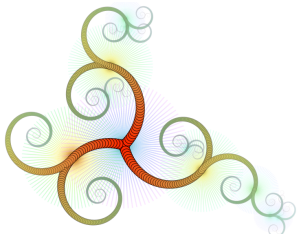
Massachusetts
Institute of
Technology

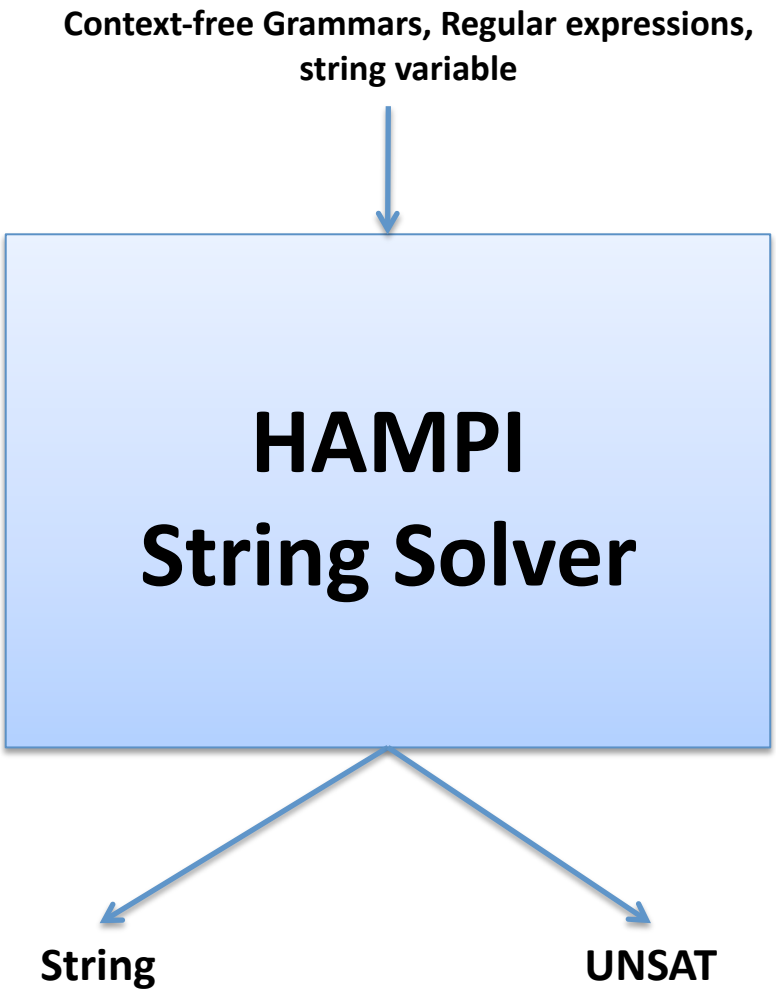
HAMPI
A Solver for String Constraints

Vijay Ganesh
MIT

(With Adam Kiezun, Philip Guo,
Pieter Hooimeijer and Mike Ernst)



The Problem



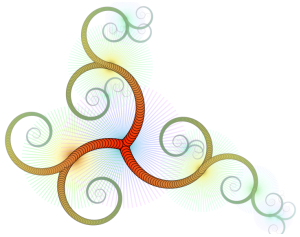
- Emptiness problem for an intersection of Context-free Grammars

$$s \text{ in } (L_1 \cap \ldots \cap L_N)$$

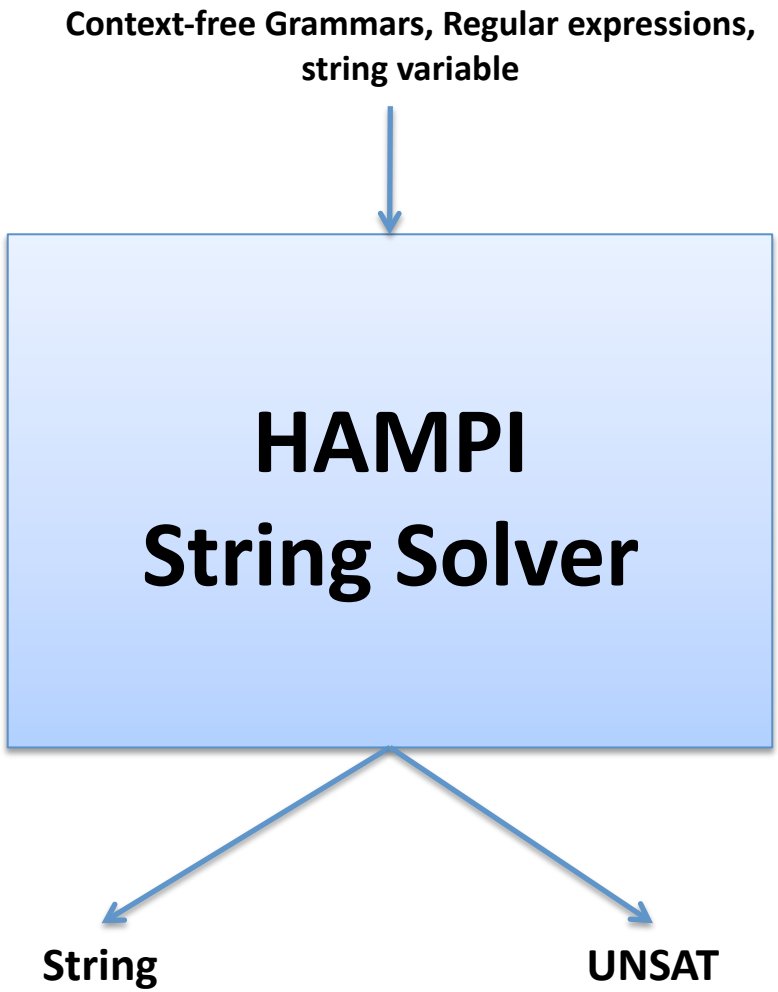
  where
  - s is bounded
  - s contains some substring

