MegoSAT

A concurrent SAT solver in Go

Today: “Search for directions”

Lukas Prokop
Advisor: Florian Mendel
Institute of Applied Information Processing and Communications

Outline

- Goals
- Go-based concurrency
- Parallelized SAT solvers
- Application: differential cryptanalysis

MegoSAT

- concurrent

MegoSAT

- concurrent
- SAT solver

MegoSAT

- concurrent
- SAT solver
- in Go

MegoSAT

- concurrent
- SAT solver
- in Go
- for differential cryptanalysis

Core business

- My bachelor thesis: "Using SAT Solvers to detect Contradictions in Differential Characteristics"
- We encode hash algorithms like MD4.
- We induce a hash difference at the output.
- Which preimages yield collisions?
- Let a SAT solver determine that!

Concurrency

```
1   package main
2
3   import ”fmt”
4
5   func main() {
6       go compute(0, 100)
7       fmt.Println(”Continue”)
8
9       var input string
10      fmt.Scanln(&input)
11  }
```

```
12  func compute(min, max int) {
13      sum, base := 0, min
14      for ; base <= max; base++ {
15          sum += base
16      }
17      fmt.Printf(”Sum is %d\n”, sum)
18  }
```

Concurrency

```
1   package main
2
3   import ”fmt”
4
5   func main() {
6       go compute(0, 100)
7       fmt.Println(”Continue”)
8
9       var input string
10      fmt.Scanln(&input)
11  }
```

wait

```
12  func compute(min, max int) {
13      sum, base := 0, min
14      for ; base <= max; base++ {
15          sum += base
16      }
17      fmt.Printf(”Sum is %d\n”, sum)
18  }
```

stdout:

Continue
Sum is 5050

Concurrency

- CSP-inspired concurrency model (Tony Hoare)
- light-weight processes
- go runtime handles scheduling
- GOMAXPROCS=4 to use 4 cores
- ~~share memory to communicate~~
  share memory by communicating
- typed channels

Concurrency

```
1  package main
2  import "fmt"
3
4  func main() {
5      data := make(chan int)
6      done := make(chan bool)
7
8      go worker(1, data, done)
9      go worker(2, data, done)
10
11     data <- 0
12     <-done
13     fmt.Println("Finished")
14 }
```

```
16 func worker(wid int, data chan int,
17                 done chan bool) {
18   for {
19     select {
20       case value := <-data:
21         if value == 10 {
22           done <- true
23         } else {
24           fmt.Printf("%d computed: %d\n",
25             wid, value)
26           data <- value + 1
27         }
28       case <-done:
29         return
30 } } }
```

Concurrency

```
1 computed: 0
2 computed: 1
1 computed: 2
2 computed: 3
1 computed: 4
2 computed: 5
1 computed: 6
2 computed: 7
1 computed: 8
2 computed: 9
Finished
```

```
1  package main
2  import ”fmt”
3
4  func main() {
5      data := make
6      done := mak
7
8      go worker(1, data, done)
9      go worker(2, data, done)
10
11     data <− 0
12     <−done
13     fmt.Println(”Finished”)
14 }
```

```
16 func worker(wid int, data chan int,
17                 done chan bool) {
18   for {
19     select {
20       case value := <−data:
21         if value == 10 {
22           done <− true
23         } else {
24           fmt.Printf(”%d computed: %d\n”,
25             wid, value)
26           data <− value + 1
27         }
28       case <−done:
29         return
30 } } }
```

Concurrency

Basic idea:

- Parallelize work of SAT solver
- Channels distribute
  - assumptions?
  - learned clauses?
  - conflicts?

Parallelized SAT solvers

- pBoolector
- Plingeling
- Treengeling
- PCASSO
- SArTagnan
- PeneLoPe
- MTSS
- ...

Parallelized SAT solvers

Observations:

1. CDCL becomes more popular than DPLL [03] [06]

2. Two major approaches
   - Divide-and-conquer strategy for search space
   - Cooperative portfolio approach

Parallelized SAT solvers

Divide and conquer [02] [04]

- Partition the formula to divide the total workload evenly over multiple SAT solver instances
- Probably a preprocessor (doing “inprocessing”, like SatELite) can be helpful for us to simplify formula and then distribute work?
- Problem: Splitting search path is not as easy as it sounds.

