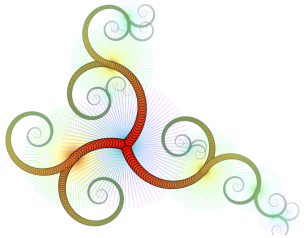
Massachusetts Institute of Technology

HAMPI
A Solver for String Theories

Vijay Ganesh
MIT

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

Dagstuhl, 2010



Motivation for String Theories

String-manipulating programs

- ✓ String in, String out
- ✓ Web applications in PHP, JavaScript
- ✓ Software to Web increasing

Web software particularly vulnerable to security bugs

String reasoning needed

- ✓ Testing
- ✓ Verification
- ✓ Analysis

HAMPI
A Novel String Solver

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

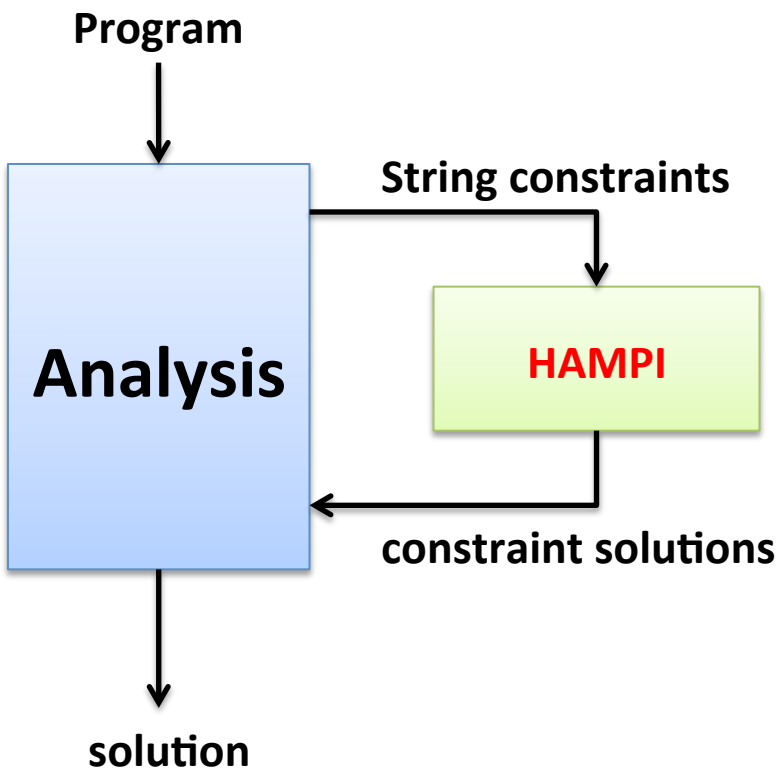
Efficient

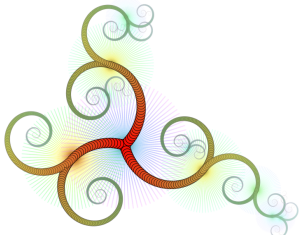
Expressive

Robust and easy to use

Tested on many diverse apps

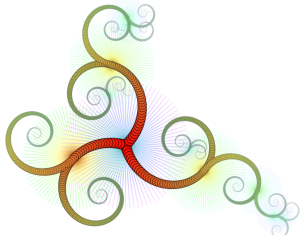
Applied to important and hard problems (ISSTA 2009)





HAMPI Results Summary

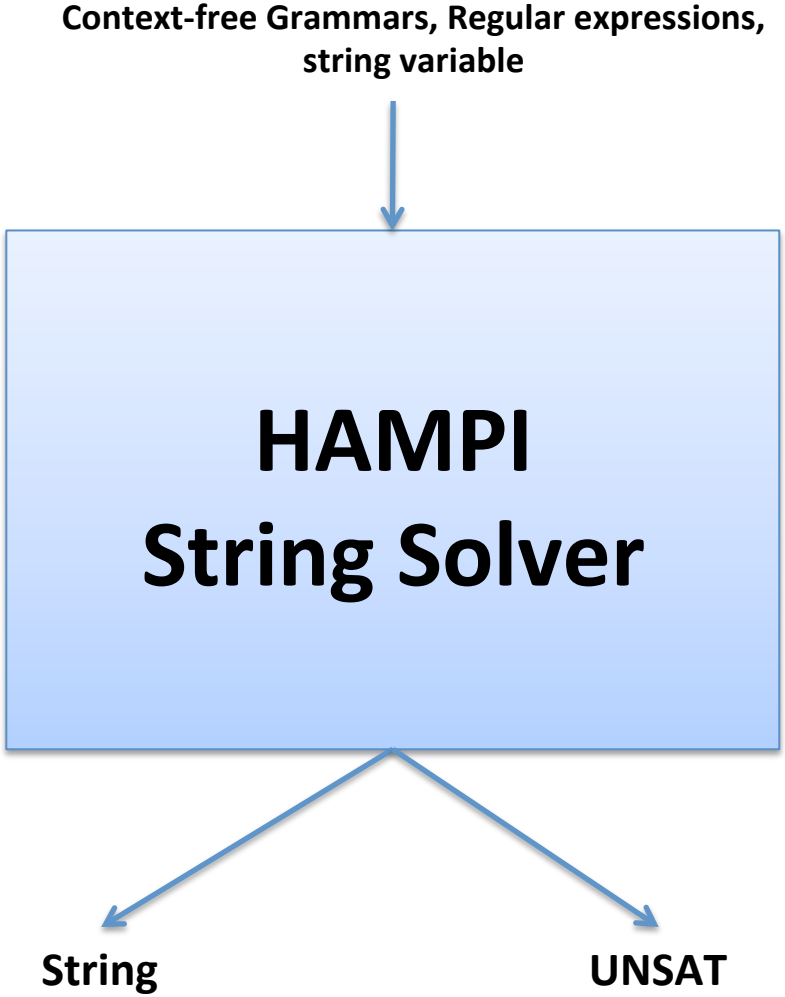
- ✔ SQL injection vulnerability detection using static analysis
- ✔ SQL injection using dynamic analysis (Ardilla tool, ICSE 2009)
- ✔ Structured input generation for Klee (OSDI, 2008)
- ✔ Plugged into NASA Java PathFinder
- ✔ Used to build Kaluza String Solver
  - ✔ Kudzu JavaScript Bugfinder (Oakland, 2010)



