# HAMPI

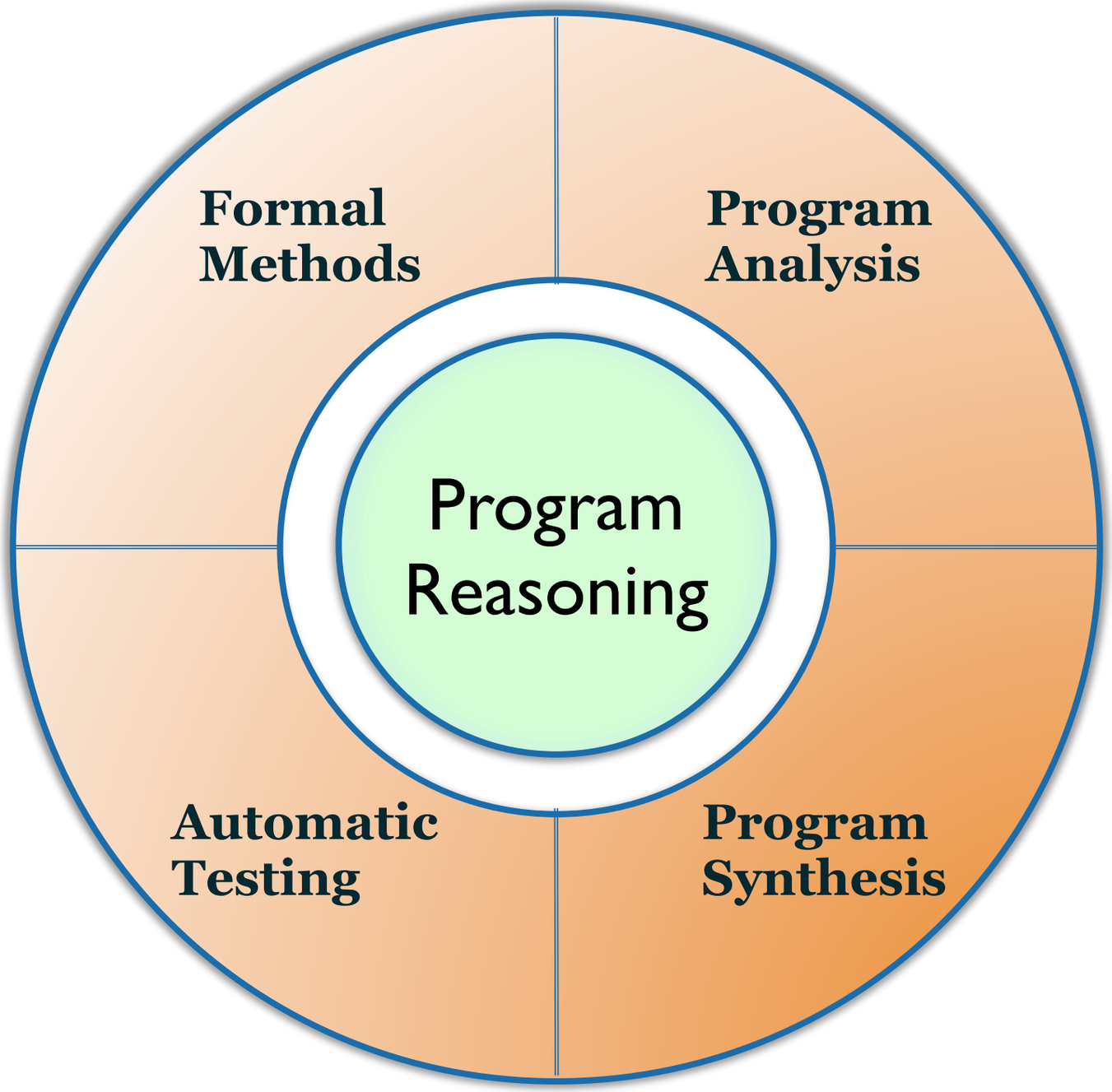
## A String Solver for Testing, Analysis and Vulnerability Detection

Vijay Ganesh
MIT

July 16, 2011

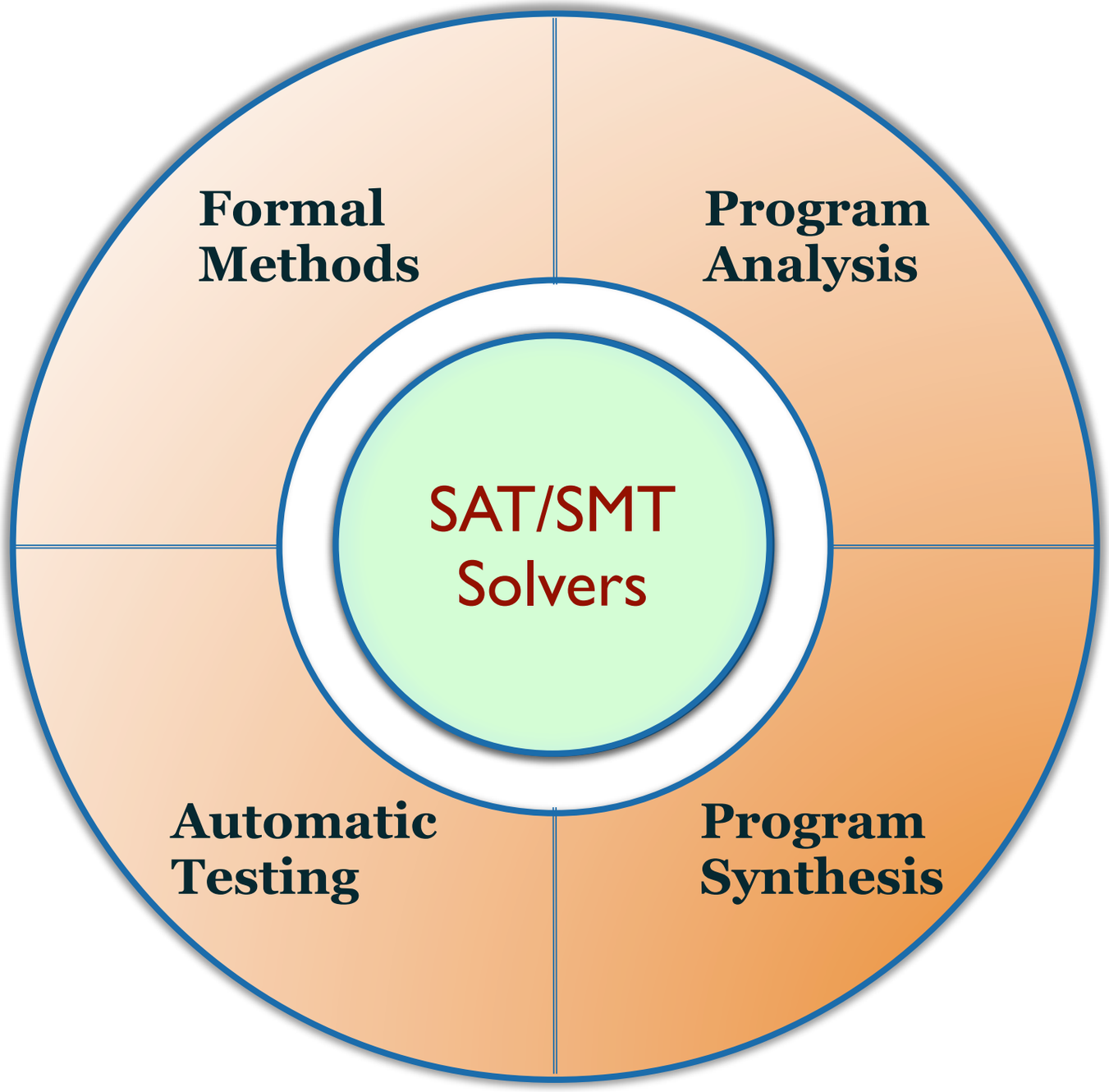
# Software Engineering & SMT Solvers

## An Indispensable Tactic



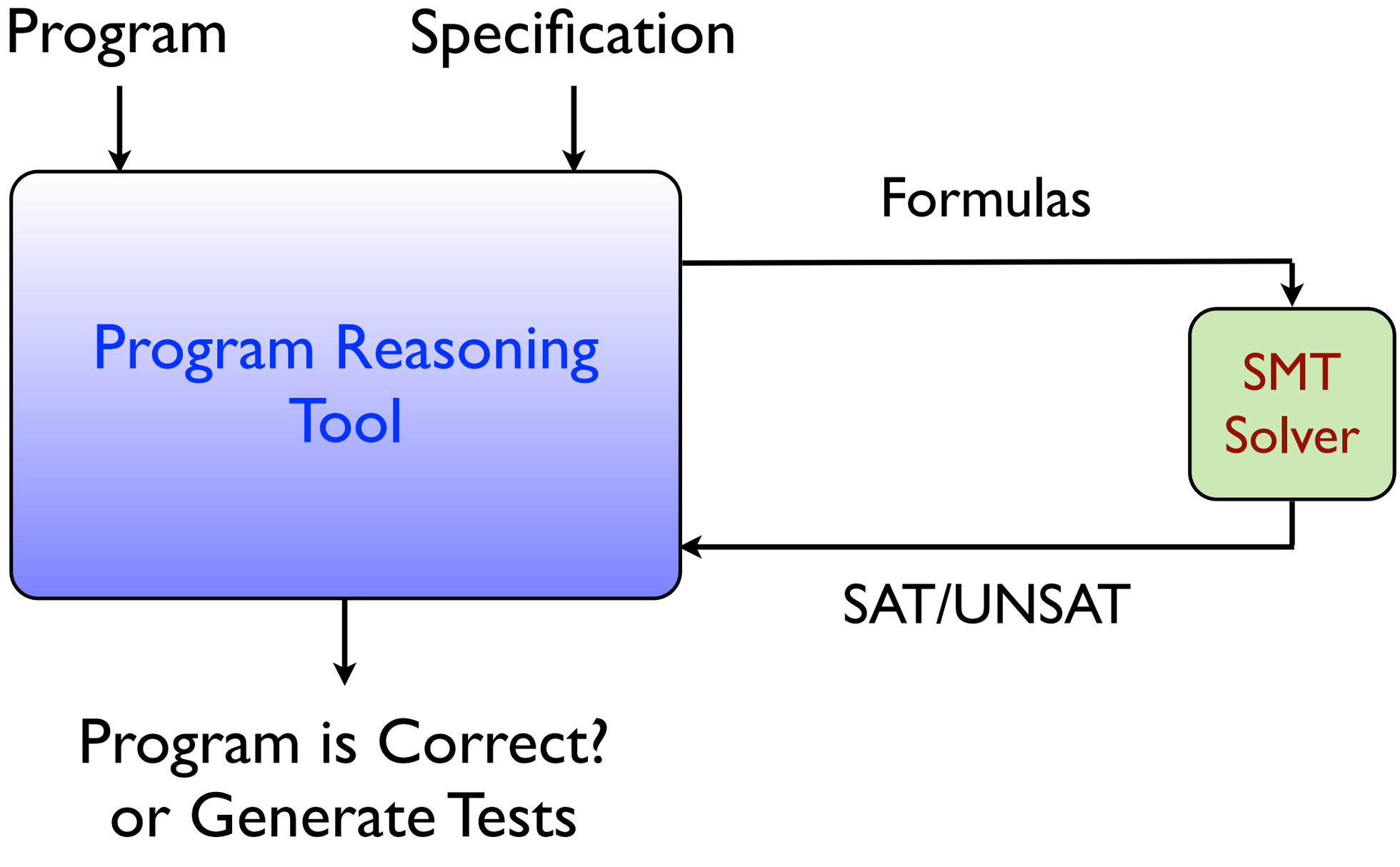
# Software Engineering & SMT Solvers

## An Indispensable Tactic



# Traditional SMT Problem Statement

## Efficient Solver for Analysis of Programs



# Traditional SMT Logics

## Efficient Solvers for Program Expressions

- Integer/Real Linear Arithmetic
- Bit-vectors
- Arrays
- Uninterpreted Functions
- Abstract Datatypes
- Quantifiers
- Non-linear Arithmetic
- Strings?

