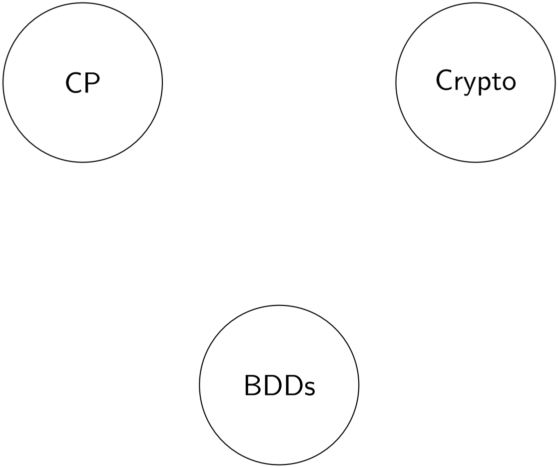
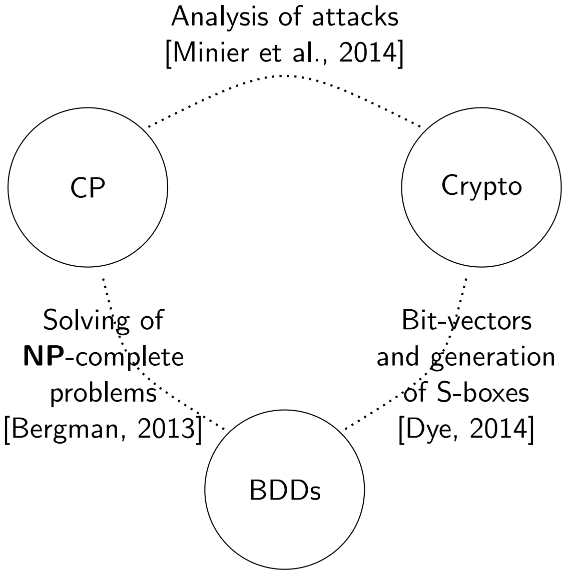
The BDD Domain and Applications to Constraint Satisfaction

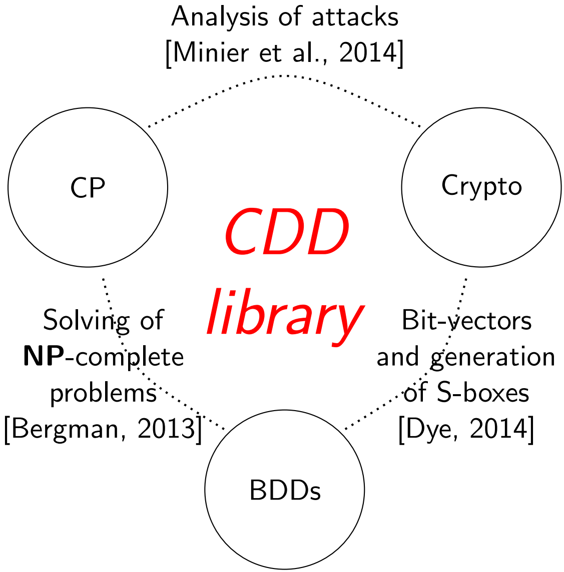
Mathieu Vavrille

ENS de Lyon
M2 internship supervised by Charlotte Truchet
in the University of Nantes (LS2N, TASC team)

July 03, 2019


CP
Crypto
BDDs


Analysis of attacks
[Minier et al., 2014]
CP
Crypto
Solving of
NP-complete
problems
[Bergman, 2013]
Bit-vectors
and generation
of S-boxes
[Dye, 2014]
BDDs


Analysis of attacks
[Minier et al., 2014]
CP
Crypto
CDD library
Solving of
NP-complete
problems
[Bergman, 2013]
Bit-vectors
and generation
of S-boxes
[Dye, 2014]
BDDs

Constraint programming

- Define a problem declaratively with variables and constraints, and find a solution.
- A field between Artificial Intelligence and Operations Research.
- Many applications on solving NP-hard problems in planning, logistics, arts, computational sustainability and, very recently, cryptography.