- Extended Backus-Naur Form

- Different from string matching



A Problem Instance

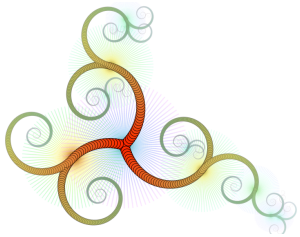



```
var v:4;

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

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

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

"Find a 4-character string v, such that:
- (v) has balanced parentheses, and
- (v) contains substring ()()"

HAMPI finds satisfying assignment
v =)()(

Hence, q = ()()()



HAMPI
A Novel String Solver

Analyses of string programs
- Formal Methods
- Testing
- Program Analysis

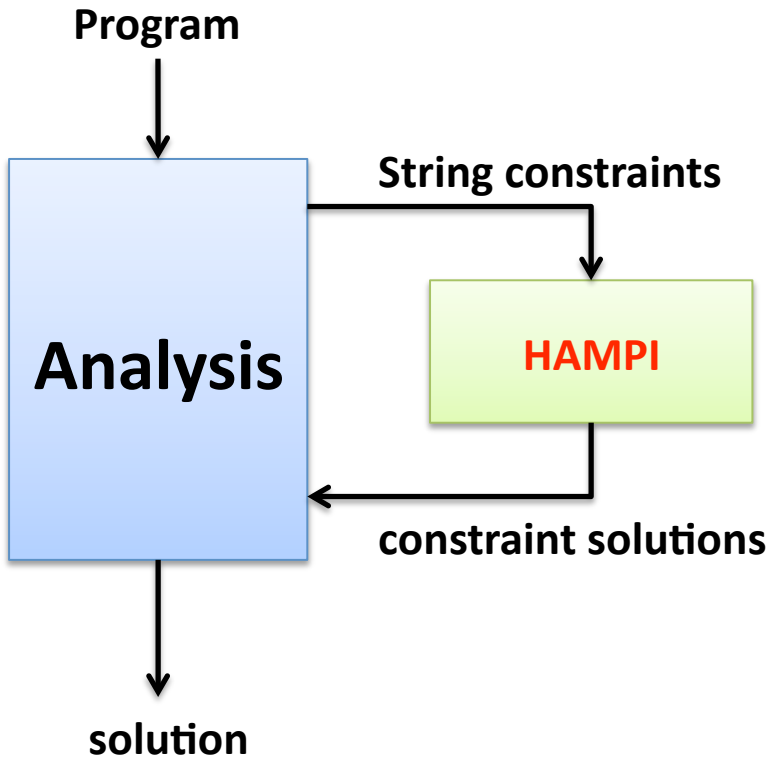
Efficient

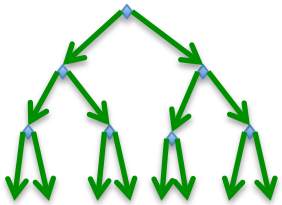
Expressive

Robust and easy to use

Tested on many diverse apps

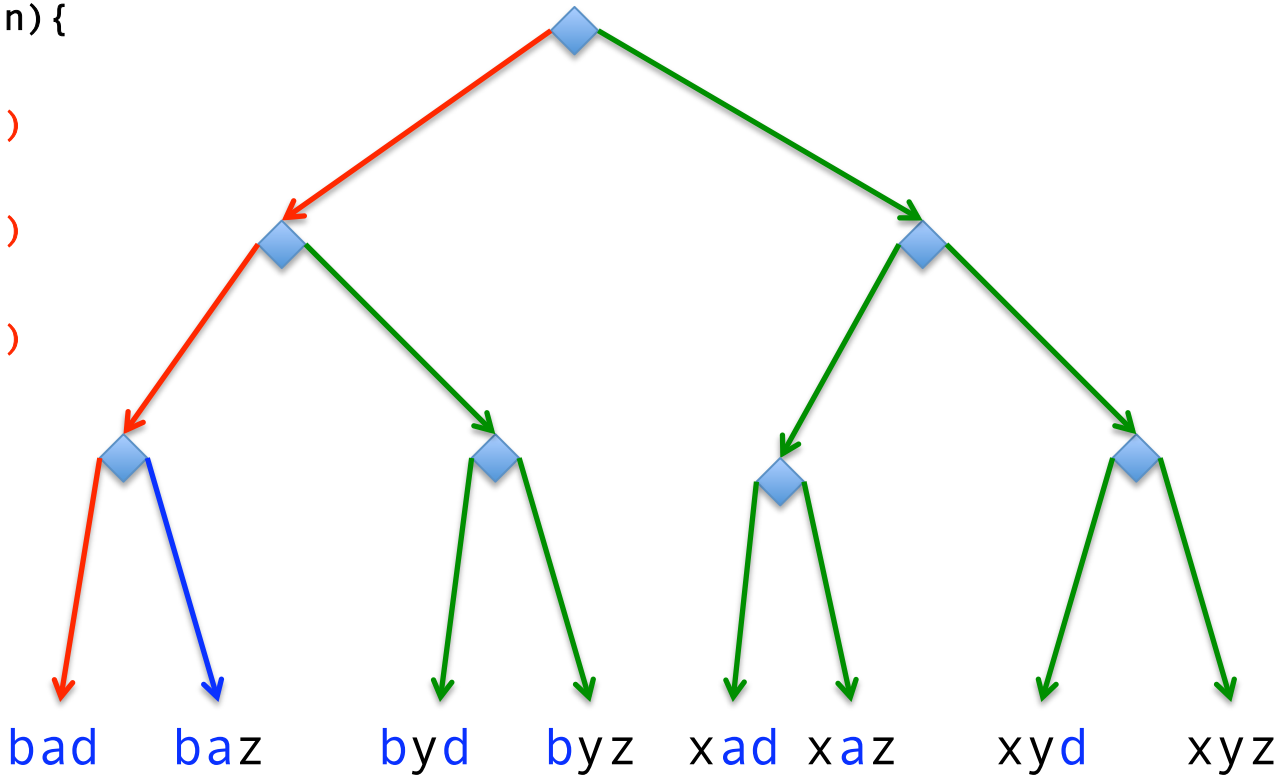
Applied to important and hard problems



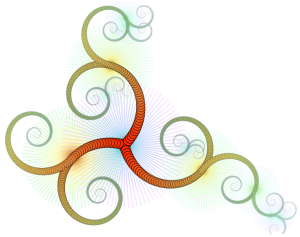


HAMPI for Dynamic Symbolic Testing