# Key SMT Concepts

# Logician's Question: What's New?

- Approximations
- Asymptotically speaking: probably the same
- SAT
  - Clause learning using conflict analysis
  - Backjumping
  - Variable selection heuristics
  - Restarts
- SMT
  - Combinations
  - Under/Over approximations of formulas
  - DPLL(T)
  - Bounding

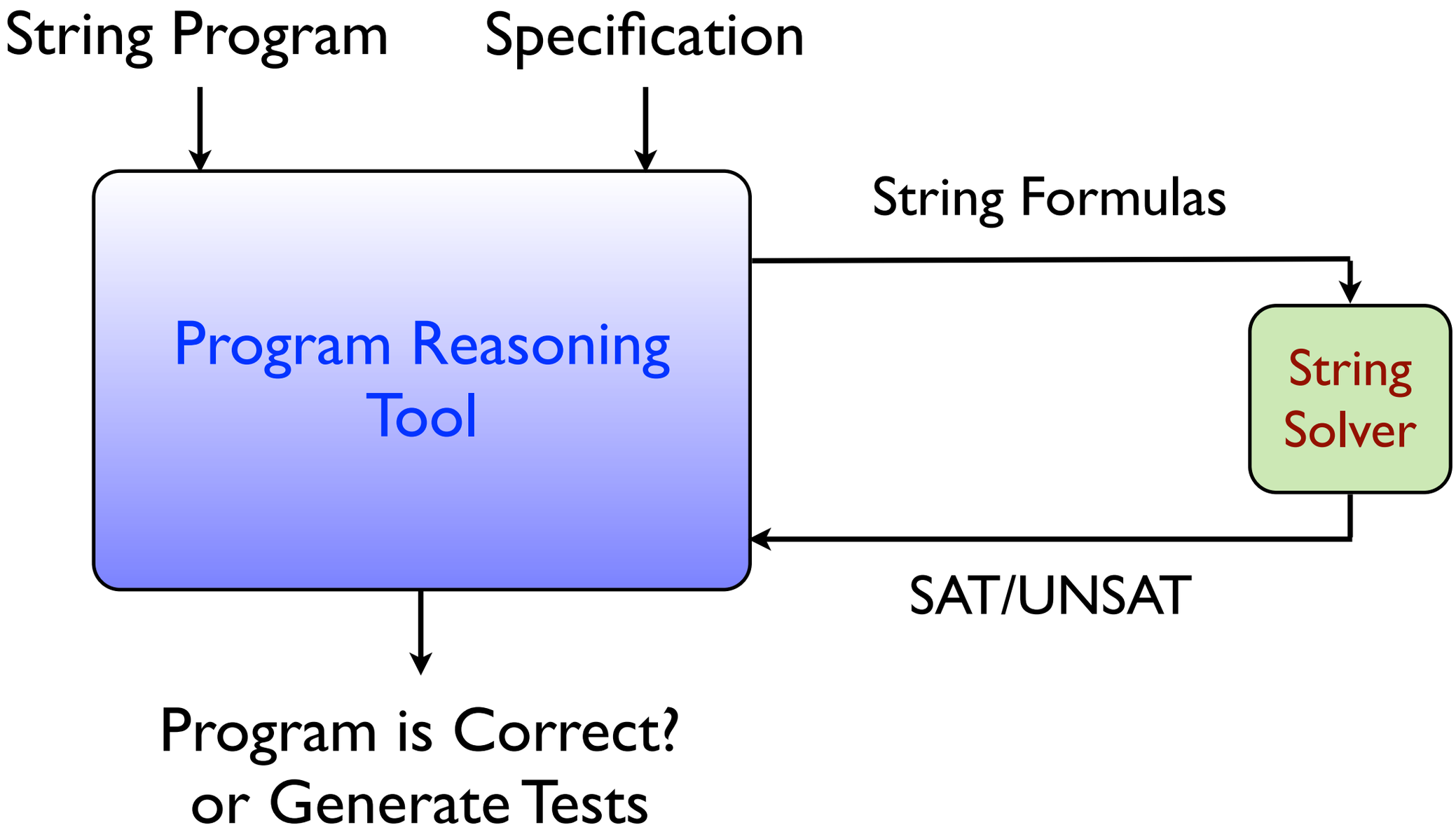
# Why a String Solver?

## Efficient Solver for Analysis of String Programs

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

# String Solver Problem Statement

## Efficient Solver for Analysis of String Programs



# HAMPI String Solver


- X = concat("SELECT...",v) AND (X ∈ SQL_grammar)
- JavaScript, PHP, ... string expressions
- NP-complete
- ACM Distinguished Paper Award 2009

# Take Home Message

- Theories of Strings are increasingly key for reliability/security

- Conceptual idea: Bounded logics

- Use HAMPI

# Rest of the Talk

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

# Theory of Strings

# The Hampi Language

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

# Theory of Strings

## The Hampi Language

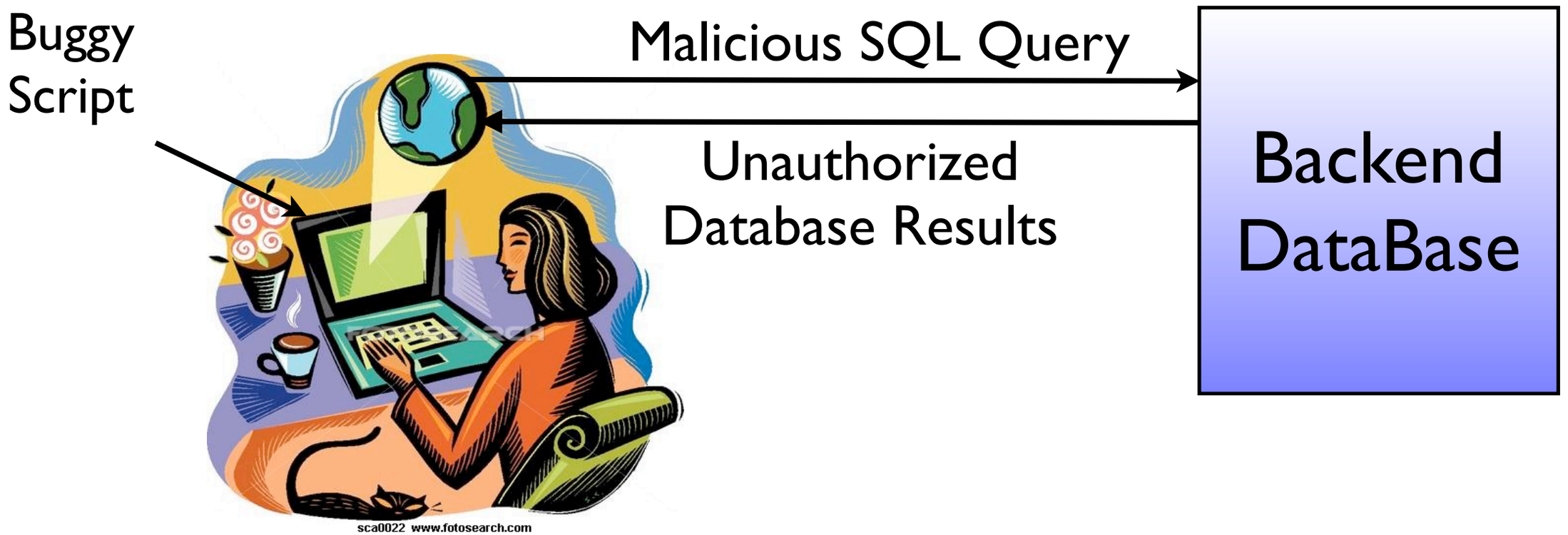
- X = concat("SELECT msg FROM msgs WHERE topicid = ",v)
  AND
  (X ∈ SQL_Grammar)

- input ∈ RegExp([0-9]+)

- X = concat (str_term1, str_term2, "c")[1:42]
  AND
  X contains "abc"

# HAMPI Solver Motivating Example

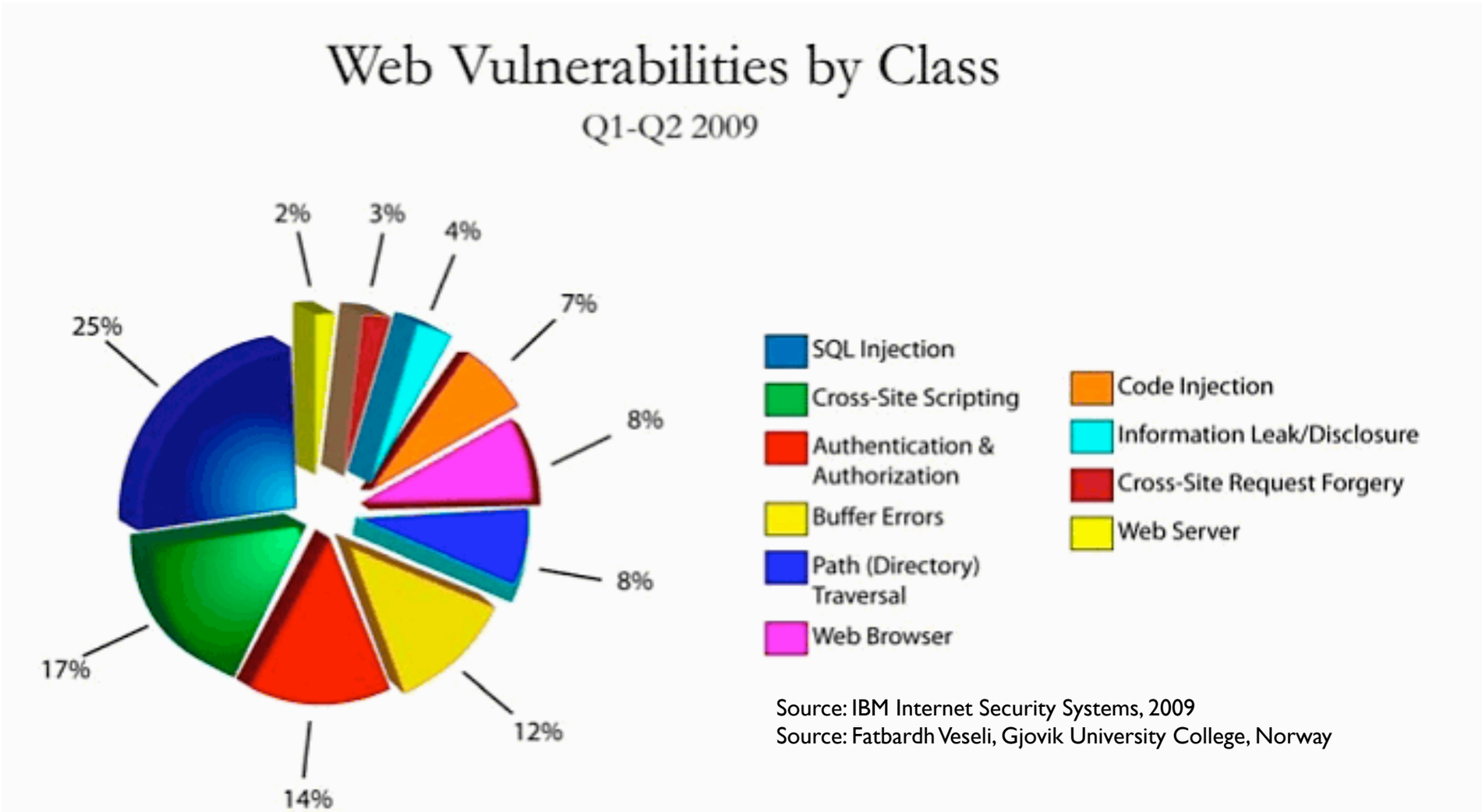
## SQL Injection Vulnerabilities




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

# HAMPI Solver Motivating Example

## SQL Injection Vulnerabilities




Source: IBM Internet Security Systems, 2009
Source: Fatbardh Veseli, Gjovik University College, Norway

# 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., 1’ OR ‘1’ = ‘1)

# HAMPI Solver Motivating Example

## SQL Injection Vulnerabilities

Should be: "^[0-9]+$"

Buggy Script

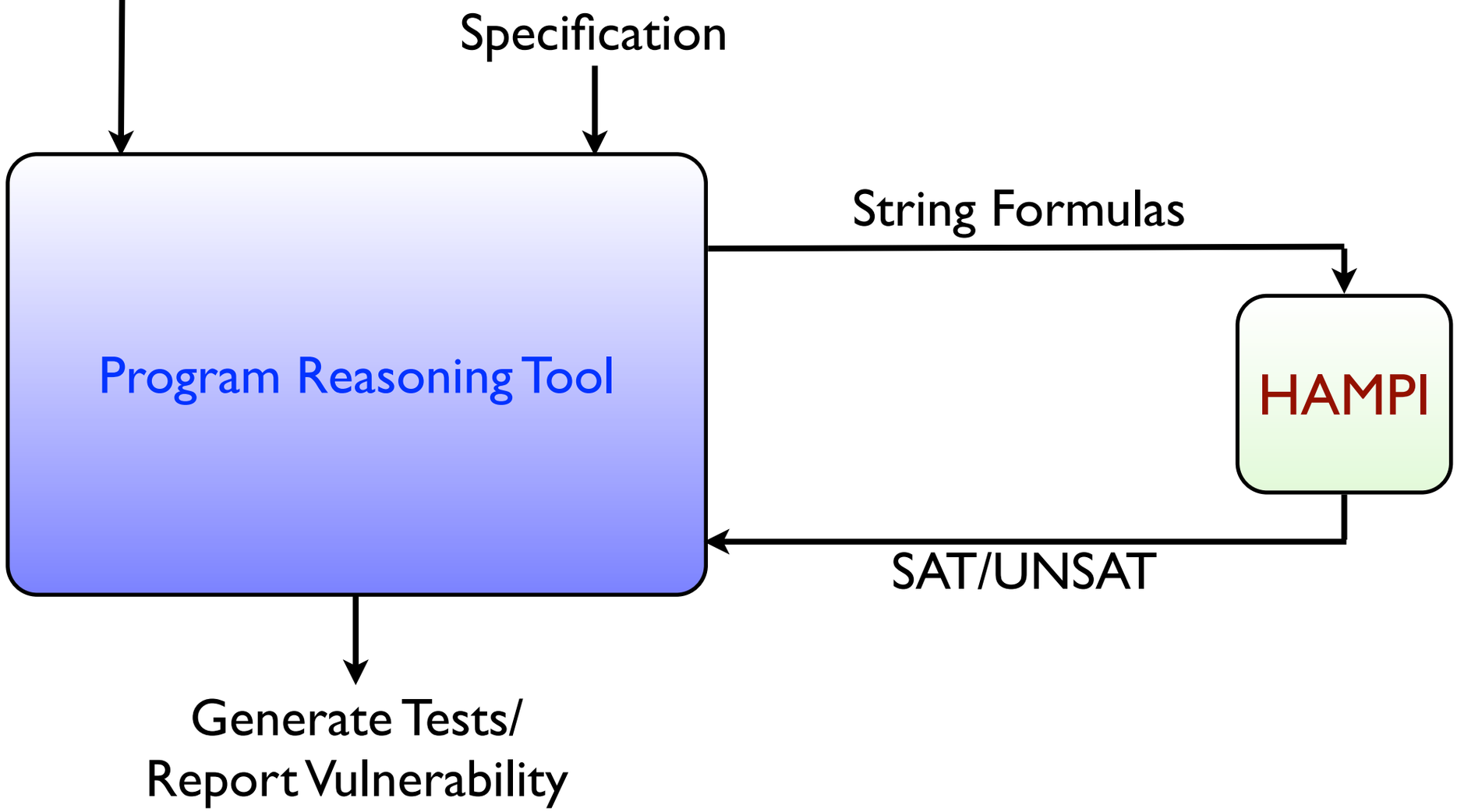
```
if (input in regexp("[0-9]+"))
   query := "SELECT m FROM messages WHERE id=' " + input +  " ' ")
```