Constraint satisfaction problem (CSP)

A CSP is the data $(X, \mathcal{D}, C)$ where X is a set of variables, $\mathcal{D}$ is a function from variables to their domains (values they can take), C is a set of constraints defined in:

- intension: high level language to express the constraint, or,
- extension: list of allowed solutions (called table constraints).

Constraint solving

Algorithm: propagate and search

Find the solutions in two steps:

- propagation: delete values from the domains of some variables that are inconsistent with respect to some constraint
- search: enumerate the domains to find the solutions.

Example of propagation:

$$x, y, z \in \{1, 2, 3\}$$
$$x + y \leq z$$

We find that the value $z = 1$ is not consistent (not possible).

Context

A limitation of CP solvers

- Constraint with divisions or modulo
- Constraints on bit-vectors (vectors of bits)

Goal

Investigate the use of another structure on these constraints: BDDs

Existing domains

- BDD [Bryant, 1986]: a data structure initially used for boolean functions
- MDD [Andersen et al., 2007]: the consistency
- bit-vector domain [Michel and Van Hentenryck, 2012]: the computations

Internship contribution [1]

Contribution : CDD library

A library based on a BDD structure to represent domains in a constraint-friendly way,

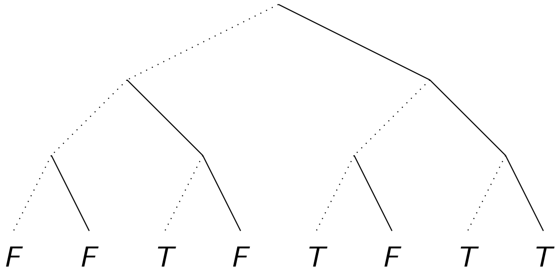
- with to type of BDDs: unrestricted and limited-width,
- implemented in OCaml within the AbSolute constraint solver,
- experimented on a cryptography problem (AES).

[1] `https://github.com/MathieuVavrille/cdd`

```
module type CDD = sig
  type bdd
  val bdd_of : bdd -> bdd -> bdd
  (*** Set operations ***)
  val intersection : bdd -> bdd -> bdd
  (* and others: difference, union, etc *)
  (*** Bitwise operations ***)
  val xor : bdd -> bdd -> bdd
  (* and others: and, or, not, etc *)
  (*** BDD manipulations ***)
  val prefix : bdd -> int -> bdd
  val suffix : bdd -> int -> Bddset.t
  val concat : bdd -> bdd -> bdd
  (*** Solving techniques ***)
  val multiple_mdd_consistency : bdd -> Bddset.t -> bdd
  val refined_consistency : bdd -> bdd -> int -> bdd
  val split: bdd -> (bdd * bdd)
```

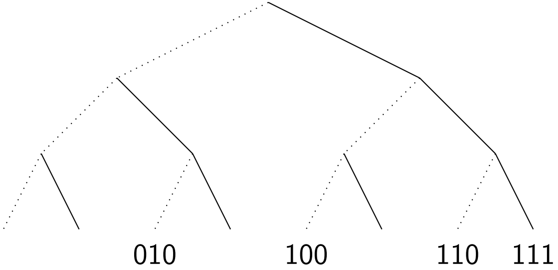
Representation of bit-vectors

- Bit-vectors are represented by the paths from the root to the leaves T.
- The set represented is $\{010, 100, 110, 111\} \Leftrightarrow \{2, 4, 6, 7\}$



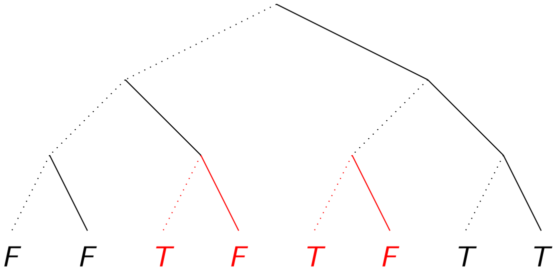
Representation of bit-vectors

- Bit-vectors are represented by the paths from the root to the leaves T.
- The set represented is $\{010, 100, 110, 111\} \Leftrightarrow \{2, 4, 6, 7\}$