HAMPI Solver Input Language

- ✓ Terms
  - ✓ String constants (‘Dagstuhl’)
  - ✓ String variables (Bounded length)
  - ✓ Concatenation (v @ Dagstuhl)
- ✓ Atomic Formulas
  - ✓ Term in Context-Free Grammars (Yacc)
  - ✓ Term in Regular Expressions
  - ✓ Term contains substring
- ✓ Conjunction of literals

The String SAT Problem

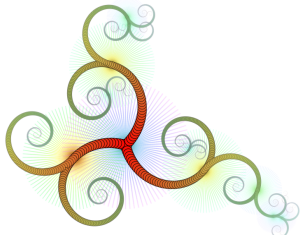


- Emptiness problem for an intersection of Context-free Grammars

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

  where
  - s is bounded
  - s contains some substring

- Different from string matching



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 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"

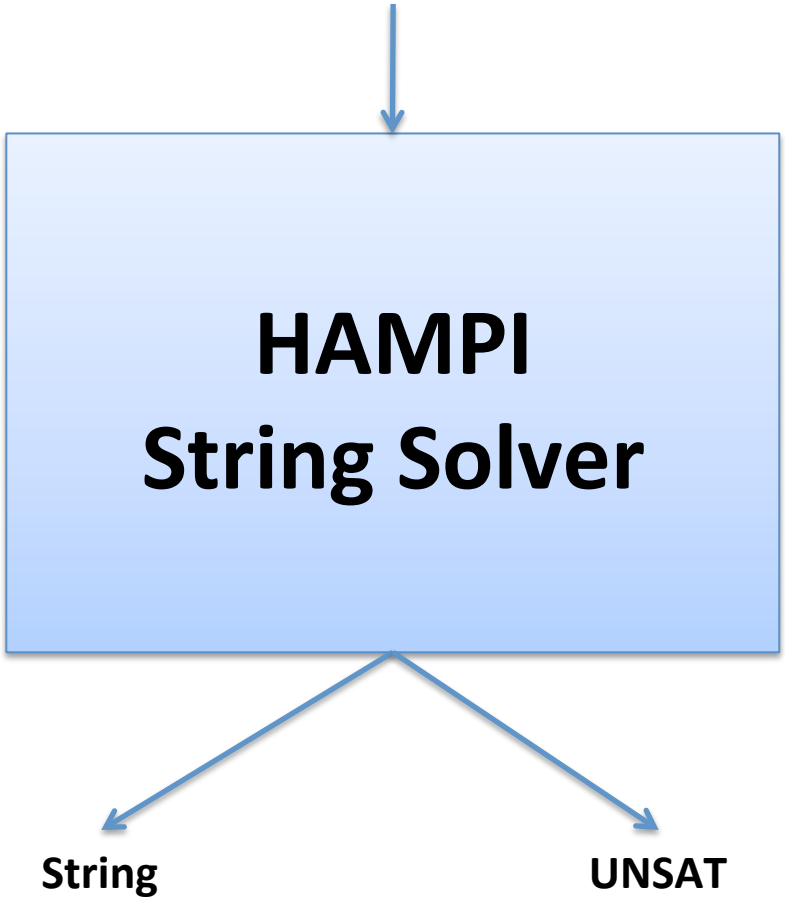
"q contains an attack tautology"