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

# 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
- Motivating Example: HAMPI-based Vulnerability Detection App
- How HAMPI works
- Experimental Results
- Related Work: Theory and Practice
- HAMPI 2.0
- SMTization: Future of Strings

# Expressing the Problem in HAMPI

## 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”**

**SQLI attack conditions**

assert q in *SqlSmall*;

assert q contains "OR ‘1'=‘1'";

**“q contains an attack vector”**

# Hampi Key Conceptual Idea

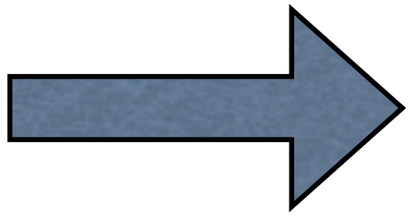
## Bounding, expressiveness and efficiency

| $L_i$ | Complexity of $\varnothing = L_1 \cap \ldots \cap L_n$ | Current Solvers |
|---|---|---|
| Context-free | Undecidable | n/a |
| Regular | PSPACE-complete | Quantified Boolean Logic |
| Bounded | NP-complete | SAT<br>Efficient in practice |

# Hampi Key Idea: Bounded Logics

## Testing, Vulnerability Detection,...

- Finding SAT assignment is key
- Short assignments are sufficient



- Bounding is sufficient
- Bounded logics easier to decide

# 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 *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”**

**SQLI attack conditions**

assert q in *SqlSmall*;

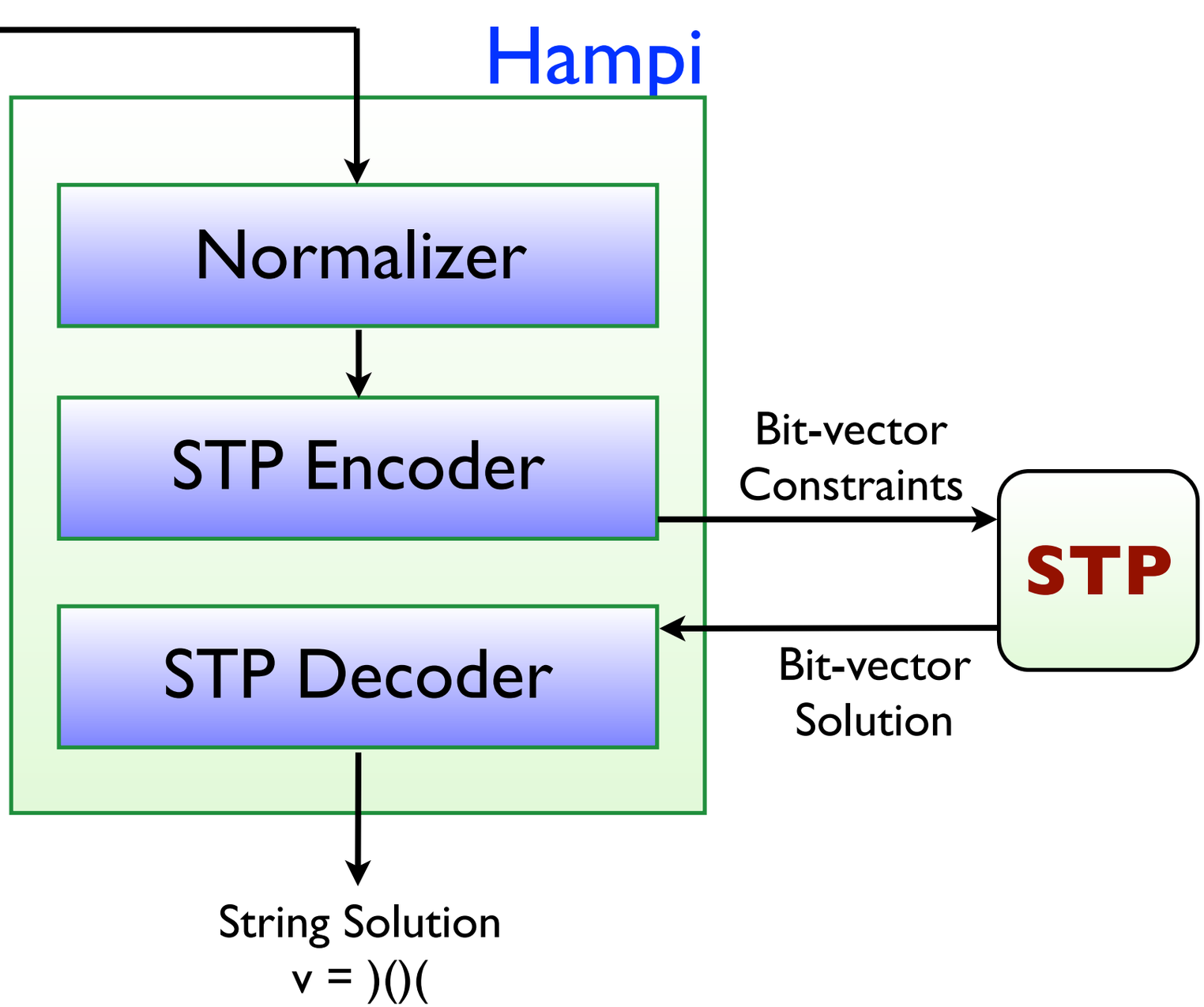
assert q contains "OR ‘1'=‘1'";

**“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 "()()";
```



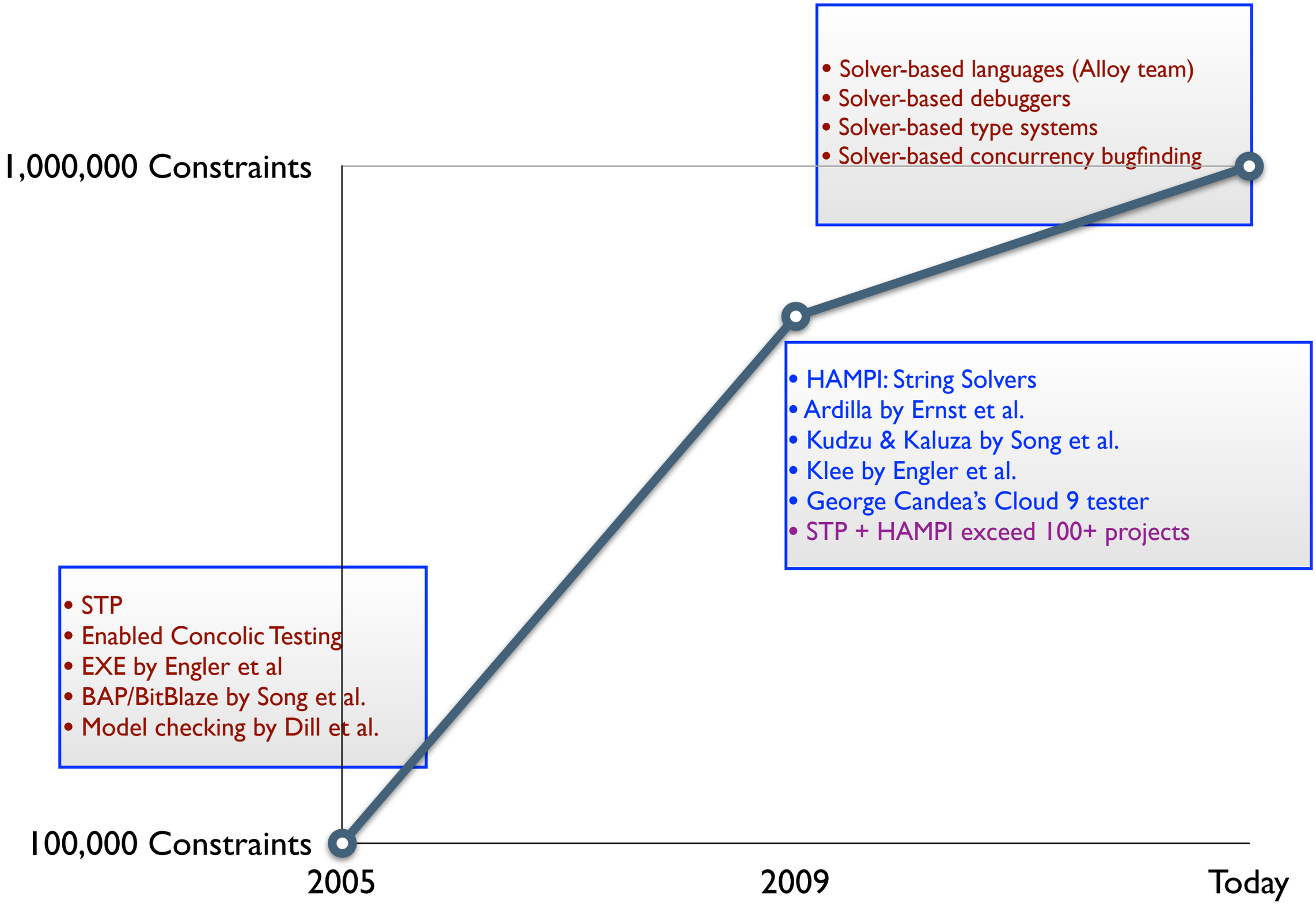

Find a 4-char string v:

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

# STP Bit-vector & Array Solver


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

# The History of STP



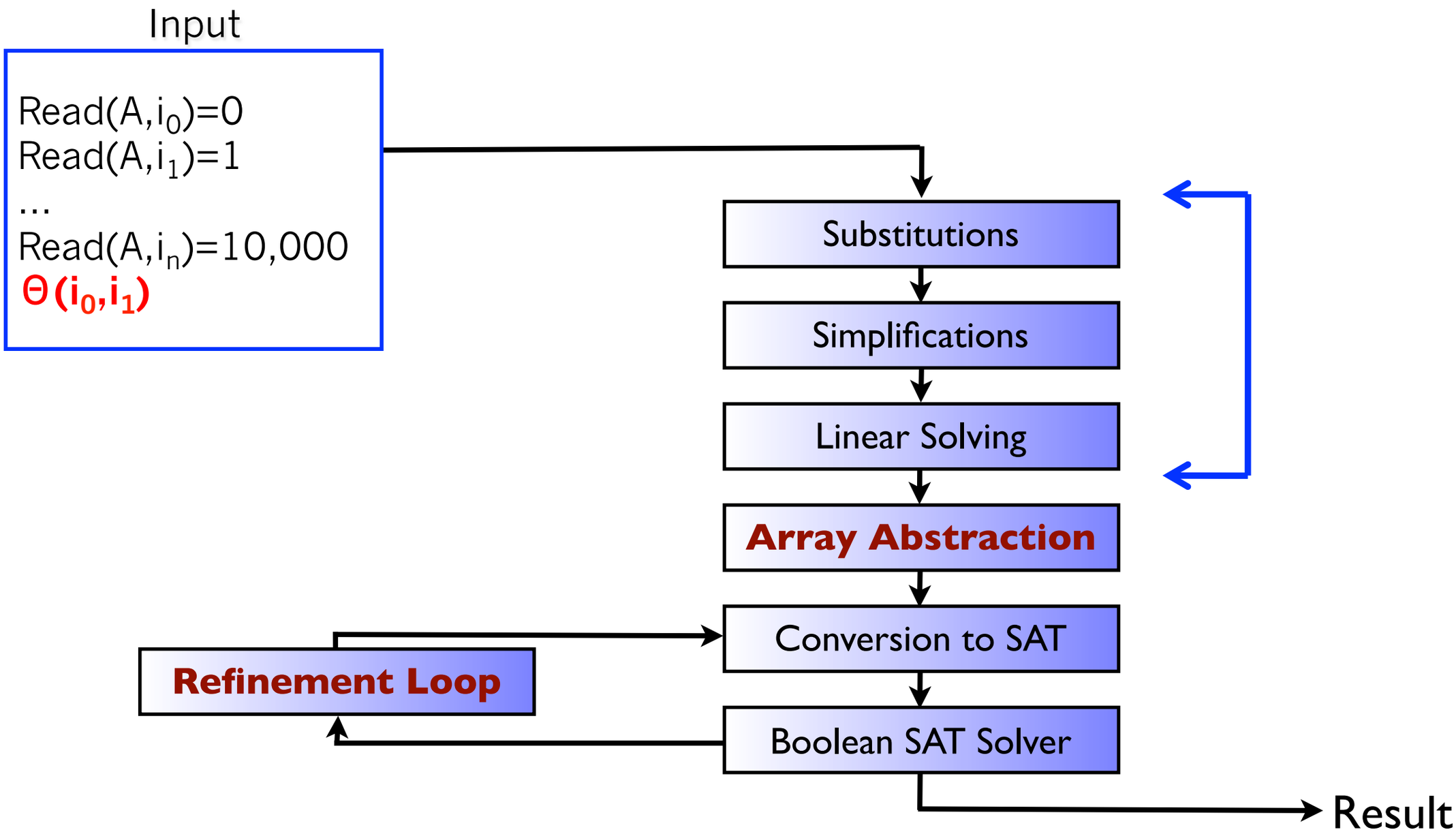
# Impact of STP: Notable Projects

- Enabled Concolic Testing
- 100+ reliability and security projects

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

# How STP Works

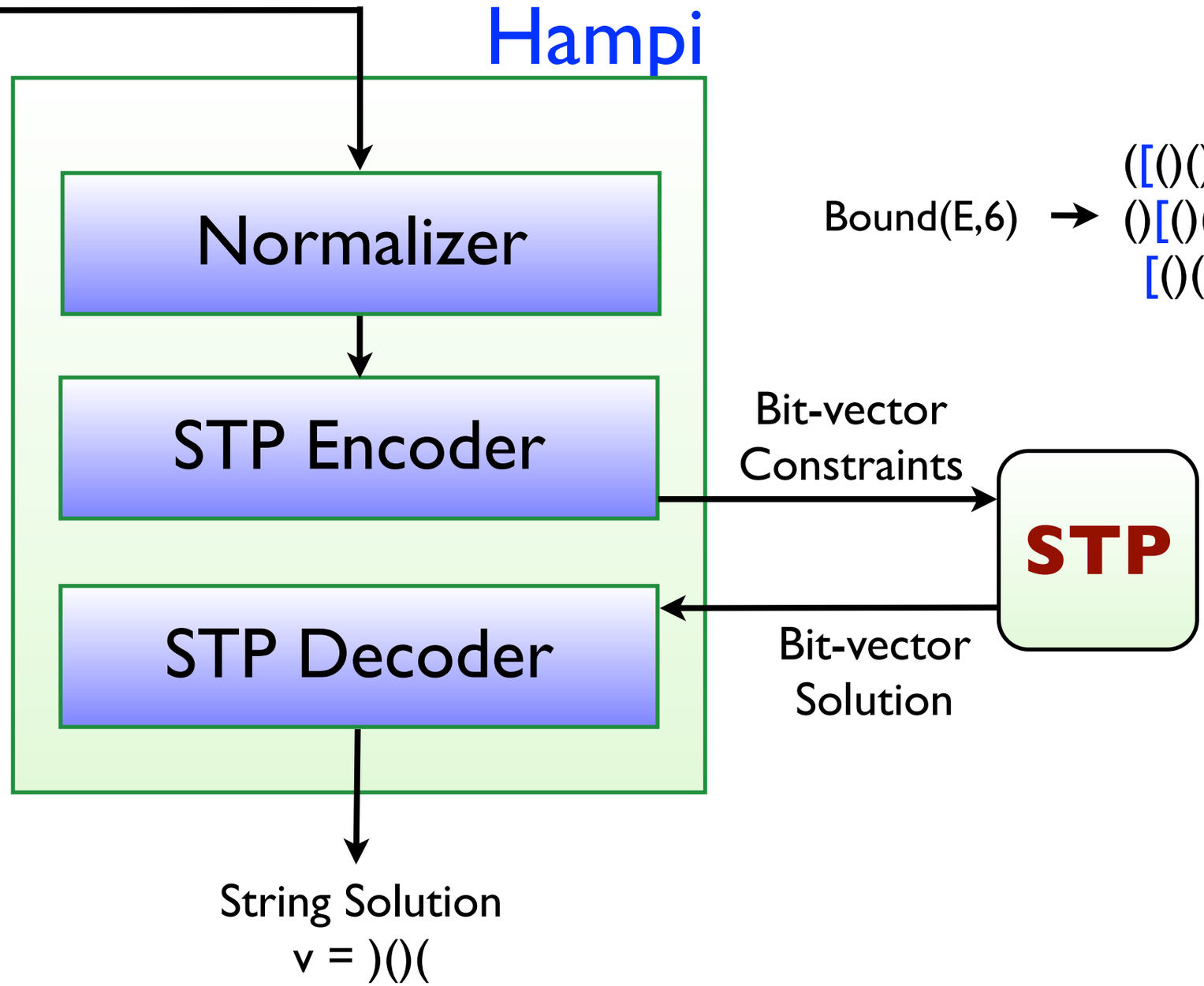
# Eager for BV and Lazy for Arrays



# How Hampi Works

# Unroll Bounded CFGs into Regular Exp.