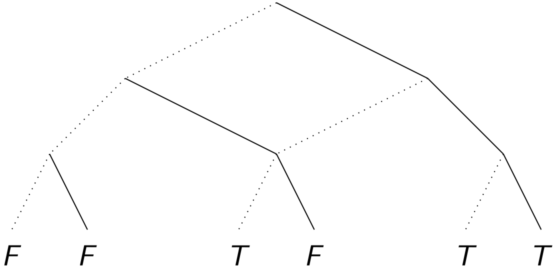
BDD: Reduction of binary tree

We now want a smaller representation



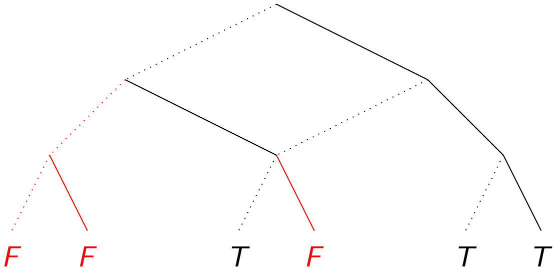
BDD: Reduction of binary tree

We now want a smaller representation



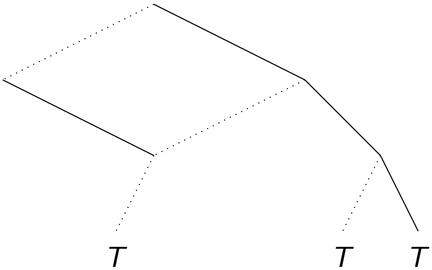
BDD: Reduction of binary tree

We now want a smaller representation



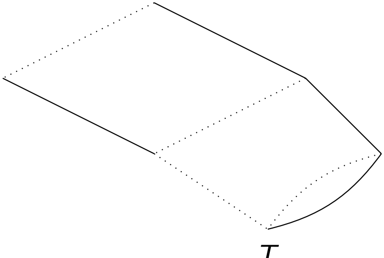
BDD: Reduction of binary tree

We now want a smaller representation



BDD: Reduction of binary tree

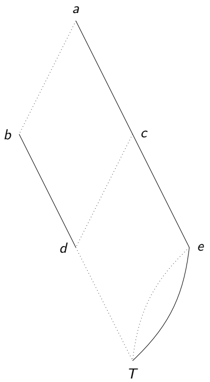
We now want a smaller representation



Implementation

Global hashtable

- keys are (0-child, 1-child), value is the BDD represented
- one entry (F, F) with value F.



name in the tree	ref in the table	key in the table	value in the table
-	h_0	F, F	F
d	h_1	T, F	T
b	h_2	F, h_1	h_1
e	h_3	T, T	T
c	h_4	h_1, h_3	h_1 h_3
a	h_5	h_2, h_4	h_2 h_4

Implementation

Function $\text{BDD\_OF}(a, b)$ **returns** *the BDD* $Node(a, b)$ *where we shared equivalent subtrees*

    if $(a, b) \in global\_hash$ **then**

        return $\text{FIND}(global\_hash, (a, b))$

    else

        $new\_ref \leftarrow Node(a, b)$

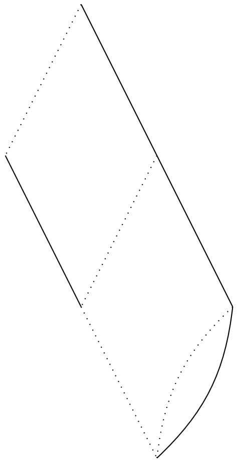
        $\text{ADD\_TO\_HASH}(global\_hash, (a, b), new\_ref)$

        return new_ref

    end

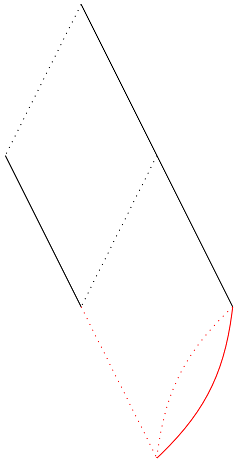
If we always use this algorithm, all the BDDs will be reduced. This is the core of the implementation.