```
void main(char[] in){
  int count=0;
  if (in[0] == 'b')
    count++;
  if (in[1] == 'a')
    count++;
  if (in[2] == 'd')
    count++;
  if (count == 3)
    ERROR;
}
```



Concolic testing creates inputs for all program paths (assuming finite loops)



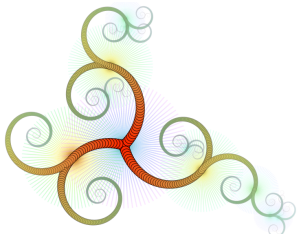
HAMPI for Concolic Testing

✔ Key to success: Efficient, expressive constraint solver

✔ HAMPI can also be used to produce structured inputs that skip parsing and penetrate deep during concolic testing

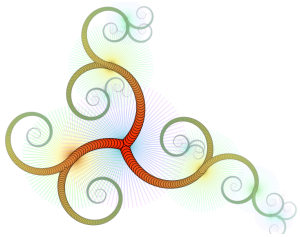
✔Godefroid et al. (DART,SAGE)

✔ EXE, KLEE, CUTE, CREST, SmartFuzz etc.



Typical HAMPI Applications

- Constraints generated by symbolic Analyses of string programs
  - Concolic Testing
  - Formal Methods
  - Program Analysis
  - Checking whether input sanitation code actually works
  - Applied to PHP scripts, JavaScript, C/Java etc.
- Automatic generation of structured inputs with constraints
  - String inputs for programs
  - Document files, e.g. PDF, to test PDF reader
  - SQL commands with attack vectors
  - Programming language source files



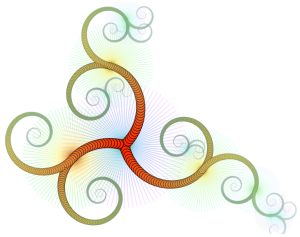
HAMPI Results Summary

✔ Expressive: Supports context-free grammars, regular expression operations

✔ Efficient: ~7x faster on-average, and