```
var v : 4;

cfg E := "()" | E E | "(" E ")";

val q := concat( "(", v, ")");

assert q in E;
assert q contains "()()";
```

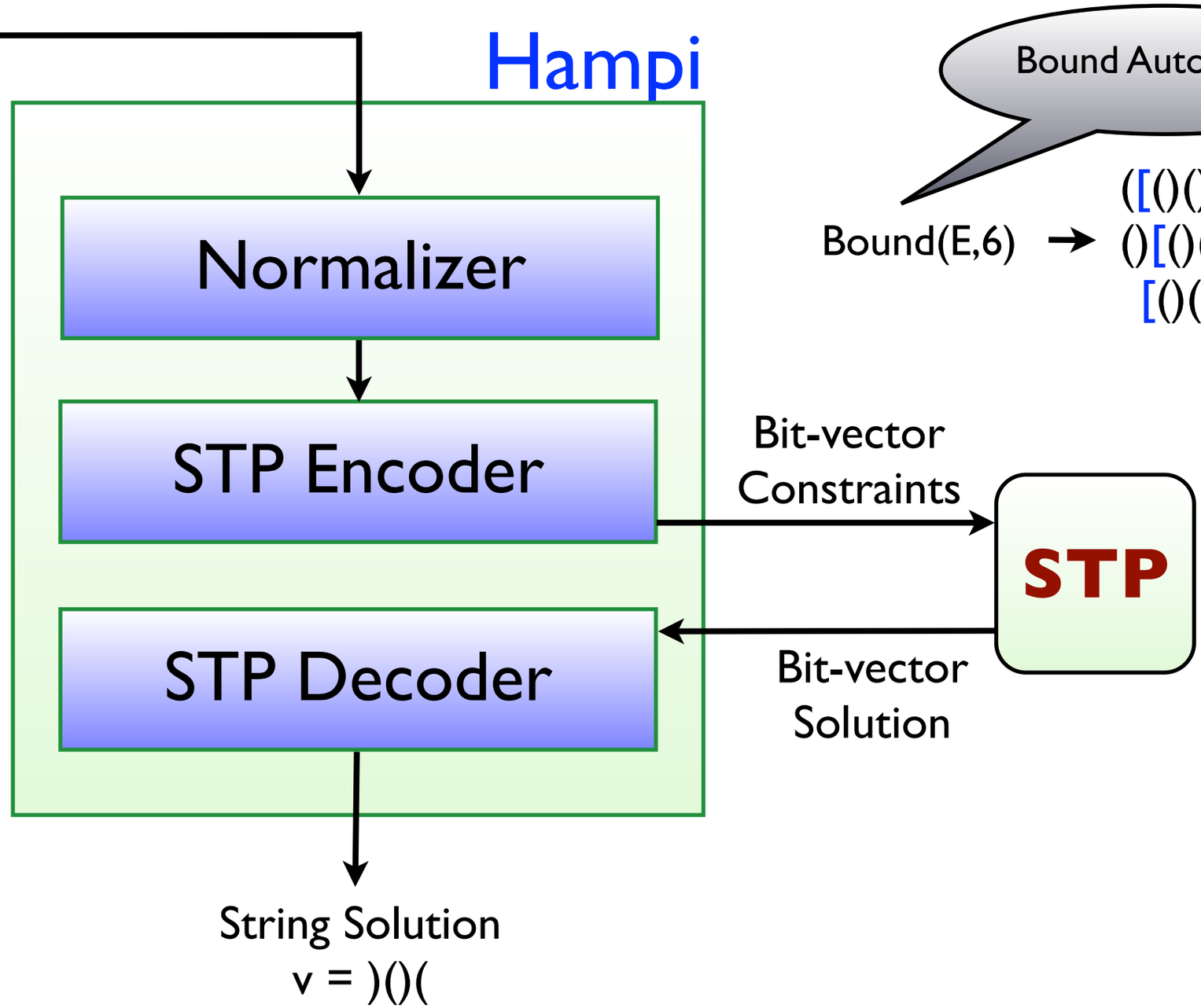



Bound(E,6) → ([()() + (())]) +
()[()() + (())] +
[()() + (())]()

# How Hampi Works

# Unroll Bounded CFGs into Regular Exp.

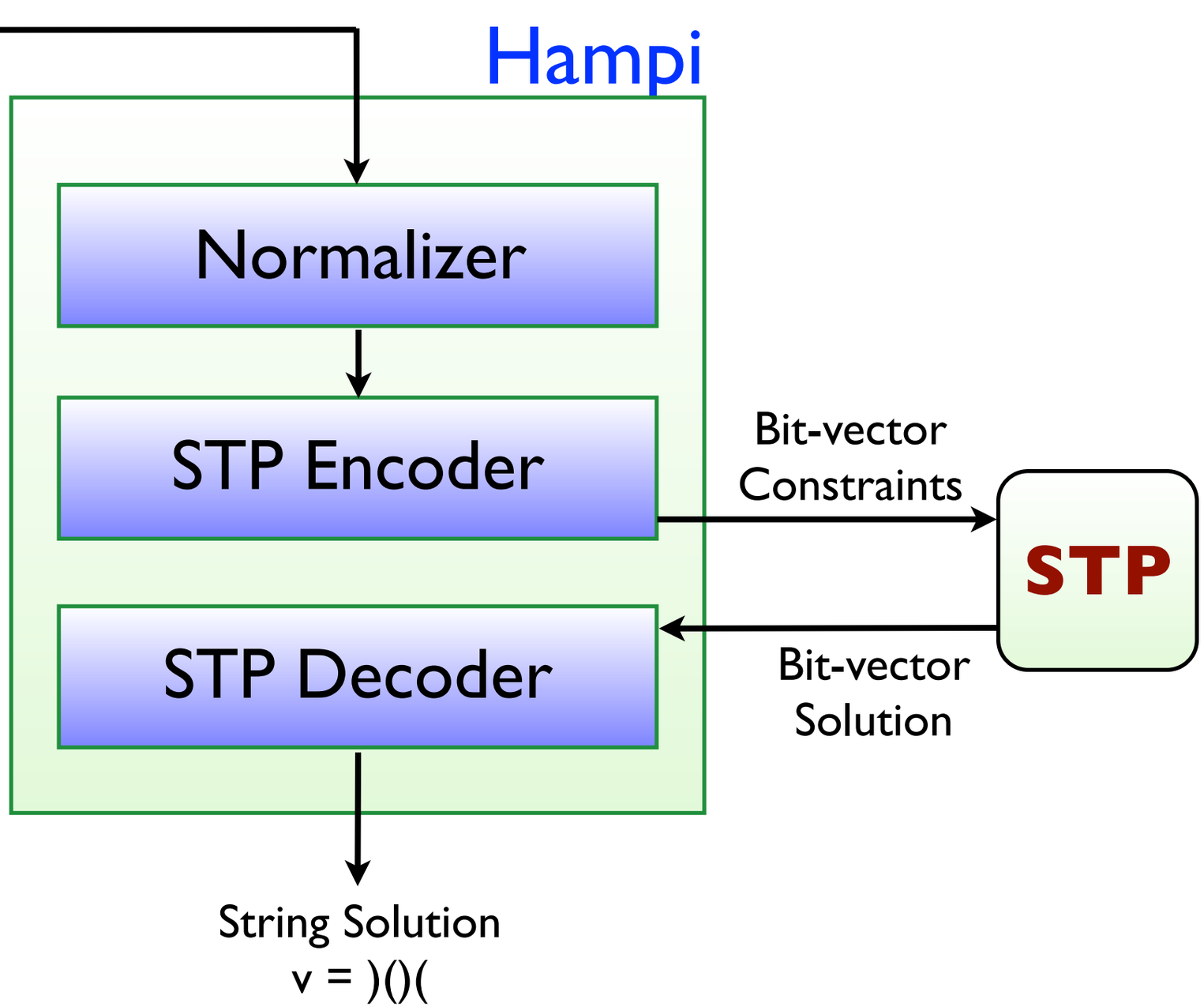
```
var v : 4;

cfg E := "()" | E E | "(" E ")";

val q := concat( "(", v, ")");

assert q in E;
assert q contains "()()";
```



# 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 "()()";
```




Find a 4-char string v:

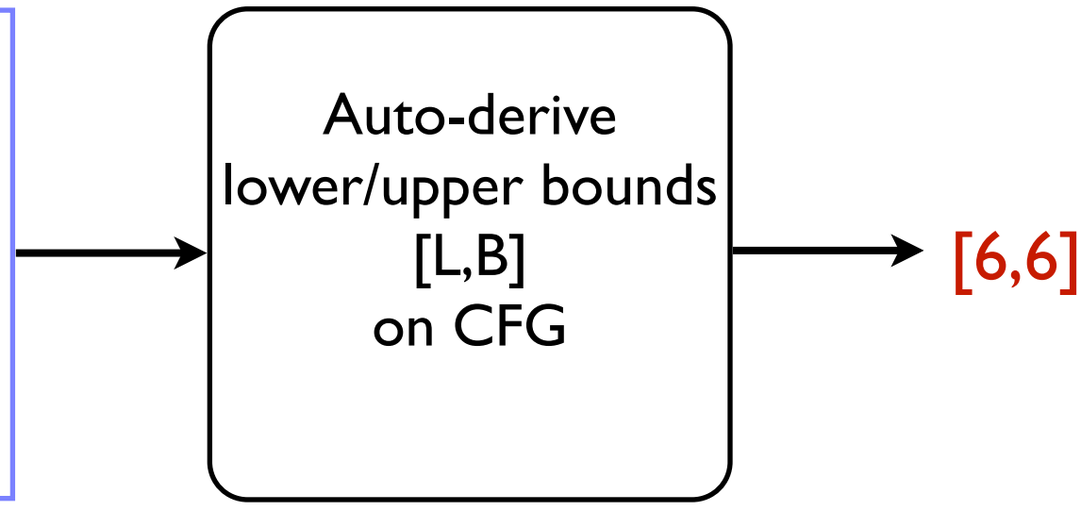
- (v) is in E
- (v) contains ()()

# How Hampi Works

## Unroll Bounded CFGs into Regular Exp.

Step 1:

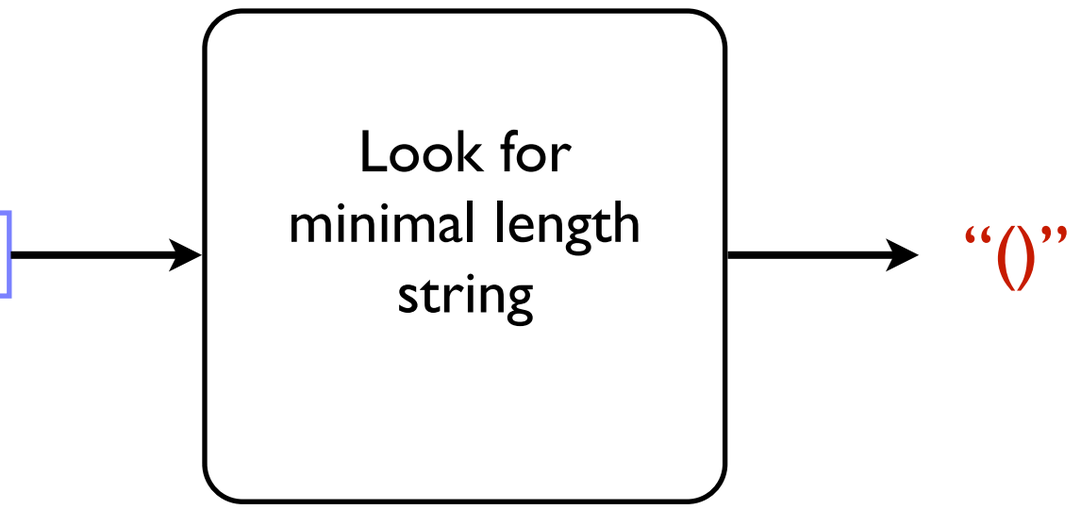
```
var v : 4;

cfg E := "()" | E E | "(" E ")";

val q := concat( "(", v, ")");

assert q in E;
assert q contains "()()";
```




[6,6]

Step 2:

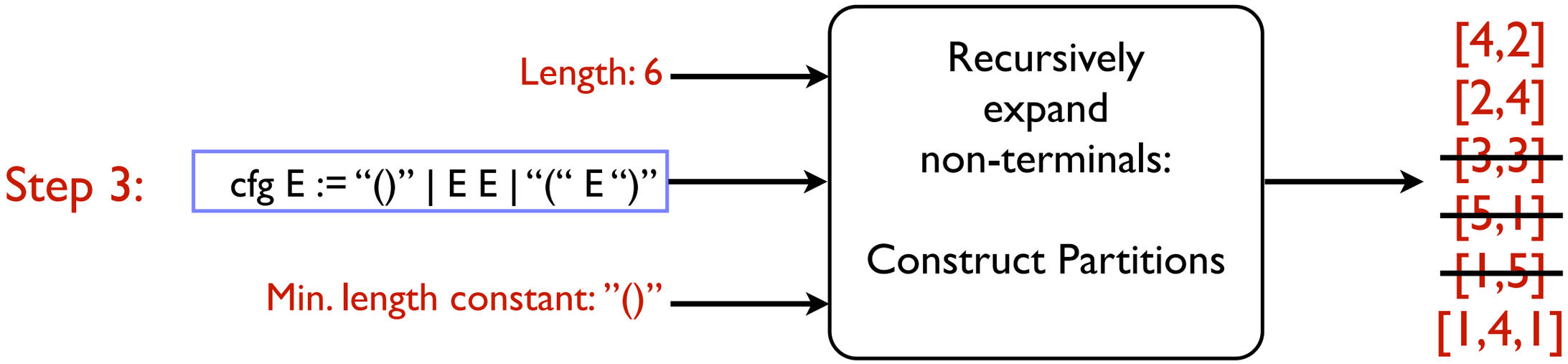
cfg E := "()" | E E | "(" E ")"

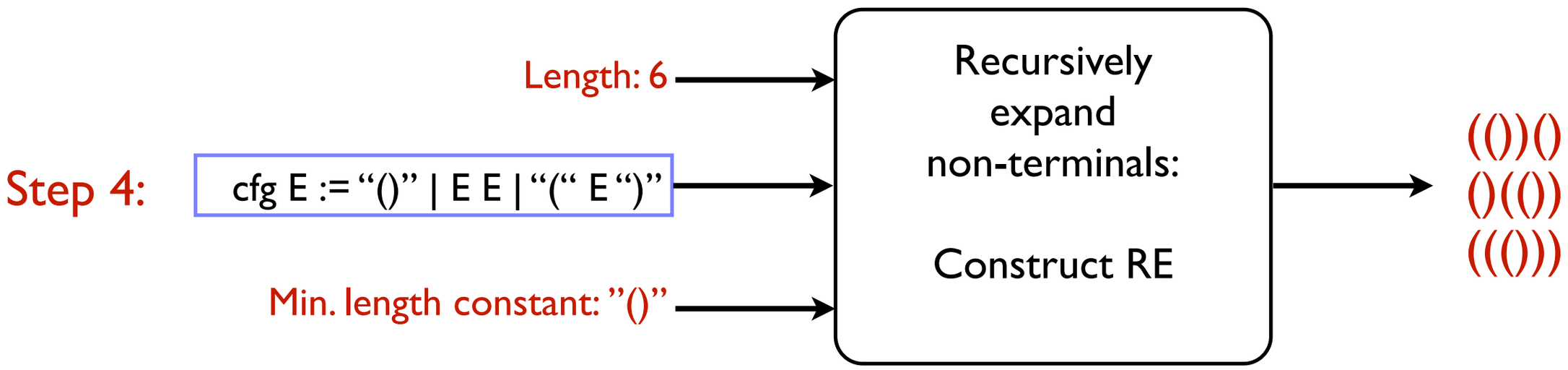



"()"

# How Hampi Works

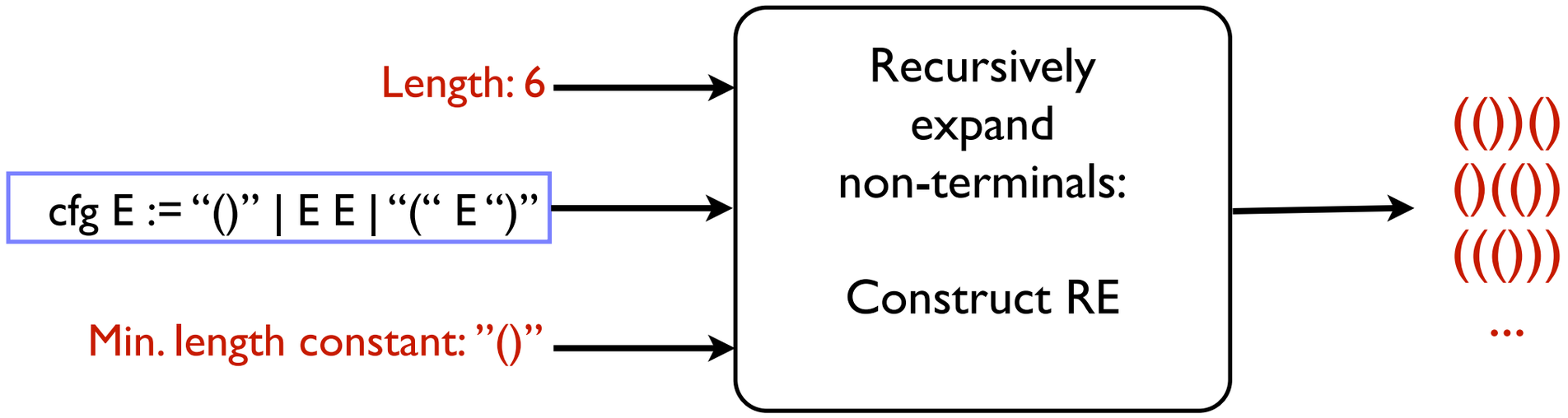
## Unroll Bounded CFGs into Regular Exp.



# Unroll Bounded CFGs into Regular Exp.

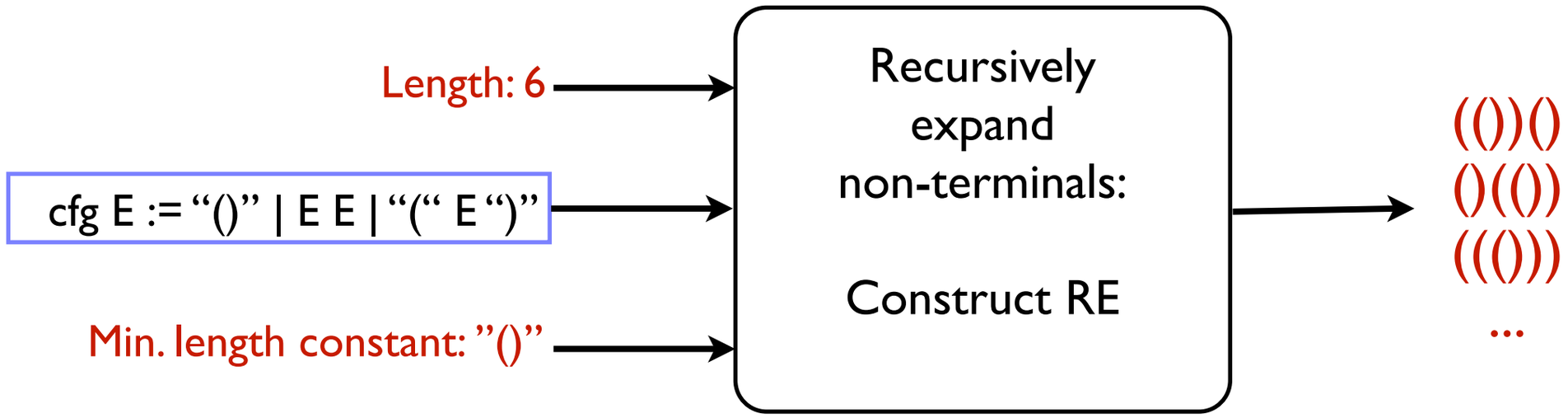
## Managing Exponential Blow-up




- Dynamic programming style
- Works well in practice

# Unroll Bounded CFGs into Regular Exp.

## Managing Exponential Blow-up




Bound(E,6) ⟶

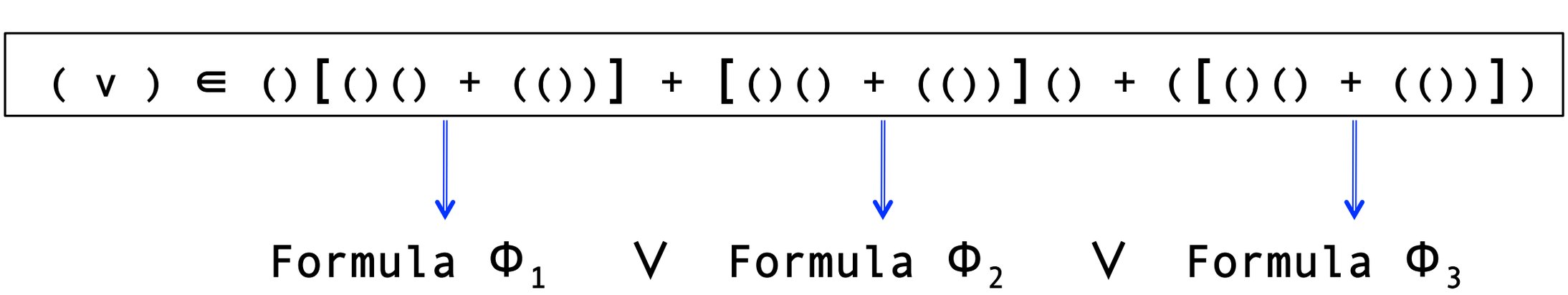
([()() + (())]) +
()[()() + (())] +
[()() + (())]()