Parallelized SAT solvers

Cooperative portfolio approach [05]

- Several SAT solver instances with different configurations/heuristics work on the *same* formula
- Fastest one returns result
- Problem: Finding a good set of configs, bad implementations of concurrency (congestion, locking, …)

Parallelized SAT solvers

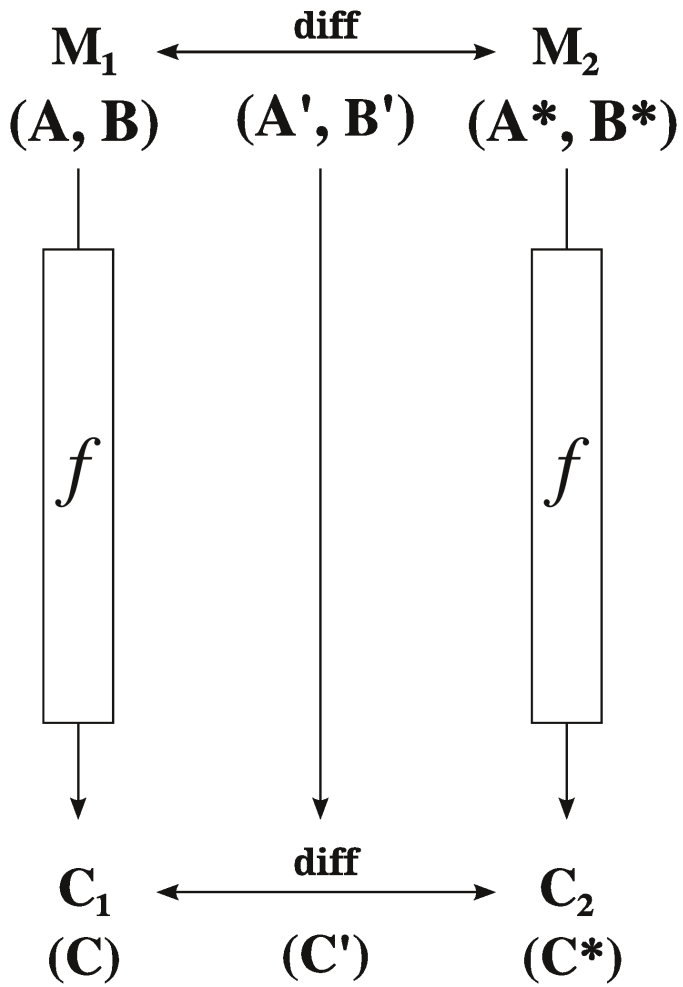
Both approaches can be *extended* by [03]

- conflict clause sharing
- parallel unit performs conflict clause analysis
- unit evaluates metrics of the formula, starts parallel units for search/conflict analysis

Problems: Don't share too much information (experiments yield < 40 literals).

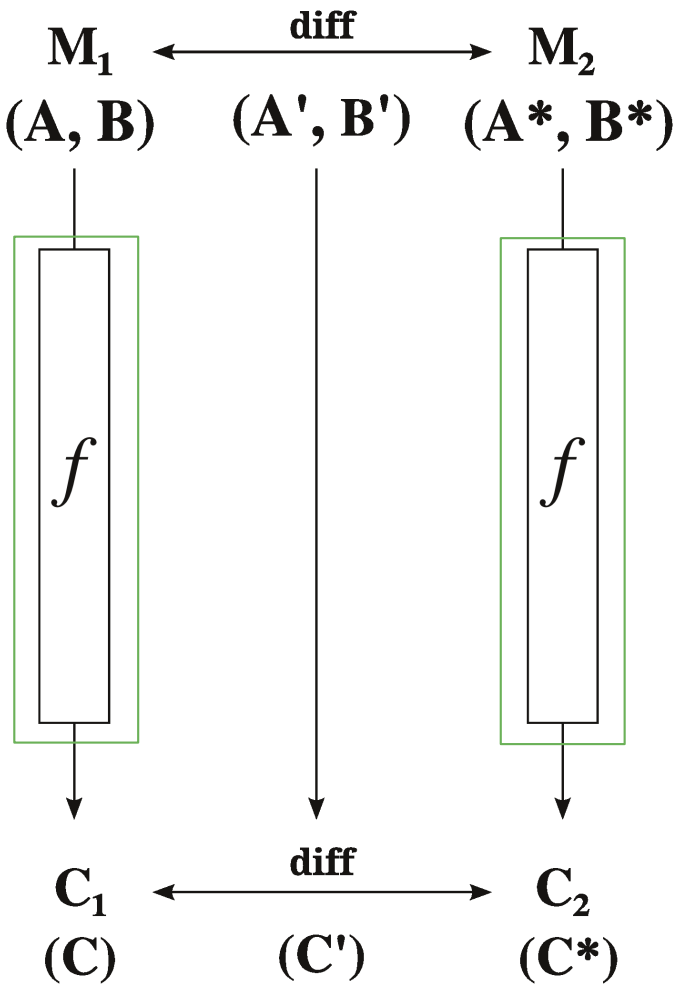
This corresponds to my idea to use channels. Not sure which approach to take.