Often 100s of times faster than CFGAnalyzer

✔ Effectively enabled dynamic systematic testing (concolic testing) of real-world string-manipulating programs

✔ Already plugged into important analysis infrastructures, such as NASA Java PathFinder



HAMPI Results Summary

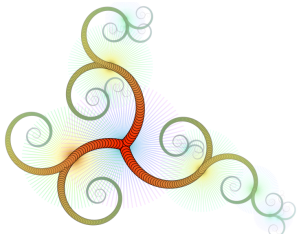
✔ SQL injection vulnerability detection using static analysis

✔ SQL injection vulnerability generation using dynamic analysis (Ardilla tool, ICSE 2009)

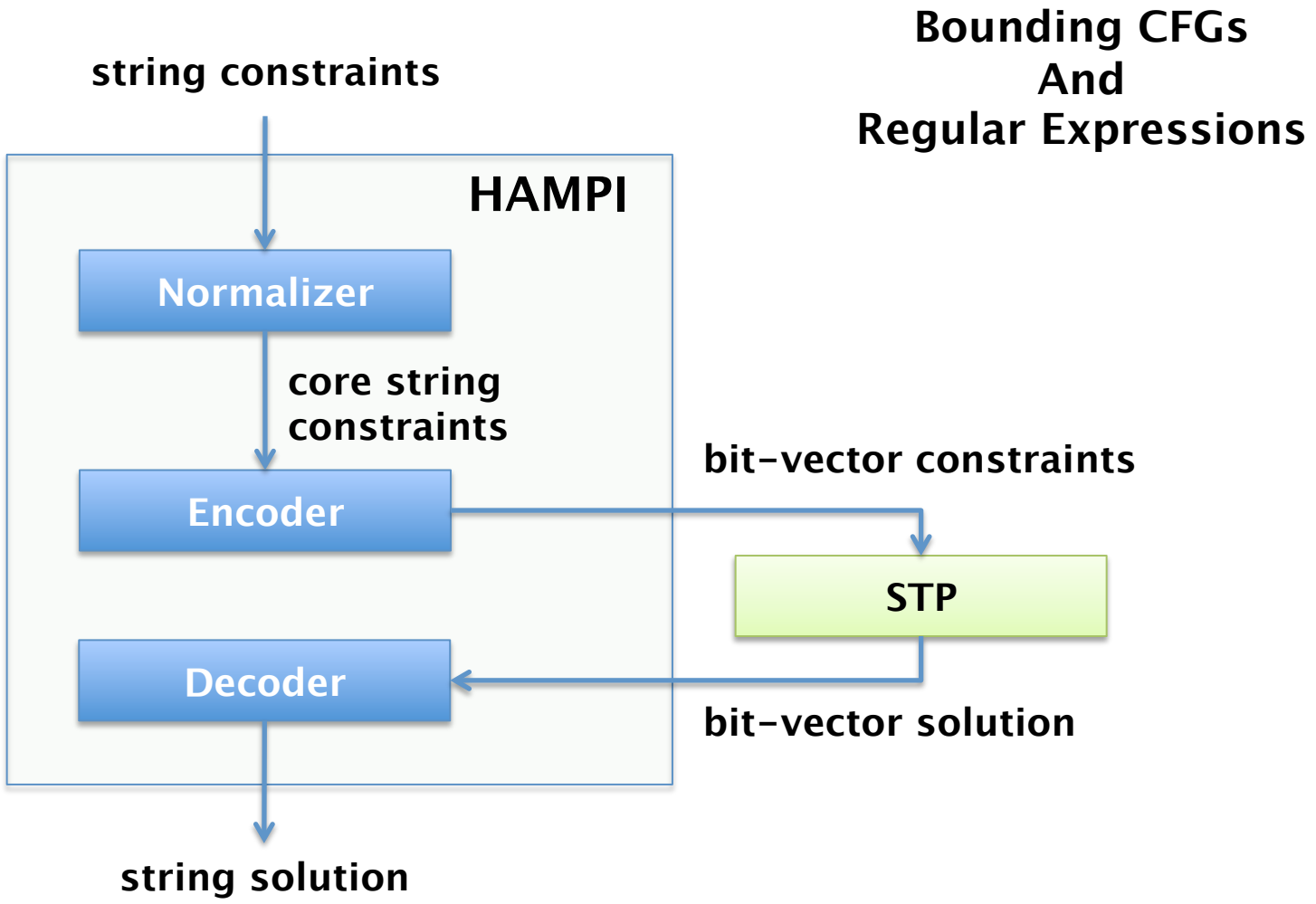
- ✔ 60 attacks (23 SQL injection, 37 XSS) on 5 PHP applications (300K+ LOC)

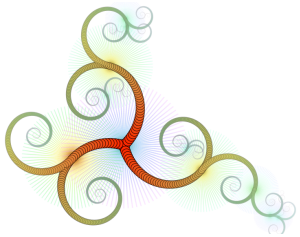
✔Automatic generation of structured inputs for Klee concolic tester, improved code coverage and bug finding

✔ Classic decision problems for context-free grammars



HAMPI Internals





HAMPI: Bounding is GOOD

more expressive

context-free	$L_1 \cap \ldots \cap L_N$	Undecidable
regular	$R_1 \cap \ldots \cap R_N$	PSPACE-complete
bounded regular	$r_1 \cap \ldots \cap r_N$	NP-complete

more tractable

bound(any language) → bounded regular

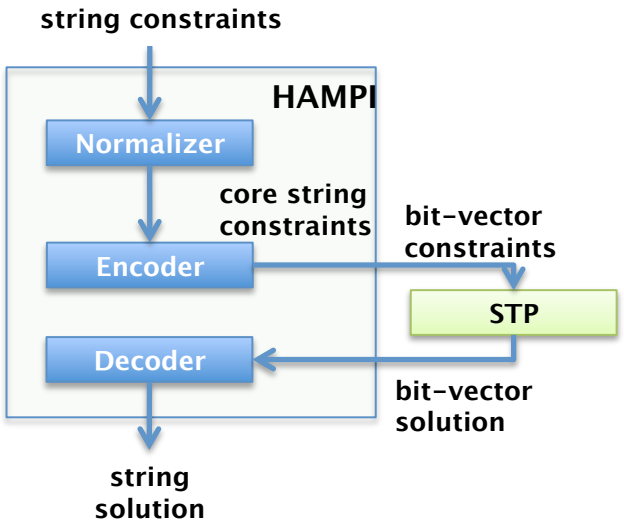
Key HAMPI idea:

1. Bound length of strings for high expressiveness, efficiency
2. Typical applications require short solutions

HAMPI Example