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

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

A Problem Instance

Context-free Grammars, Regular expressions, string variable



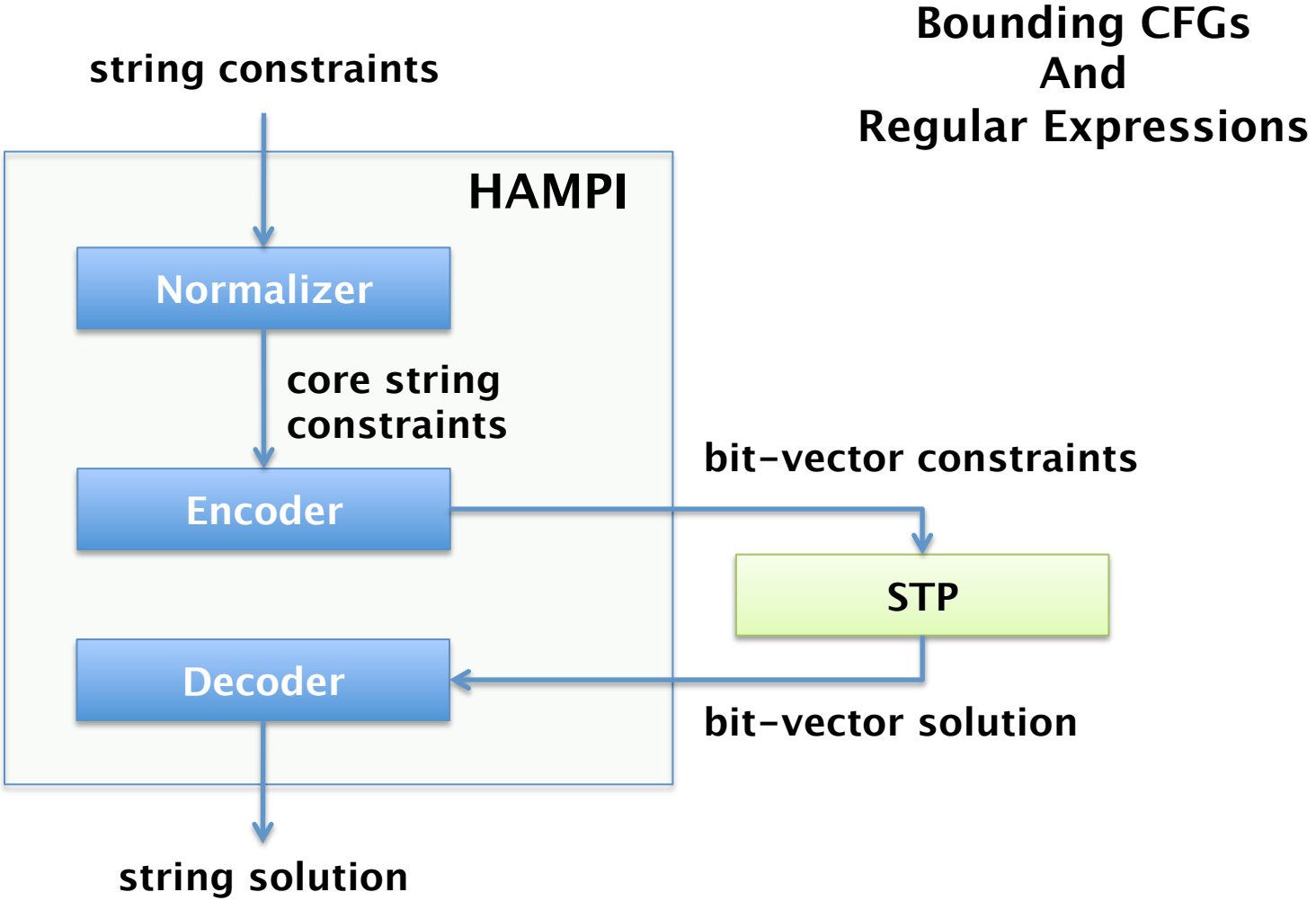

```
var v:4;

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

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

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

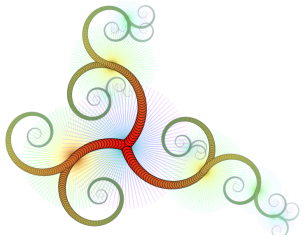
"Find a 4-character string v, such that:
- (v) has balanced parentheses, and
- (v) contains substring ()()"

HAMPI finds satisfying assignment
v =)()(

Hence, q = ()()()

HAMPI Internals





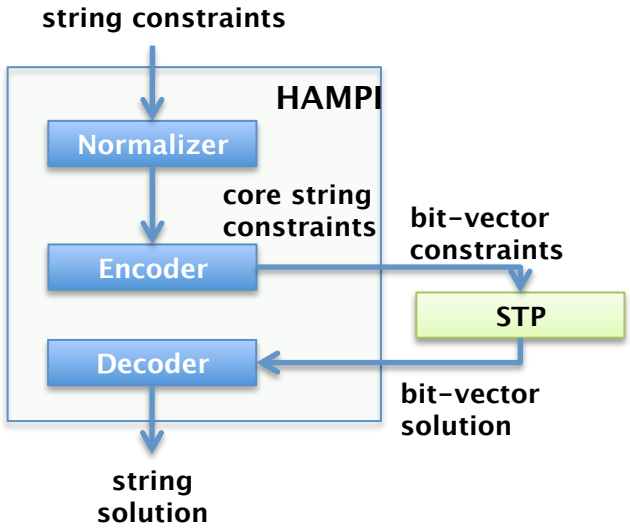
Re-introducing STP

- ✔ SMT solver for bit-vectors and arrays (CAV, 2007)
- ✔ Enabled Dynamic Systematic testing
  - ✔ EXE (CCS, 2006)
  - ✔ Klee (OSDI, 2008)
- ✔ Used in dozens of research projects (25 listed on STP website)
- ✔ Used primarily in
  - ✔ Testing (e.g., Klee and EXE by Dawson Engler)
  - ✔ System Security (e.g., Bitblaze by Dawn Song)
  - ✔ Formal Verification (e.g., ACL2 plugin by Smith & Dill, NVIDIA)
  - ✔ Cryptography (e.g., Inverting hashes by Dan Kaminsky)

HAMPI Example