Differential cryptanalysis

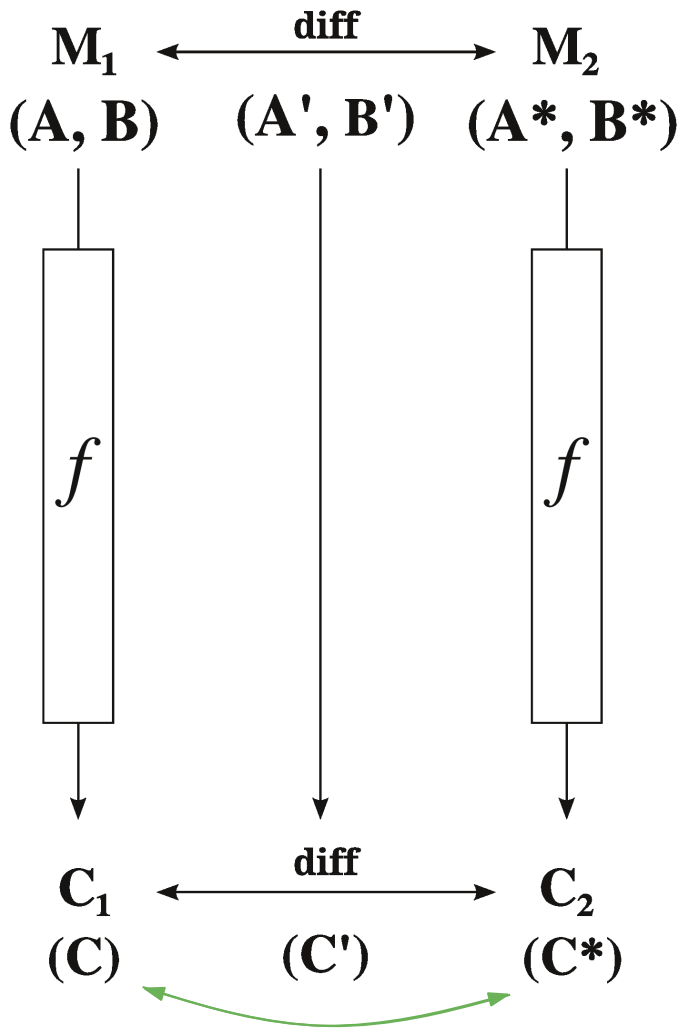


Differential cryptanalysis

we have a boolean (?) description of the hash algorithm



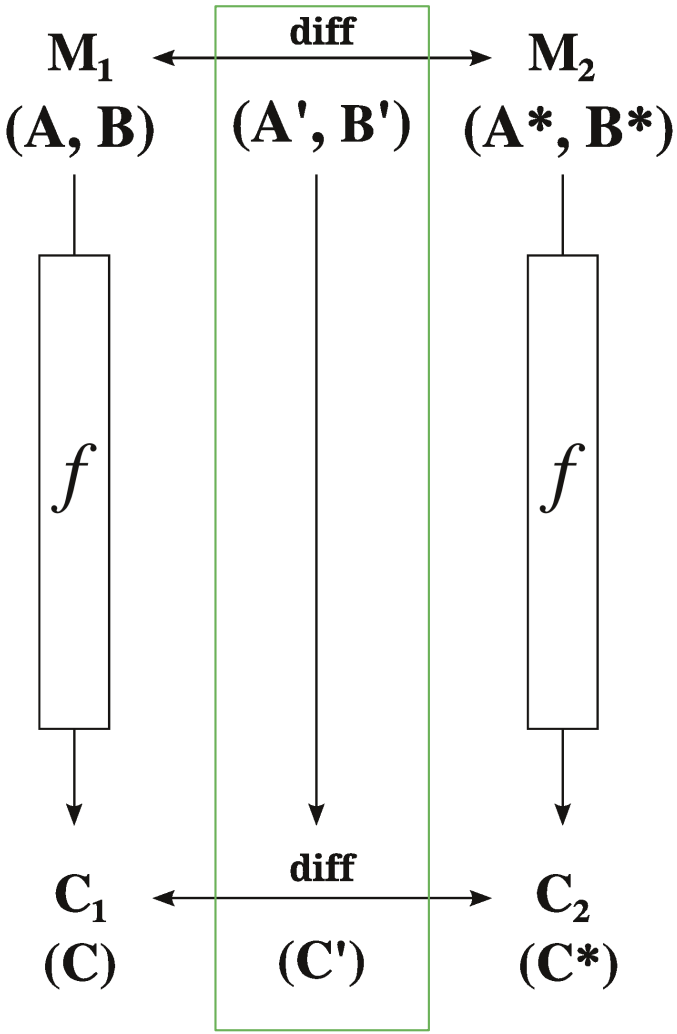
Differential cryptanalysis



we induce a difference between message using assumptions

Differential cryptanalysis

we hopefully retrieve information about the differential when analysing conflicts



Example of my bachelor thesis

Add operation

A:	0	0	1	1
B:	0	1	0	1
S:	1	0	0	0

Example of my bachelor thesis

Add operation

A:	0	0	1	1
B:	0	1	0	1
S:	1	0	0	0

SATISFIABLE

Example of my bachelor thesis

Add operation

A:	0	0	1	1
B:	0	1	0	1
S:	1	0	1	0

Example of my bachelor thesis

Add operation

A:	0	0	1	1
B:	0	1	0	1
S:	1	0	1	0

UNSATISFIABLE

Example of my bachelor thesis

Add operation

Δ A:	–	–	–	x
Δ B:	–	–	–	x
Δ S:	?	?	?	-

SATISFIABLE

– ⇒ bits in both instances are equivalent
x ⇒ bits in both instances are different
? ⇒ bits in both instances are unknown

Differential cryptanalysis