# How Hampi Works

# Converting Regular Exp. into Bit-vectors

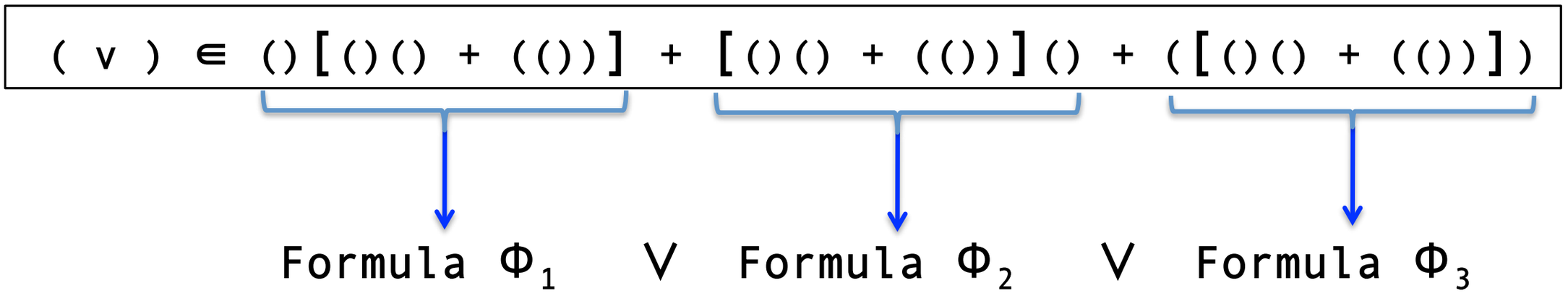
Encode regular expressions recursively

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



# How Hampi Works

# Converting Regular Exp. into Bit-vectors



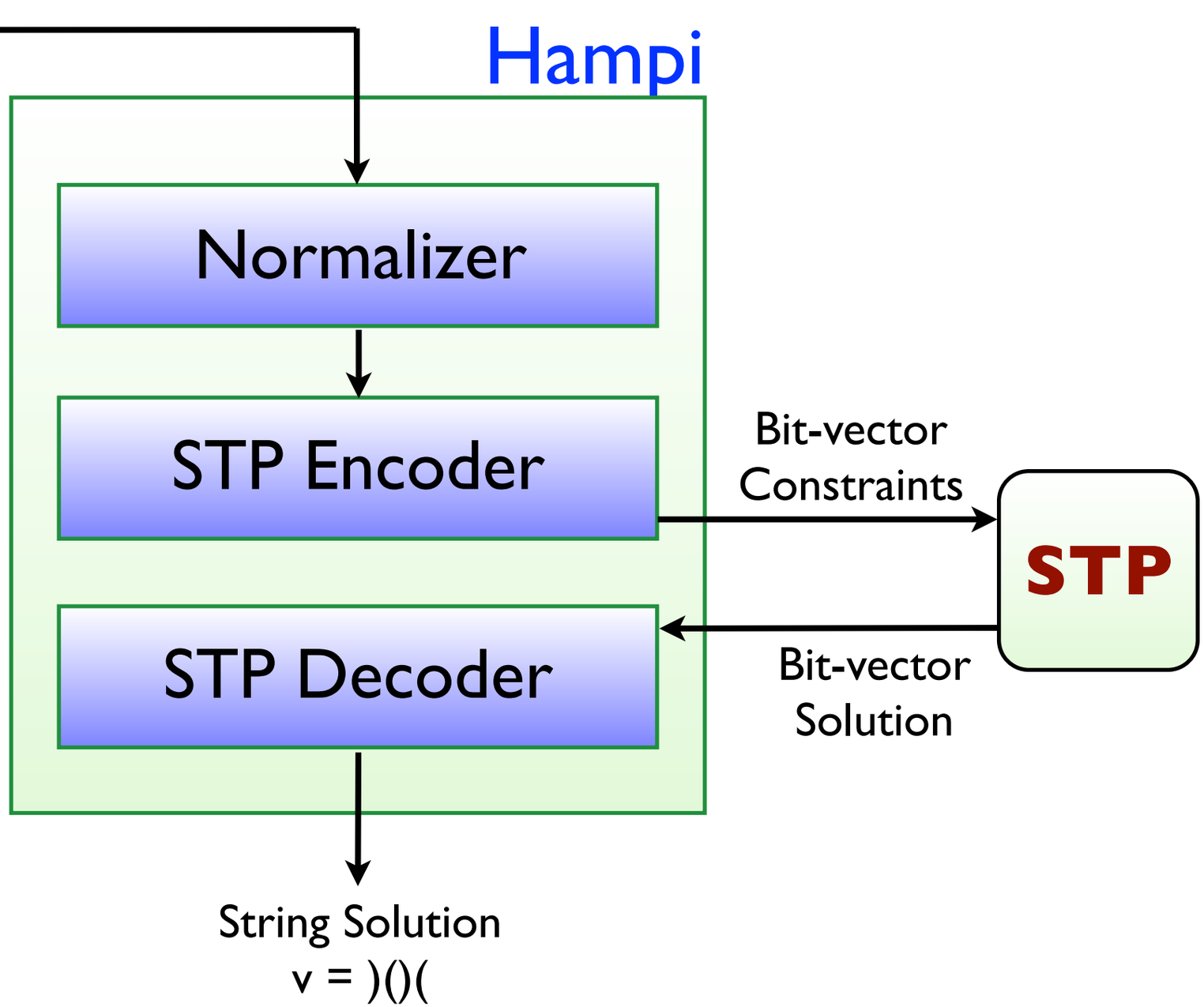

$B[0]=0 \wedge B[1]=1 \wedge \{B[2]=0 \wedge B[3]=1 \wedge B[4]=0 \wedge B[5]=1 \vee \ldots$