```
var v:4;

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

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

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

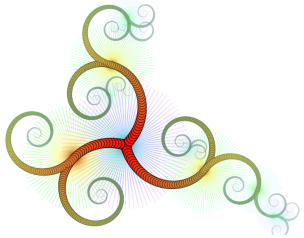
Constraints



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

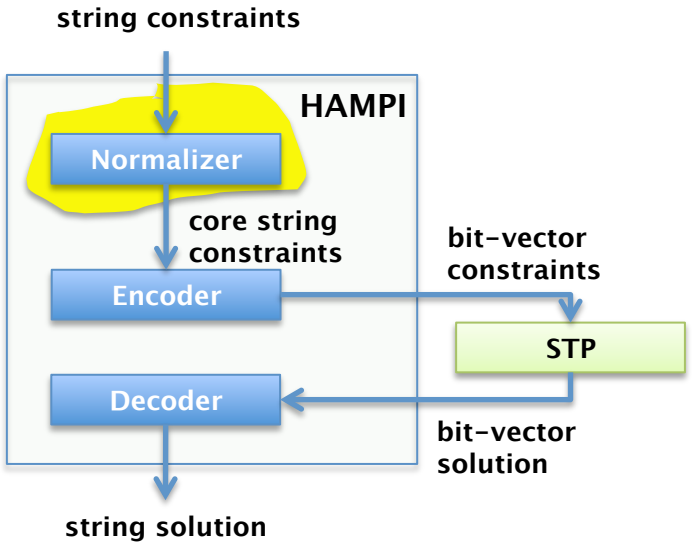
- (v) has balanced parentheses, and
- (v) contains substring ()()"

HAMPI finds satisfying assignment v =)()(



HAMPI Normalizer

Core string constraint have only regular expressions

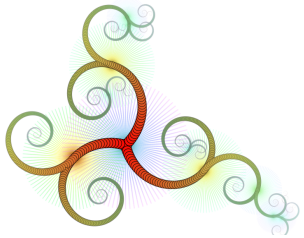


Expand grammars to regexps

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

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

```
                    ([()() + (())]) +
bound(E, 6) ➔ ()[()() + (())]   +
                    [()() + (())]()
```