```
var v:4;

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

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

assert q in bound(E, 6);
assert q contains "()()";
```

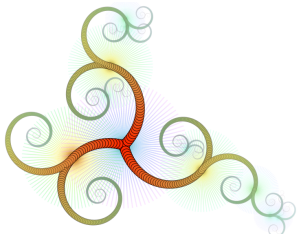
Constraints



"Find a 4-character string v, such that:

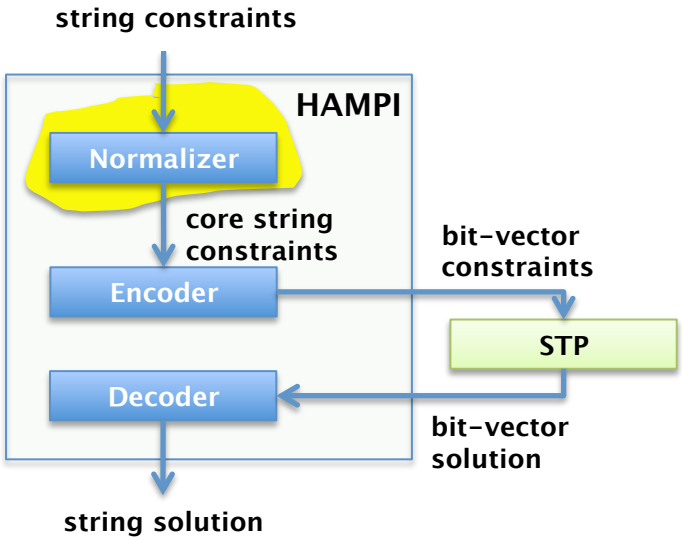
- **(v) has balanced parentheses, and**
- **(v) contains substring ()()"**

HAMPI finds satisfying assignment v =)()(



HAMPI Normalizer

Core string constraint have only regular expressions



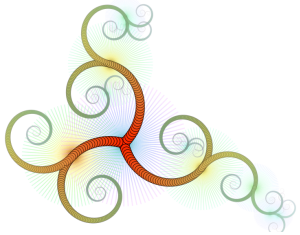
Expand grammars to regexps

- **expand nonterminals**
- **eliminate inconsistencies**
- **enumerate choices exhaustively**
- **sub-expression sharing**

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

bound(*E*, 6) ➔

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

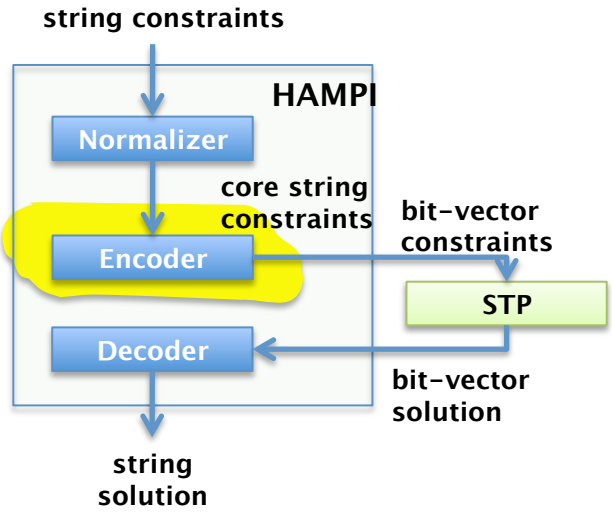


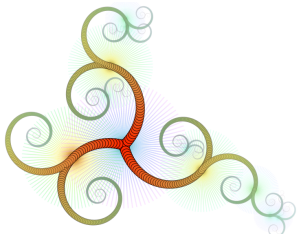
Bit Vectors Are Ordered, Fixed-Size, Sets Of Bits

Bit vector B (length 6 bits)

0	1	0	1	0	1

$$(B[0:1] = B[2:3]) \wedge (B[1:3] = 101)$$





HAMPI Encodes Input As Bit-Vectors

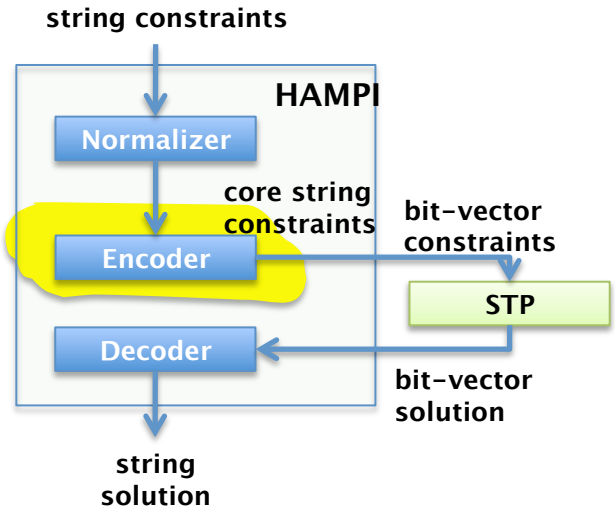
Map alphabet Σ to bit-vector constants:

(→ 0

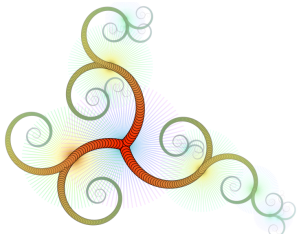
) → 1

Compute size of bit-vector B:

(1+4+1) * 1 bit = 6 bits



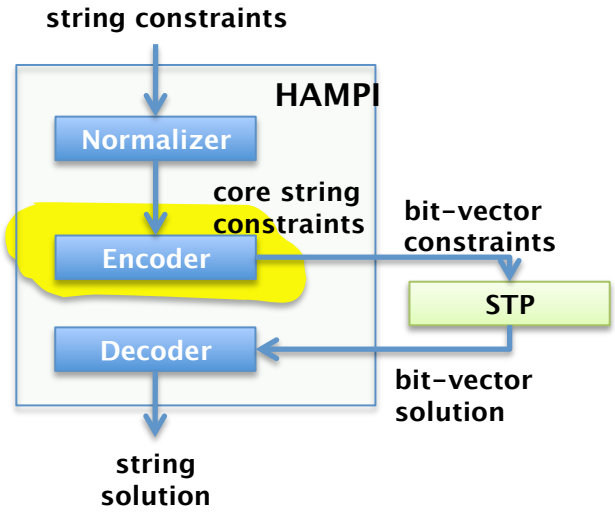
(v) ∈ ()[()() + (())] + [()() + (())]() + ([()() + (())])