- 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 "()()";
```




Find a 4-char string v:

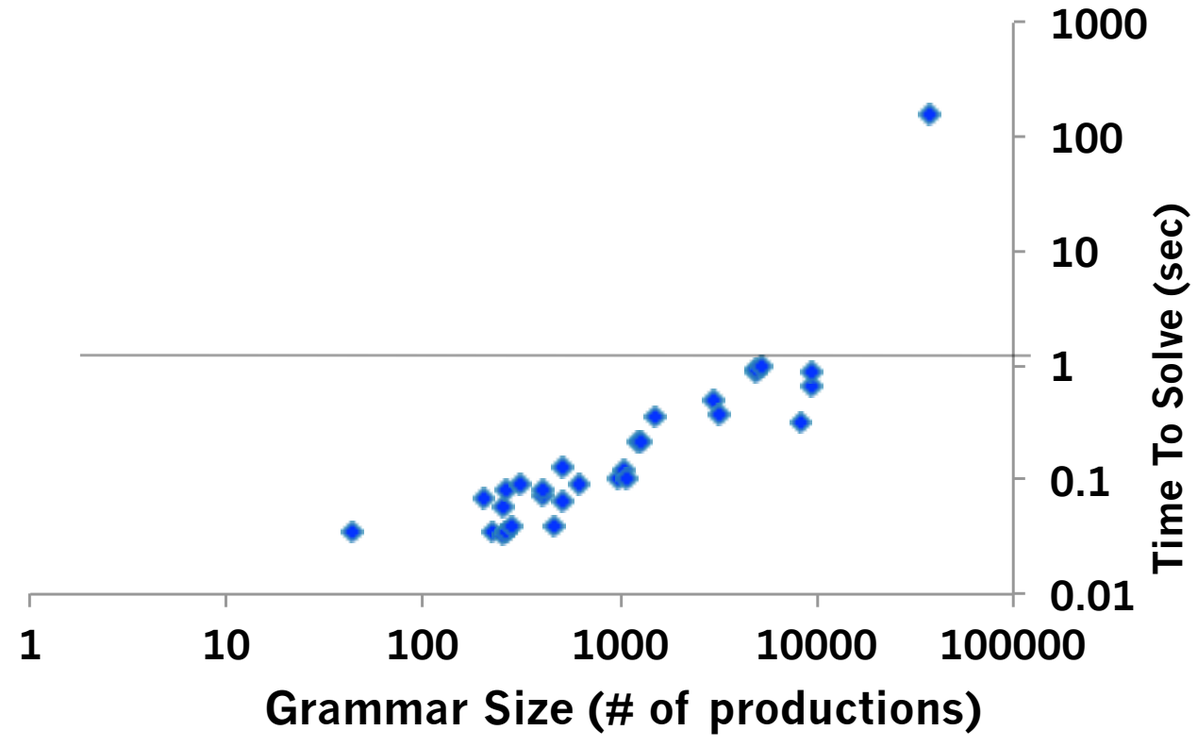
- (v) is in E
- (v) contains ()()

# Rest of the Talk

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

# HAMPI: Result 1

# 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 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 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)

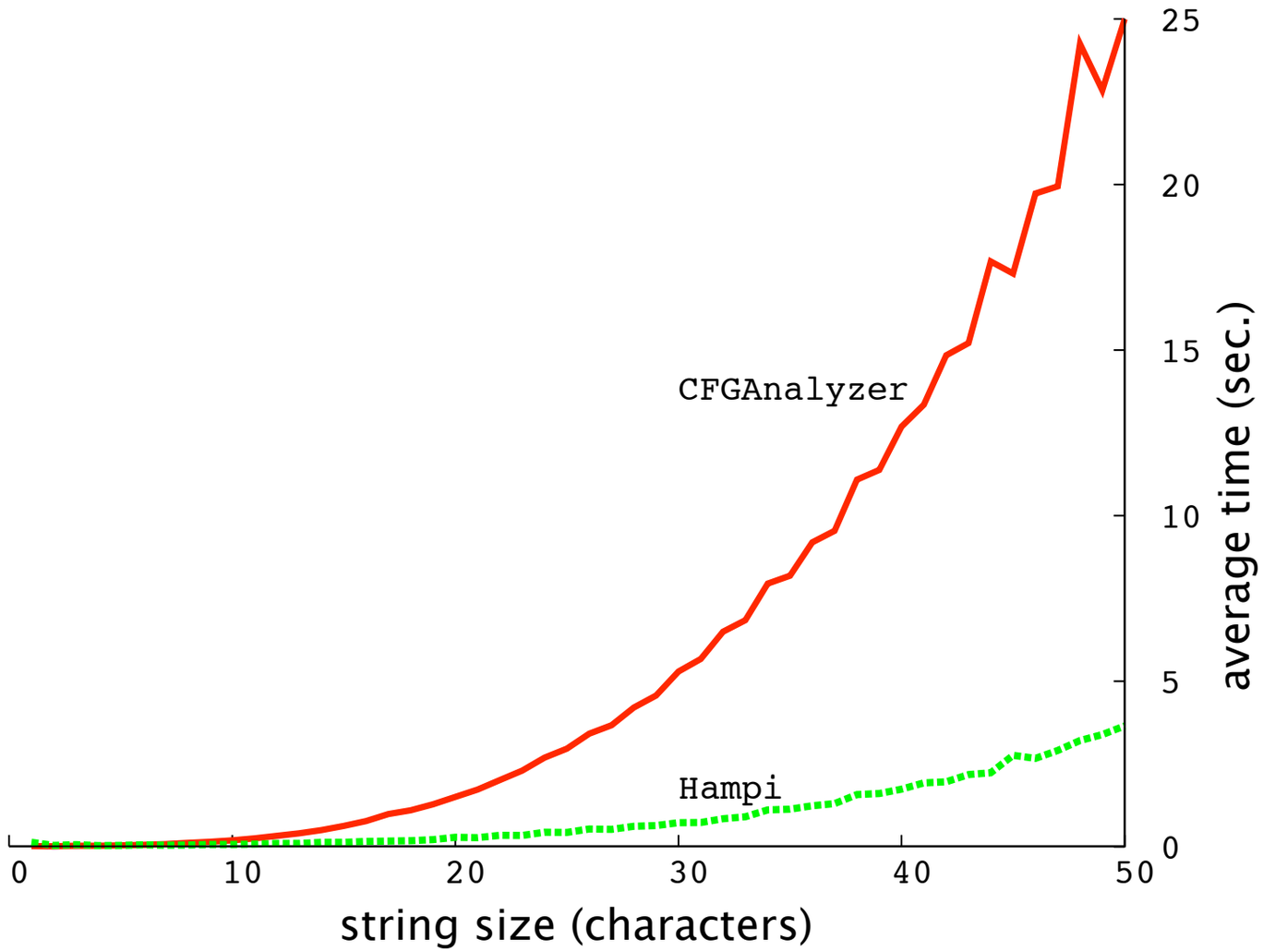
5 added to
US National Vulnerability DB

- 46% of constraints solved in < 1 second per constraint

- 100% of constraints solved in <10 seconds per constraint

# HAMPI: Result 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

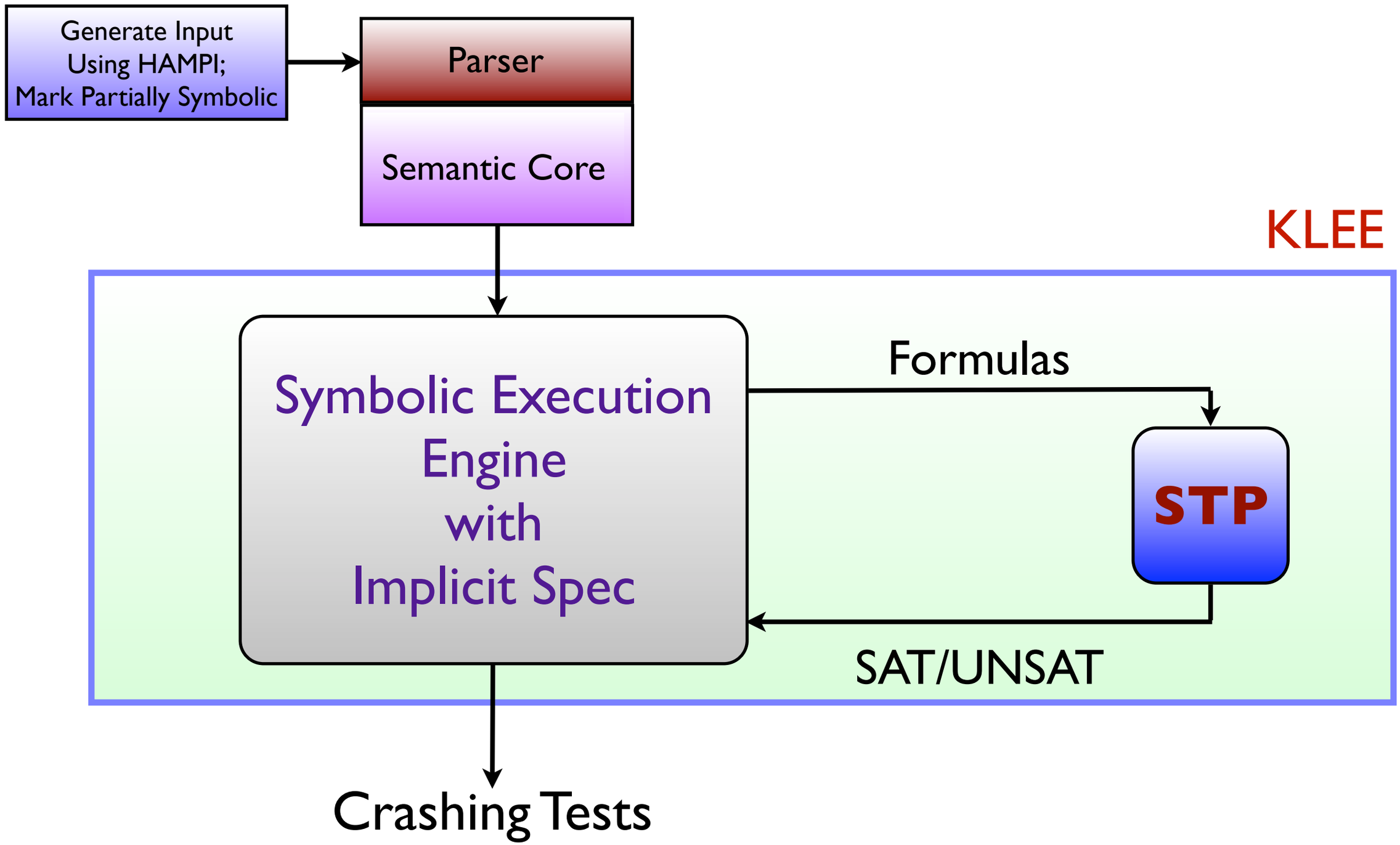
- HAMPI 100x faster than Rex (MSR)

- HAMPI 1000x faster than DPRLE (U.Virginia)

- Pieter Hooimeijer 2010 paper titled 'Solving String Constraints Lazily'

# HAMPI: Result 4

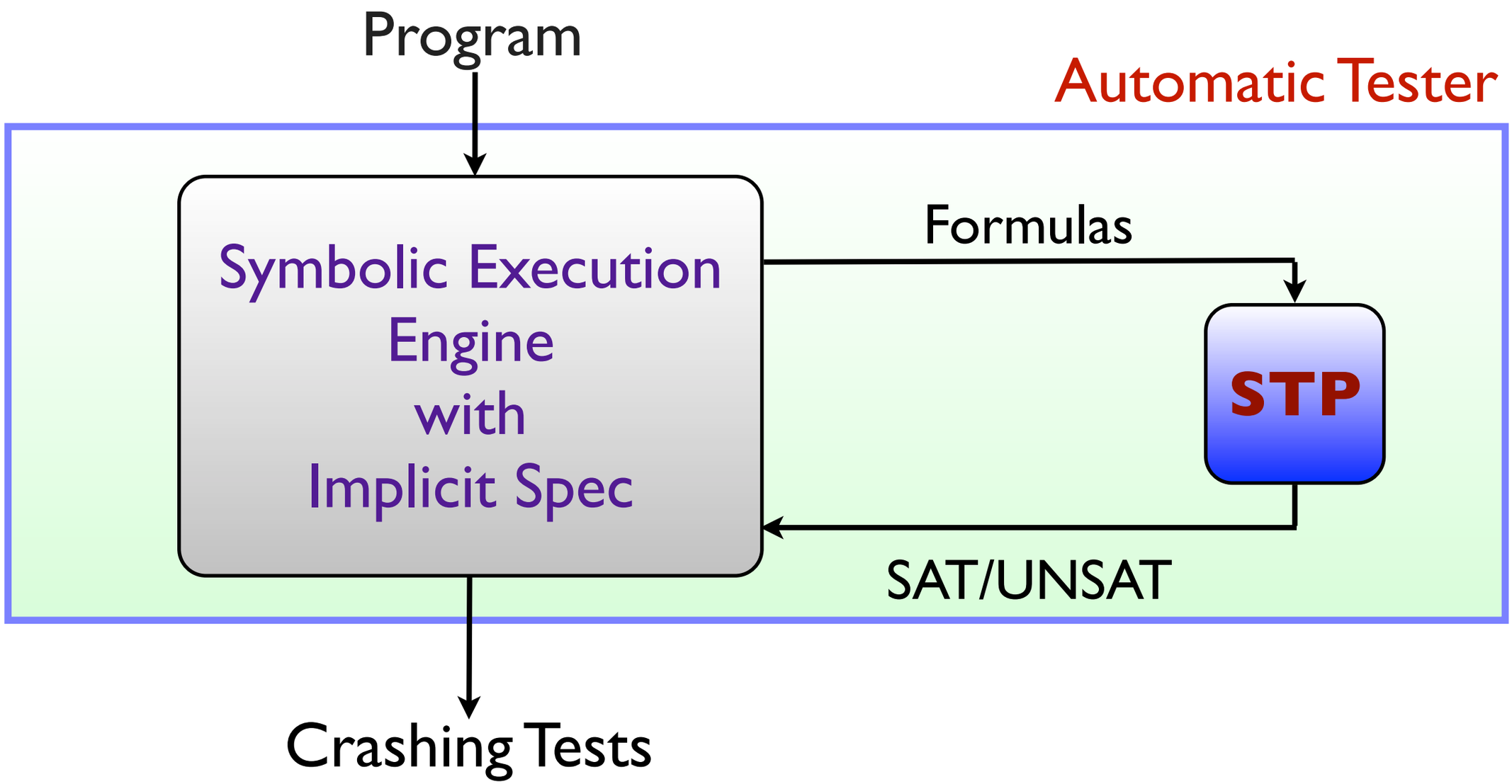
## Helping KLEE Pierce Parsers



# How to Automatically Crash Programs?

# KLEE: Concolic Execution-based Tester

Problem: Automatically generate crashing tests given only the code



# How to Automatically Crash Programs?

# KLEE: Concolic Execution-based Tester

Structured input processing code:
PDF Reader, Movie Player,...

```
Buggy_C_Program(int* data_field, int len_field) {

    int * ptr = malloc(len_field*sizeof(int));
    int i; //uninitialized

    while (i++ < process(len_field)) {
      //1. Integer overflow causing NULL deref
      //2. Buffer overflow
      *(ptr+i) = process_data(*(data_field+i));
    }
}
```

- Formula captures computation
- Tester attaches formula to capture spec

# How to Automatically Crash Programs?

# KLEE: Concolic Execution-based Tester

Structured input processing code:
PDF Reader, Movie Player,...

```
Buggy_C_Program(int* data_field, int len_field) {

    int * ptr = malloc(len_field*sizeof(int));
    int i; //uninitialized

    while (i++ < process(len_field)) {
      //1. Integer overflow causing NULL deref
      //2. Buffer overflow
      *(ptr+i) = process_data(*(data_field+i));
    }
}
```

Equivalent Logic Formula derived using
symbolic execution

```
data_field, mem_ptr :  ARRAY;
len_field : BITVECTOR(32); //symbolic
i,  j, ptr     : BITVECTOR(32);//symbolic
.
.
mem_ptr[ptr+i] = process_data(data_field[i]);
mem_ptr[ptr+i+1] = process_data(data_field[i+1]);
.
.
```

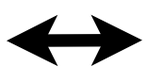
- Formula captures computation
- Tester attaches formula to capture spec

# How to Automatically Crash Programs?

# KLEE: Concolic Execution-based Tester

Structured input processing code:
PDF Reader, Movie Player,...

```
Buggy_C_Program(int* data_field, int len_field) {

    int * ptr = malloc(len_field*sizeof(int));
    int i; //uninitialized

    while (i++ < process(len_field)) {
      //1. Integer overflow causing NULL deref
      //2. Buffer overflow
      *(ptr+i) = process_data(*(data_field+i));
    }
}
```



Equivalent Logic Formula derived using
symbolic execution

```
data_field, mem_ptr :  ARRAY;
len_field : BITVECTOR(32); //symbolic
i,  j, ptr     : BITVECTOR(32);//symbolic
.
.
mem_ptr[ptr+i] = process_data(data_field[i]);
mem_ptr[ptr+i+1] = process_data(data_field[i+1]);
.
.
```

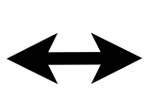
- Formula captures computation
- Tester attaches formula to capture spec

# How to Automatically Crash Programs?

# KLEE: Concolic Execution-based Tester

Structured input processing code:
PDF Reader, Movie Player,...

```
Buggy_C_Program(int* data_field, int len_field) {

    int * ptr = malloc(len_field*sizeof(int));
    int i; //uninitialized

    while (i++ < process(len_field)) {
      //1. Integer overflow causing NULL deref
      //2. Buffer overflow
      *(ptr+i) = process_data(*(data_field+i));
    }
}
```



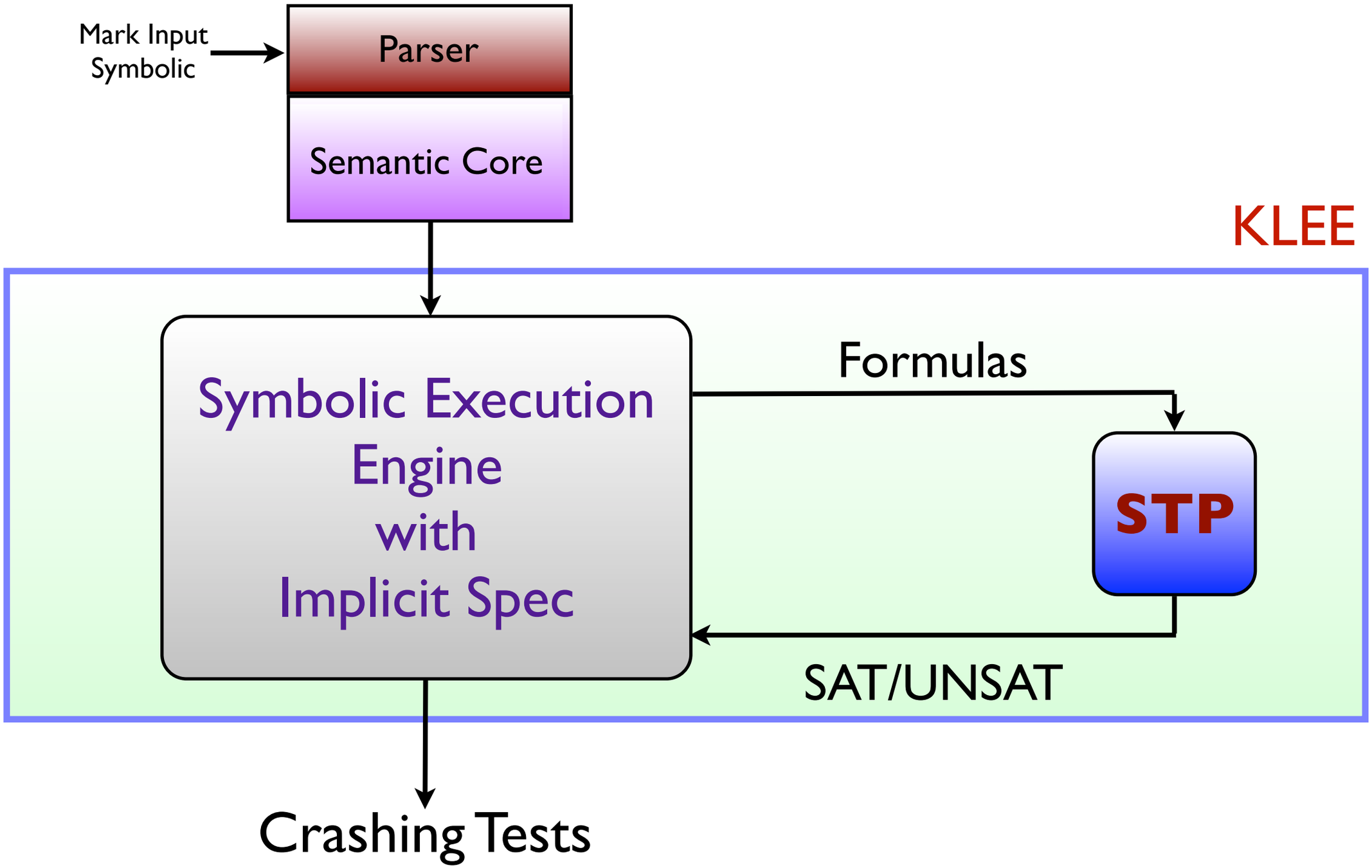
Equivalent Logic Formula derived using
symbolic execution