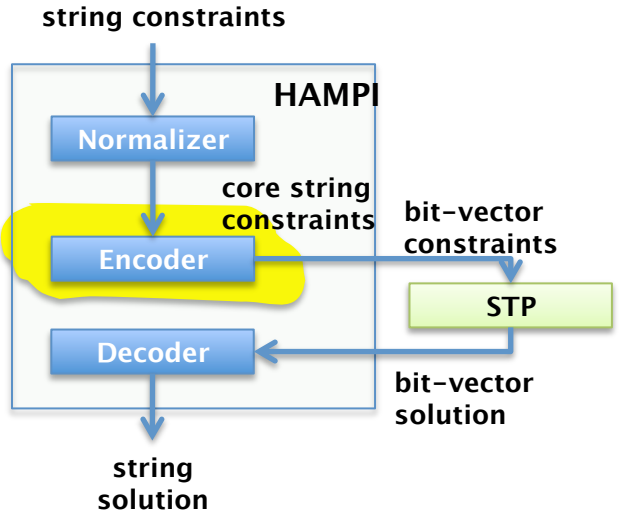


Bit Vectors Are Ordered, Fixed-Size, Sets Of Bits

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

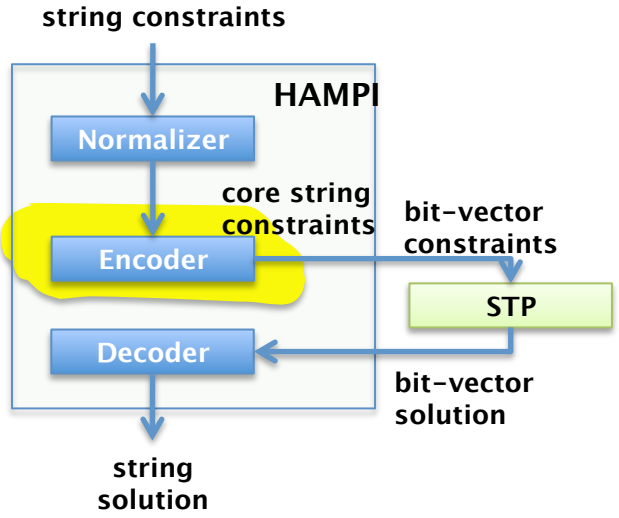
Map alphabet Σ to bit-vector constants:

(→ 0

) → 1

Compute size of bit-vector B:

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

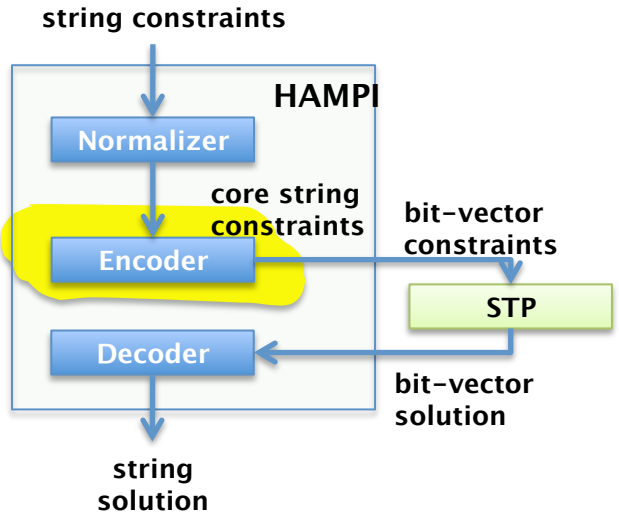


(v) ∈ ()[()() + (())] + [()() + (())]() + ([()() + (())])