HAMPI Encodes Regular Expressions Recursively

Encode regular expressions recursively

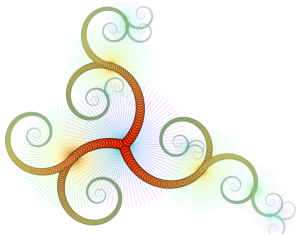
- union + → disjunction ∨
- concatenation → conjunction ∧
- Kleene star * → conjunction ∧
- constant → bit-vector constant



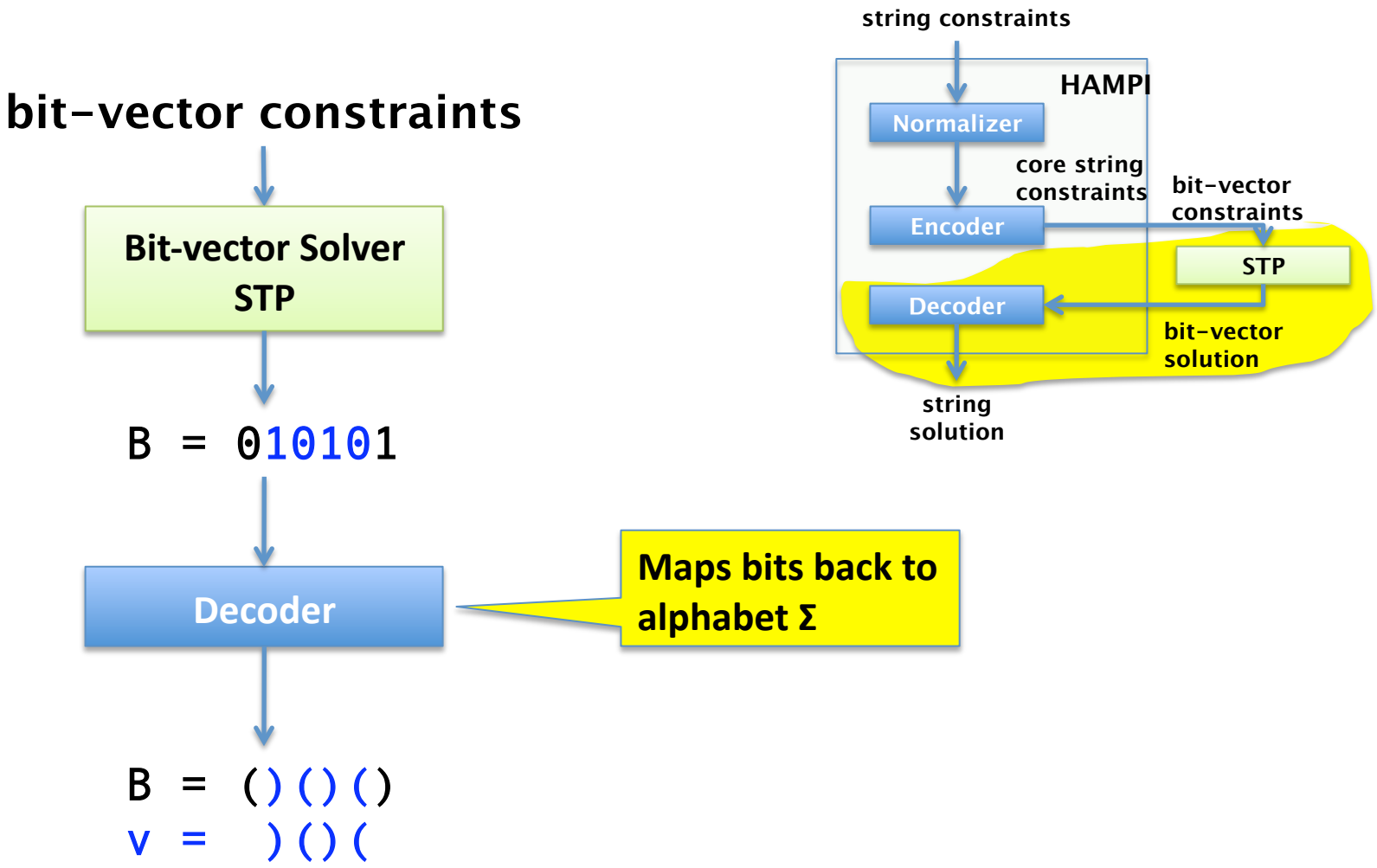
(v) ∈ ()[()() + (())] + [()() + (())]() + ([()() + (())])

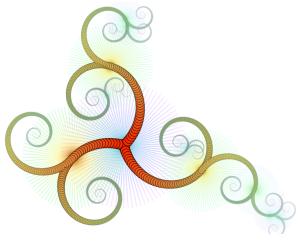
Formula Φ_1 ∨ Formula Φ_2 ∨ Formula Φ_3

B[0]=0∧B[1]=1∧ {B[2]=0∧B[3]=1∧B[4]=0∧B[5]=1 ∨…



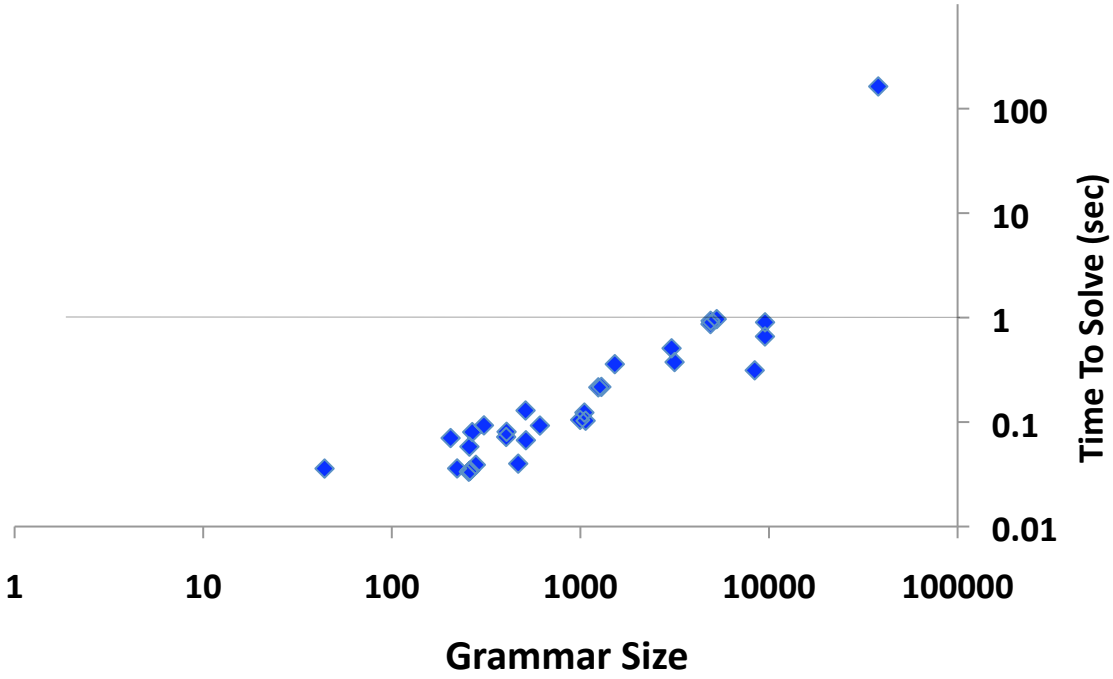
HAMPI Uses STP Solver And Decodes Solution





Result 1: HAMPI Is Effective In Static SQL Injection Analysis

1367 string constraints from [Wassermann PLDI'07]



HAMPI scales to large grammars

HAMPI solved 99.7% of constraints in < 1 sec per constraint

All solvable constraints had short solutions N≤4

Result 2: HAMPI helps Ardilla Find New Vulnerabilities (Dynamic Analysis)

60 attacks on 5 PHP applications (300K+ LOC)

- 23 SQL injection
- 29 XSS

4 cases of data corruption
19 cases of information leak

216 HAMPI constraints solved

- 46% of constraints in < 1 second per constraint
- 100% of constraints in < 10 seconds per constraint

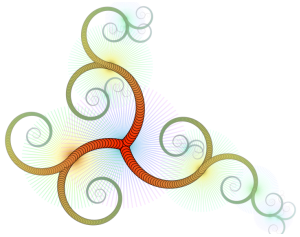
Result 3: HAMPI helps Klee Concolic Tester Find New Bugs

- **Problem: For programs with highly structured inputs, concolic testers can spend too long in the parser**