Representation of crypto algorithms?

- SAT? some SMT like QF_NIA? bitvectors?
- STP like Robert Primas? [01]

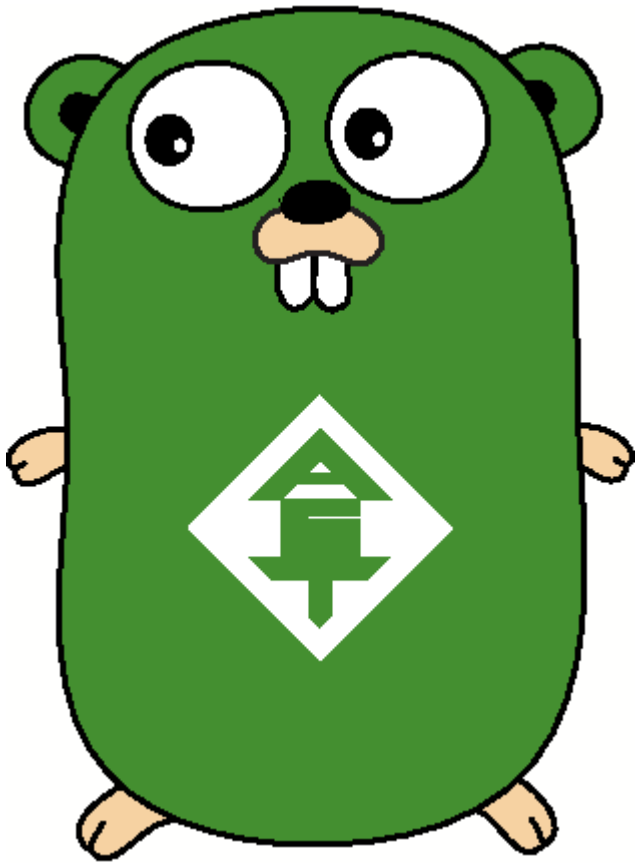
Encoding of the problem?

- How do we make SAT solver recognize that two variables represent the same bit? And then deduce conflicts correspondingly?

Thanks

- Thanks for any feedback!
- Thank you for your attention!
- Feel free to ask/discuss

http://lukas-prokop.at/proj/megosat



Thanks

01 Robert Primas: “SAT Solver Attacks on Hash Functions” (2014)

02 Armin Biere: “Lingeling, Plingeling and Treengeling Entering the SAT Competition 2013” (2013)

03 Siert Wieringa, Keijo Heljanko: “Concurrent clause strengthening” (2013)

04 Ahmed Irfan, Davide Lanti, Norbert Manthey: “PCASSO – a Parallel CooperAtive Sat SOlver” (2013)

05 Christoph M. Wintersteiger, Youssef Hamadi, Leonardo de Moura: “A Concurrent Portfolio Approach to SMT Solving” (2009)

06 Joao Marques-Silva, Ines Lynce, Sharad Malik: “Conflict-Driven Clause Learning SAT Solvers” (2008)