HAMPI Encodes Regular Expressions Recursively

Encode regular expressions recursively

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

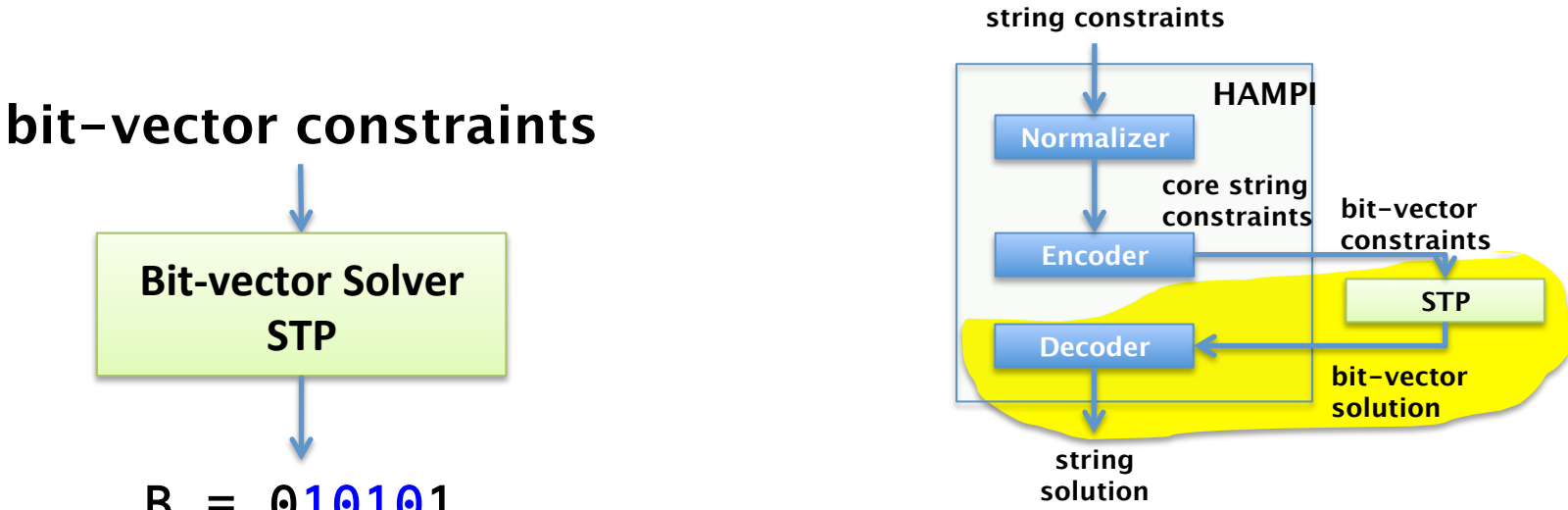


$$(v) \in ()[()() + (())] + [()() + (())]() + ([()() + (())])$$

Formula Φ_1 ∨ Formula Φ_2 ∨ Formula Φ_3

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

HAMPI Uses STP Solver And Decodes Solution



bit-vector constraints

Bit-vector Solver STP

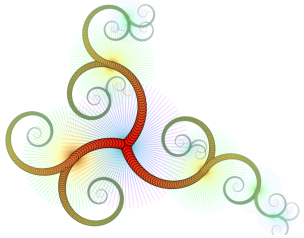
B = 010101

Decoder

Maps bits back to alphabet Σ

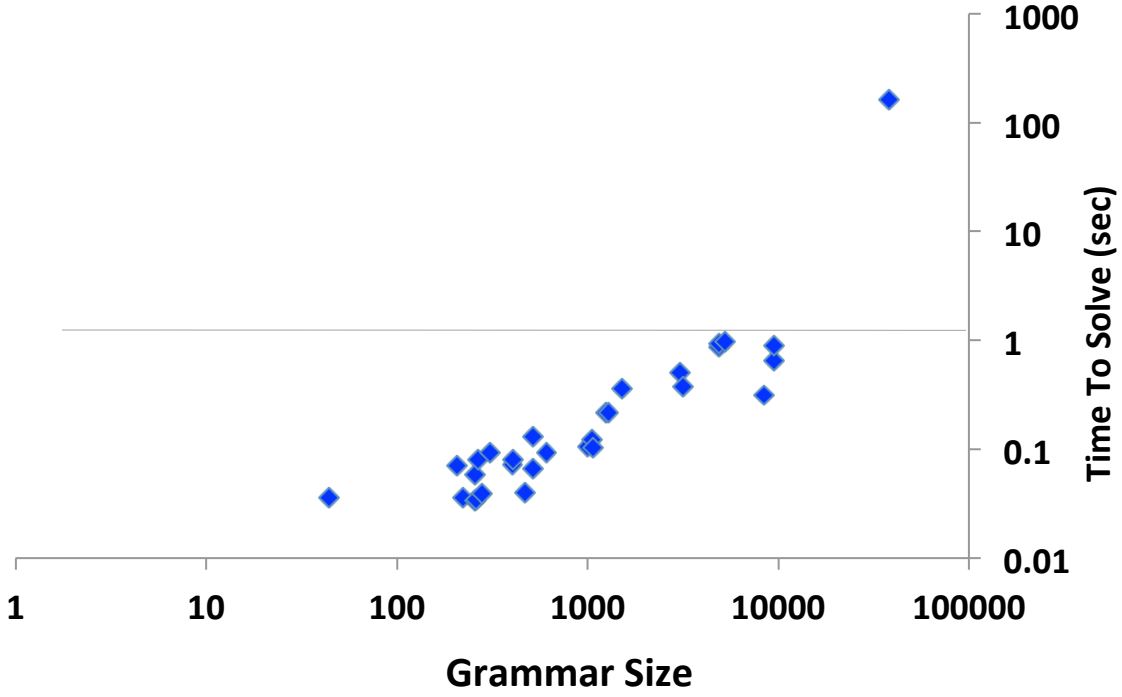
B = ()()()

v =)()(



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

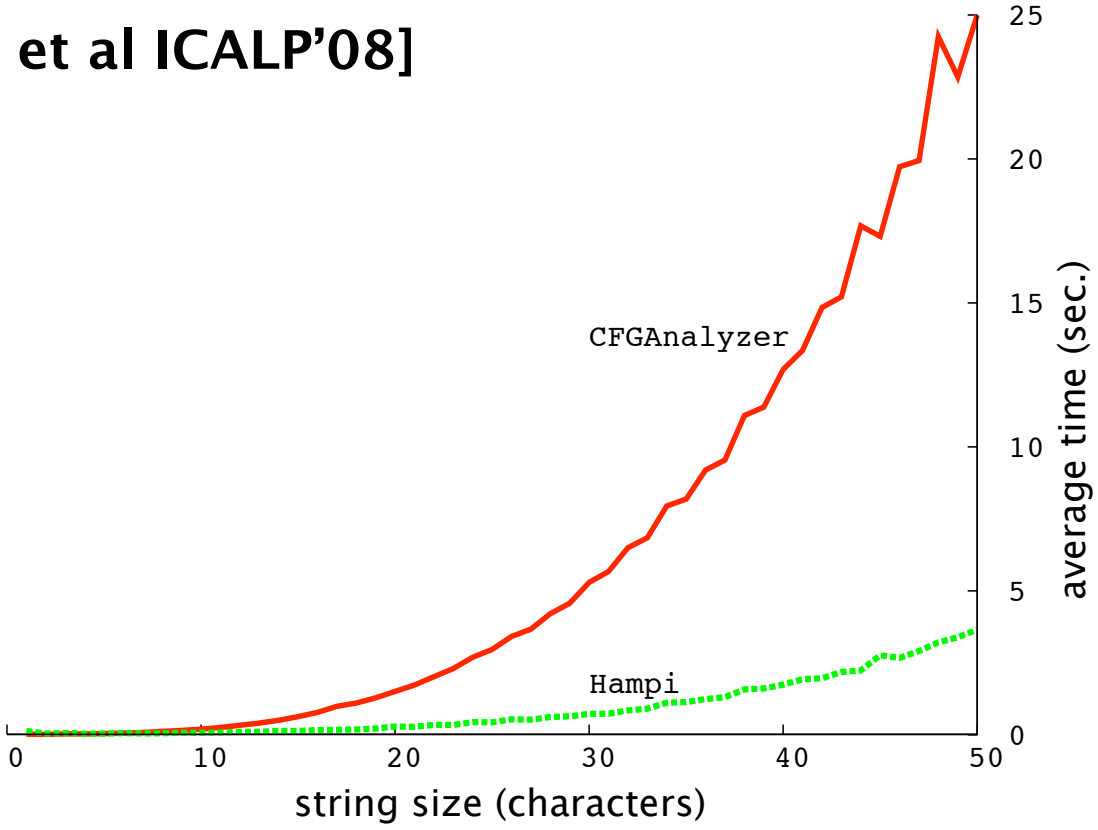
- **Improved Code Coverage dramatically (from 30 to 50% with 1 hour work)**
- **Found 3 new errors in Logictree**

