Mao McCamant Song/Berkeley
NoTamper	Parameter Tamper Detection	Bisht Hinrichs/U of Chicago Skrupsky Bobrowicz Vekatakrisnan/ U. of Illinois, Chicago

Impact of Hampi: Notable Projects

NoTamper



sca0022 www.fotosearch.com



- Client-side checks (C), no server checks
- Find solutions S_1, S_2, \dots to C, and solutions E_1, E_2, \dots to $\sim C$ by calling HAMPI
- E_1, E_2, \dots are candidate exploits
- Submit $(S_i, E_i), \dots$ to server
- If server response same, ignore
- If server response differ, report error

Related Work

<u>Tool Name</u>	<u>Project Leader/ Institution</u>	<u>Comparison with HAMPI</u>
Rex	Bjorner, Tillman, Veanes et al. (Microsoft Research, Redmond)	<ul style="list-style-type: none">• HAMPI + Length+Replace(s_1, s_2, s_3) - CFG• Translation to int. linear arith. (Z3)
Mona	Karllund et al. (U. of Aarhus)	<ul style="list-style-type: none">• Can encode HAMPI & Rex• User work• Automata-based• Non-elementary
DPRLE	Hooimeijer (U. of Virginia)	<ul style="list-style-type: none">• Regular expression constraints

Topics Covered Today

- **HAMPI Logic**: A Theory of Strings
- **HAMPI**-based Testing App
- How **HAMPI** works
- Another **HAMPI**-based App: Tamper Detection
- Experimental Results
- Related Work (Kaluza, Rex,...)

Conclusions

- String solvers essential for many apps
- HAMPI supports
string vars, constants, concat/extract, equality, membership, contains predicate
- Demand for even richer theories
 - Attribute grammars
 - String theories with length
- Bounding: Powerful and versatile idea (BMC, bounded logics,...)
- Using completeness as a resource

HAMPI 2.0

- HAMPI logic + length function + replace function
- Small model property (under certain conditions)
- Combination with other theories such as functions