- **The reason: We may not know which part of input to mark symbolic, and hence mark too much**
- **It is better to generate valid highly structured inputs**
- **Penetrate deep into the program's semantic core**

Result 3: HAMPI helps Klee Concolic Tester Find New Bugs

Program Name	Marking Input Symbolic Klee style (imperative) legal /total inputs Generated (1 hour)	Marking Input Symbolic HAMPI-2-Klee style (declarative) legal /total inputs generated (1 hour)
Cueconvert (music format converter)	0/14	146/146
Logictree (SAT solver)	70/110	98/98
Bc (calculator)	2/27	198/198

- **Improved Code Coverage dramatically (from 30 to 50% with 1 hour work)**
- **Found 3 new errors in Logictree**

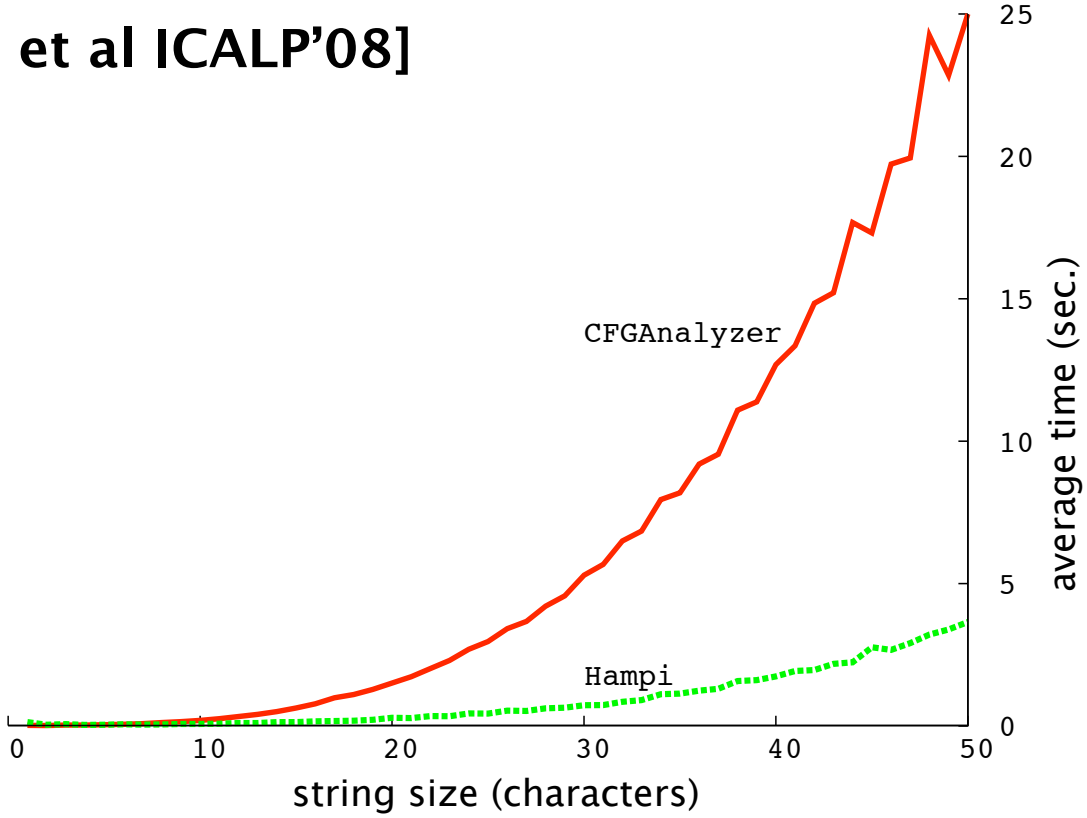


Result 4: HAMPI Is Faster Than The CFGAnalyzer Solver

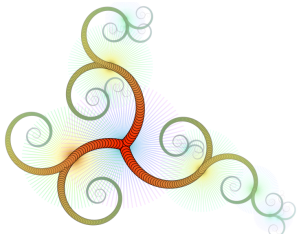
CFGAnalyzer encodes bounded grammar problems in SAT

- **Encodes CYK Parse Table as SAT Problem**

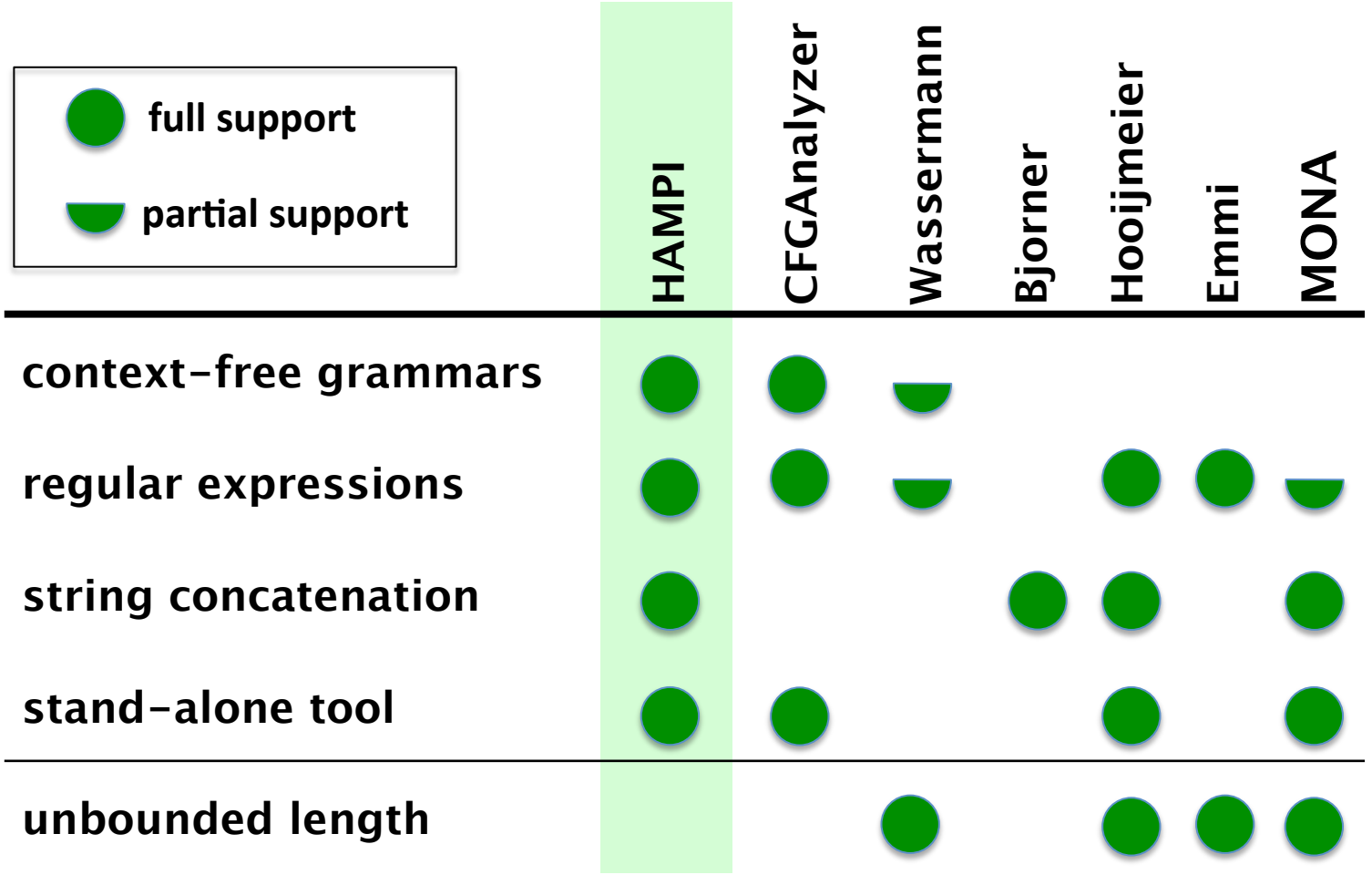
[Axelsson et al ICALP'08]

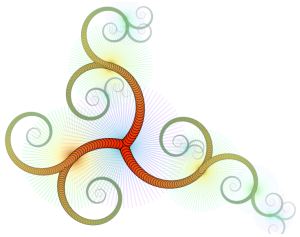


For size 50, HAMPI is 6.8x faster on average (up to 3000x faster)



HAMPI Supports Rich String Constraints





Conclusions

✔ HAMPI: A Novel Solver for String Constraints

✔ Efficient

✔ Rich Input Language

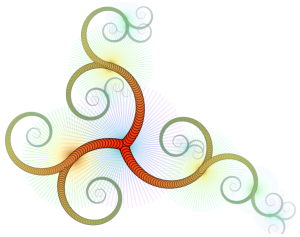
✔ Widely applicable: Formal Methods, Testing, Analysis

✔ Already tested in many real-world apps