Result 4: HAMPI Is Faster Than The CFGAnalyzer Solver

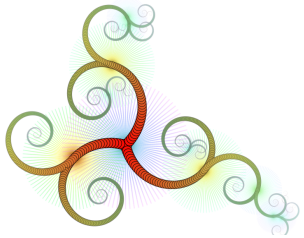
CFGAnalyzer encodes bounded grammar problems in SAT

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

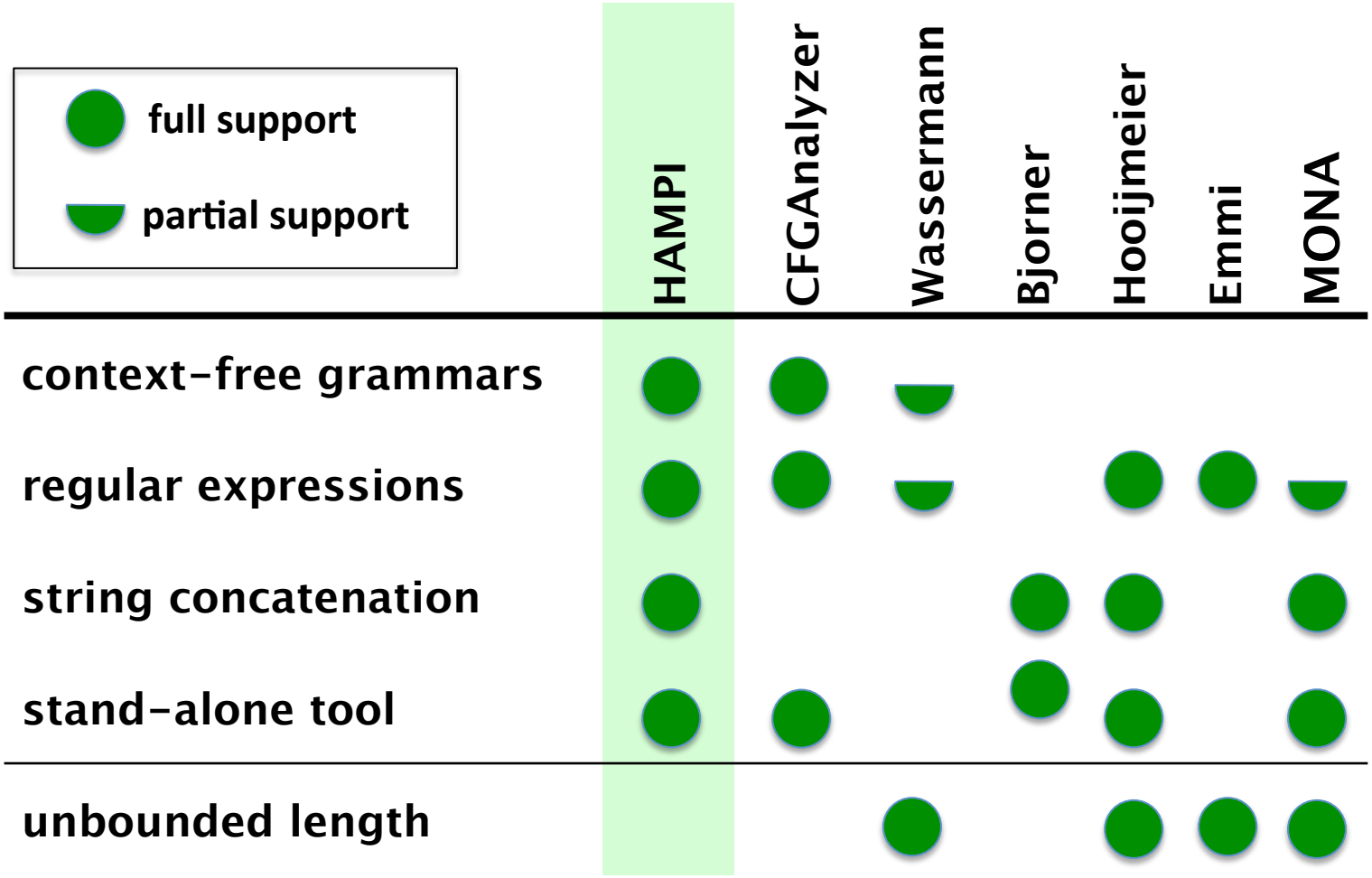
[Axelsson et al ICALP'08]