```
data_field, mem_ptr : ARRAY;
len_field : BITVECTOR(32); //symbolic
i,  j, ptr    : BITVECTOR(32);//symbolic
.
.
mem_ptr[ptr+i] = process_data(data_field[i]);
mem_ptr[ptr+i+1] = process_data(data_field[i+1]);
.
.
//INTEGER OVERFLOW QUERY
0 <= j <= process(len_field);
ptr + i + j = 0?
```

- Formula captures computation
- Tester attaches formula to capture spec

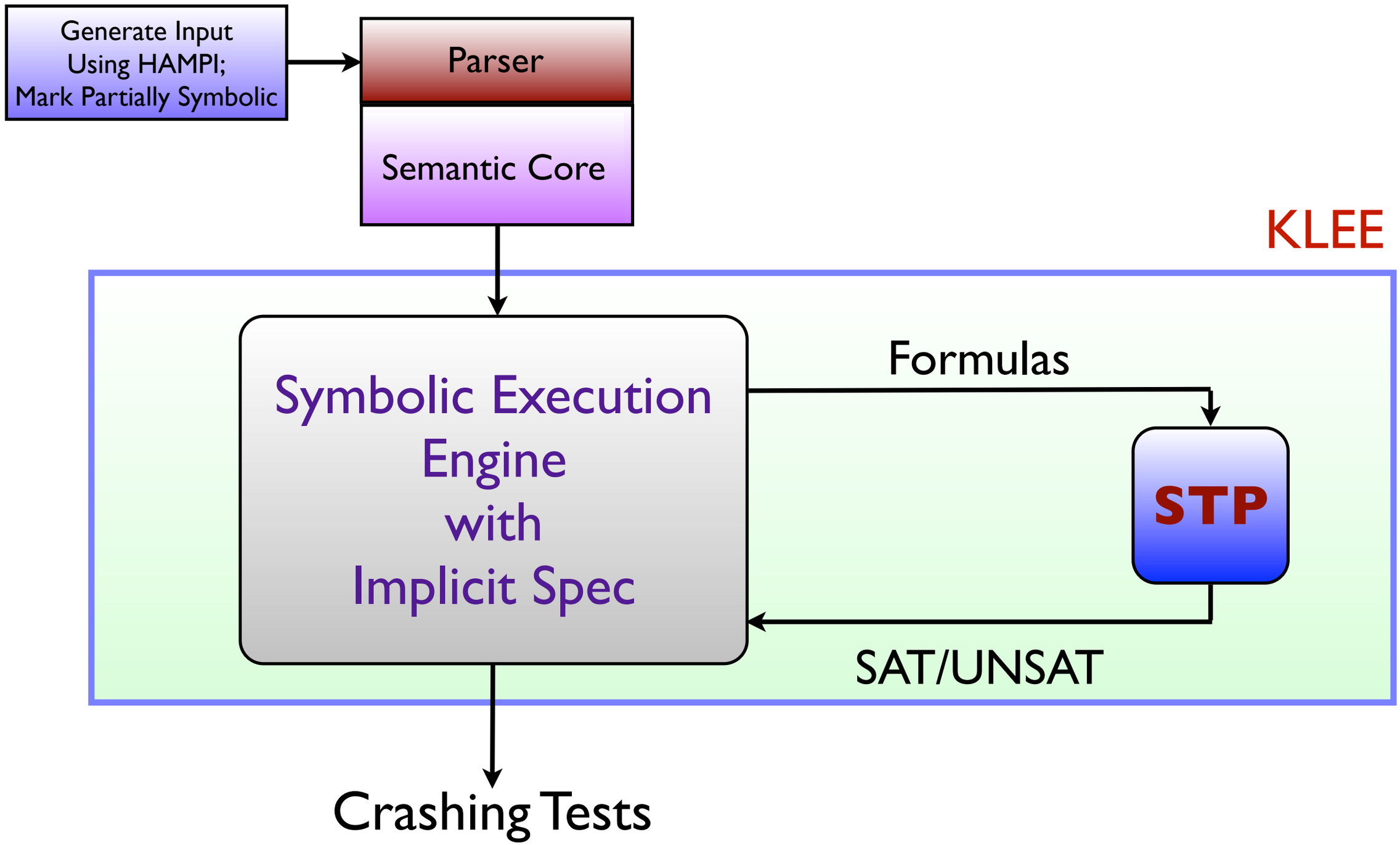
# HAMPI: Result 4

## Helping KLEE Pierce Parsers



# HAMPI: Result 4

# Helping KLEE Pierce Parsers



# HAMPI: Result 4

# Helping KLEE Pierce Parsers

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

# Impact of Hampi: Notable Projects

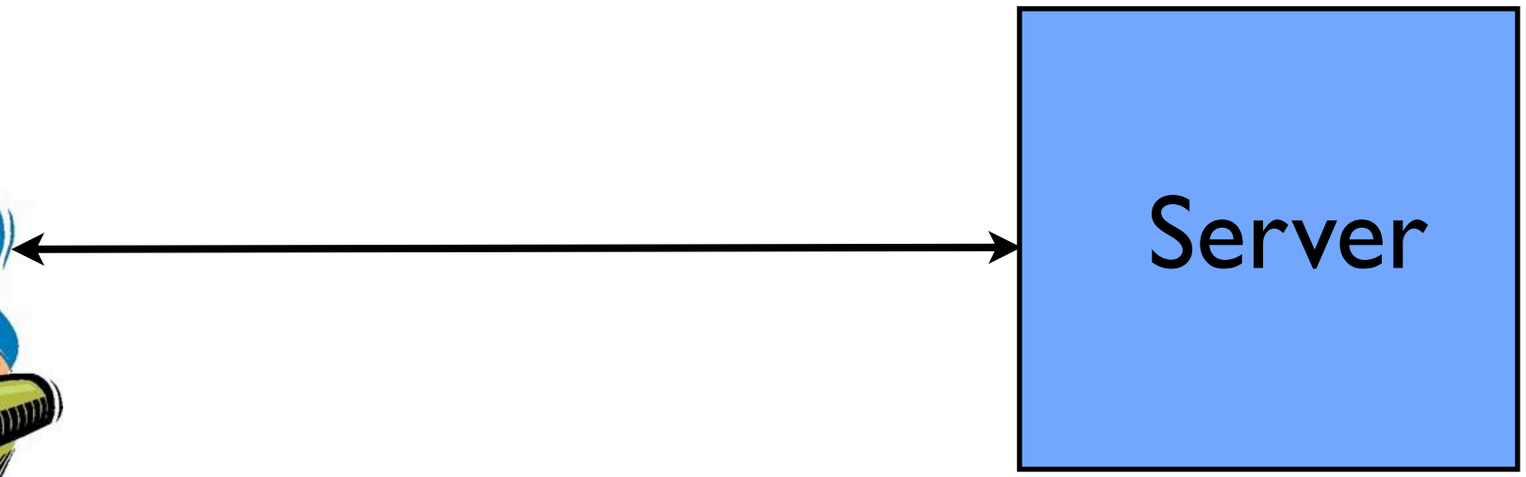
| Category | Research Project | Project Leader/Institution |
|---|---|---|
| Static Analysis | SQL-injection vulnerabilities | Wasserman & Su/UC, Davis |
| Security Testing | Ardilla for PHP (SQL injections, cross-site scripting) | Kiezun & Ernst/MIT |
| Concolic Testing | Klee<br>Kudzu<br>NoTamper |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

# Impact of Hampi: Notable Projects

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

# Impact of Hampi: Notable Projects

## NoTamper


- Client-side checks (C), no server checks
- Find solutions $S_1, S_2, \ldots$ to C, and solutions $E_1, E_2, \ldots$ to ~C by calling HAMPI
- $E_1, E_2, \ldots$ are candidate exploits
- Submit (S1, E1),... to server
- If server response same, ignore
- If server response differ, report error

# Related Work (Practice)

| Tool Name | Project Leader/ Institution | Comparison with HAMPI |
|---|---|---|
| Rex | Bjorner, Tillman, Vornkov et al. (Microsoft Research, Redmond) | • HAMPI<br>+ Length+Replace($s_1$,$s_2$,$s_3$)<br>- CFG<br>• Translation to int. linear arith. (Z3) |
| Mona | Karlund et al. (U. of Aarhus) | • Can encode HAMPI & Rex<br>• User work<br>• Automata-based<br>• Non-elementary |
| DPRLE | Hooimeijer (U. of Virginia) | • Regular expression constraints |

# Related Work (Theory)

| Result | Person (Year) | Notes |
|---|---|---|
| Undecidability of Quantified Word Equations | Quine (1946) | Multiplication reduced to concat |
| Undecidability of Quantified Word Equations with single alternation | Durnev (1996), G. (2011) | 2-counter machines reduced to words with single quantifier alter. |
| Decidability (PSPACE) of QF Theory of Word Equations | Makanin (1977)<br>Plandowski (1996, 2002/06) | Makanin result very difficult<br>Simplified by Plandowski |
| Decidability (PSPACE-complete) of QF Theory of Word Equations + RE | Schultz (1992) | RE membership predicate |
| QF word equations + Length() (?) | Matiyasevich (1971) | Unsolved<br>Reduction to Diophantine |
| QF word equations in solved form + Length() + RE | G. (2011) | Practical |

# Future of HAMPI & STP

- HAMPI will be combined with STP
  - Bit-vectors and Arrays
  - Integer/Real Linear Arithmetic
  - Uninterpreted Functions
  - Strings
  - Floating Point
  - Non-linear
- Additional features planned in STP
  - UNSAT Core
  - Quantifiers
  - Incremental
  - DPLL(T)
  - Parallel STP
  - MAXSMT?
- Extensibility and hackability by non-expert

# Future of Strings

- Strings SMTization effort started
  - Nikolaj Bjorner, G.
  - Andrei Voronkov, Ruzica Piskac, Ting Zhang
  - Cesare Tinelli, Clark Barrett, Dawn Song, Prateek Saxena, Pieter Hooimeijer, Tim Hinrichs

- SMT Theory of Strings
  - Alphabet (UTF, Unicode,...)
  - String Constants and String Vars (parameterized by length)
  - Concat, Extract, Replace, Length Functions
  - Regular Expressions, CFGs (Extended BNF)
  - Equality, Membership Predicate, Contains Predicate

- Applications
  - Static/Dynamic Analysis for Vulnerability Detection
  - Security Testing using Concolic Idea
  - Formal Methods
  - Synthesis

# Conclusions & Take Away

- String solvers essential for testing, analysis, vulnerability detection
  - String applications in C/C++/Java/C#
  - Web applications in PHP/JavaScript (client and server-side)
- HAMPI
  - Multiple string vars, constants
  - Concat/extract function
  - Equality between string terms
  - Membership predicate over RE/CFGs
  - 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 (can we be more systematic?)

# Topics Covered Today

- HAMPI Logic: A Theory of Strings
- HAMPI-based vulnerability detection app
- How HAMPI works
- Another HAMPI-based app: Tamper Detection
- Experimental results (Static, Dynamic, Competing tools, KLEE)
- Related work (Kaluza, Rex,...)
- Future of strings & SMTization

# Key Contributions

## http://people.csail.mit.edu/vganesh

| **Name** | **Key Concept** | **Impact** | **Pubs** |
|---|---|---|---|
| **STP**<br>Bit-vector & Array Solver[1,2] | Abstraction-refinement for Solving | Concolic Testing | CAV 2007<br>CCS 2006<br>TISSEC 2008 |
| **HAMPI**<br>String Solver[1] | App-driven Bounding for Solving | Analysis of Web Apps | ISSTA 2009[3]<br>TOSEM 2011<br>(CAV 2011) |
| **Taint-based Fuzzing** | Information flow is cheaper than concolic | Scales better than concolic | ICSE 2009 |
| **Automatic Input Rectification** | Acceptability Envelope: Fix the input, not the program | New way of approaching SE | Under Submission |

1. 100+ research projects use STP and HAMPI
2. STP won the SMTCOMP 2006 and 2010 competitions for bit-vector solvers
3. HAMPI: ACM Best Paper Award 2009
4. Retargetable Compiler (DATE 1999)
5. Proof-producing decision procedures (TACAS 2003)
6. Error-finding in ARBAC policies (CCS 2011)