✔ Part of well-known infrastructure: e.g., NASA Java PathFinder

✔ Download Source + All Experimental Data

- ✔http://people.csail.mit.edu/akiezun/hampi
- ✔http://people.csail.mit.edu/vganesh/stp.html



Moral of the Story

✔ USE HAMPI and STP

✔ Download Source + All Experimental Data

- ✔http://people.csail.mit.edu/akiezun/hampi
- ✔http://people.csail.mit.edu/vganesh/stp.html

Analysis for SQL Injection Attacks

```
$my_topicid = $_GET['topicid'];

$sqlstmt = "SELECT msg FROM messages WHERE topicid = '$my_topicid';
$result  = mysql_query($sqlstmt);

WHILE ($row = mysql_fetch_assoc($result)) {
    echo "Message " . $row['msg'];
}
```

- PHP Program as part of a website: http://www.mysite.com/?topicid=1
- Access Database with command:
  SELECT msg FROM messages WHERE topicid = 1
- Attacker can reveal the entire database by URL:
  http://www.mysite.com/?topicid=' OR '1'='1

HAMPI Constraints That Create SQL Injection Attacks

user 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]+;
```

bounded SQL grammar

```
reg SqlSmallBounded := bound(SqlSmall, 53);
```

SQL query

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

SQLI attack conditions

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

"q is a valid SQL query"

"q contains an attack tautology"

HAMPI finds an attack input: v → 1' OR '1'='1

SELECT msg FROM messages WHERE topicid=1' OR '1'='1'