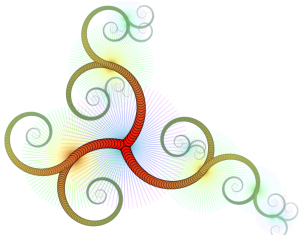


For size 50, HAMPI is 6.8x faster on average (up to 100x faster)



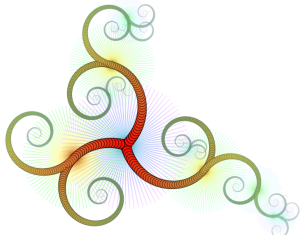
HAMPI Supports Rich String Constraints





HAMPI vs. MSR String Solver

- ✓ HAMPI
  - ✓ Context-free grammars
  - ✓ Regular Expressions
  - ✓ Explicit bounding
- ✓ MSR String Solver (Bjorner's et al.)
  - ✓ Length function
  - ✓ Substring
  - ✓ Word Equations
- ✓ Addressed in Kaluza



Conclusions

✔ HAMPI: A Novel Solver for String Constraints

✔ Efficient

✔ Rich Input Language

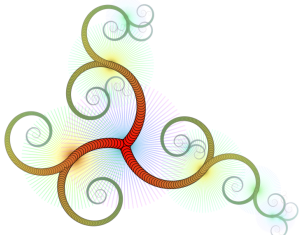
✔ Widely applicable: Formal Methods, Testing, Analysis

✔ Already tested in many real-world apps

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

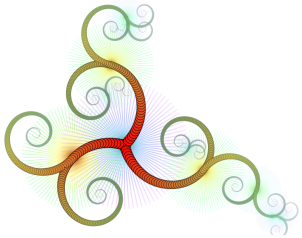
✔ Download Source + All Experimental Data

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



HAMPI Results Summary

- ✔ SQL injection vulnerability detection using static analysis
  - ✔ 6 PHP apps (339, 750 LOC)
- ✔ SQL injection using dynamic analysis
  - ✔ (Ardilla tool, ICSE 2009)
  - ✔ 60 attacks (23 SQL injection)
  - ✔ 5 PHP applications (14K+ LOC)



HAMPI Results Summary

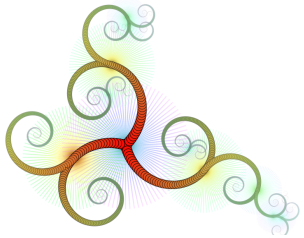
- ✔ Automatic generation of structured inputs
  - ✔ Klee tester
  - ✔ Skip parsing
  - ✔ Exercise deep code
  - ✔ Dramatically improved code coverage and bug finding
- ✔ Efficient: an order of magnitude faster on-average
  - ✔ Often 100s of times faster than CFGAnalyzer

Massachusetts Institute of Technology

Kaluza
A Solver for Word Equations and Regular Expressions

Jointly with

(With Prateek Saxena, Devdatta Akhawe, Adam Kiezun, Stephen McCamant, Dawn Song)

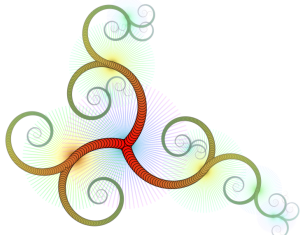


Kaluza Solver Input Language

- Terms
  - String constants (‘Dagstuhl’)
  - Multiple String variables (Bounded length)
  - Concatenation (v @ Dagstuhl)
  - Length(term)

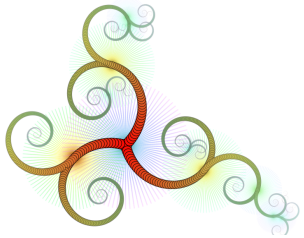
- Atomic Formulas
  - Term in Regular Expressions
  - Term contains substring
  - Term = Term
  - Length(term) <= n

- Boolean combination of formulas



Kaluza Solver

- Built using HAMPI and STP
- Used for JavaScript bugfinding
- Found many bugs in iGoogleGadget and FacebookConnect
- More than 50,000+ tests available on Kaluza website
- SMTization of String theories needed



STP, HAMPI, Kaluza

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

(SMT Solvers and Fuzzers)