Extraction of prefixes



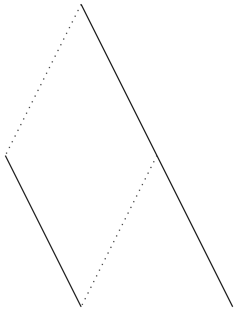
- Prefixes of size k are the first k bits of the BDD B.

Extraction of prefixes



- Prefixes of size k are the first k bits of the BDD B.

Extraction of prefixes



- Prefixes of size k are the first k bits of the BDD B.
- The set $Pref(B,k)$ is $\{01, 10, 11\}$
- We have that $\{2,4,6,7\}//2 = \{1,2,3\}$

General propagation

And constraint

Let the constraint $X \wedge Y = Z$. We compute

$$And^{-1}(\mathcal{D}(Z), \mathcal{D}(X)) = \{y | x \wedge y = z, x \in \mathcal{D}(X), y \in \mathcal{D}(Y), z \in \mathcal{D}(Z)\}$$

Then the propagation on Y is simply

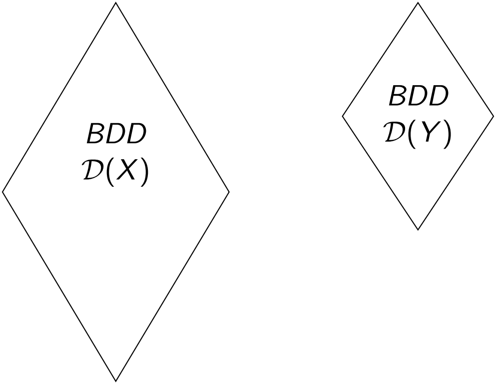
$$\mathcal{D}(Y) \leftarrow \mathcal{D}(Y) \cap And^{-1}(\mathcal{D}(Z), \mathcal{D}(X))$$

What do we need ?

- Bitwise operations
- Intersection

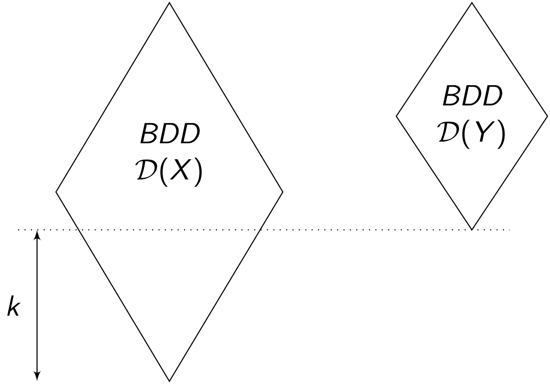
Div constraint: $X//2^k = Y$

Condition on the depth: $depth(X) = k + depth(Y)$
Propagation on Y: $\mathcal{D}(Y) \leftarrow \mathcal{D}(Y) \cap Pref(\mathcal{D}(X), depth(X) - k)$
Propagation on X: $\mathcal{D}(X) \leftarrow \mathcal{D}(X) \cap \mathcal{D}(Y) \cdot \textsc{complete}(k)$



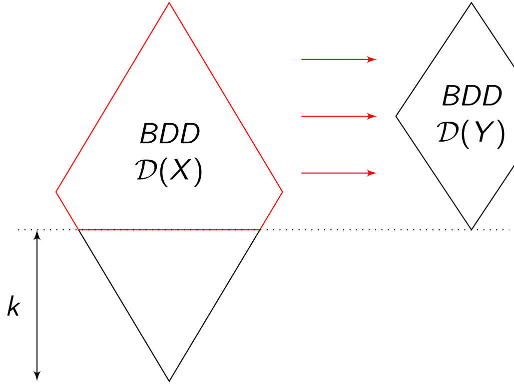
Div constraint: $X//2^k = Y$

Condition on the depth: $depth(X) = k + depth(Y)$
Propagation on Y: $\mathcal{D}(Y) \leftarrow \mathcal{D}(Y) \cap Pref(\mathcal{D}(X), depth(X) - k)$
Propagation on X: $\mathcal{D}(X) \leftarrow \mathcal{D}(X) \cap \mathcal{D}(Y) \cdot \textsc{complete}(k)$



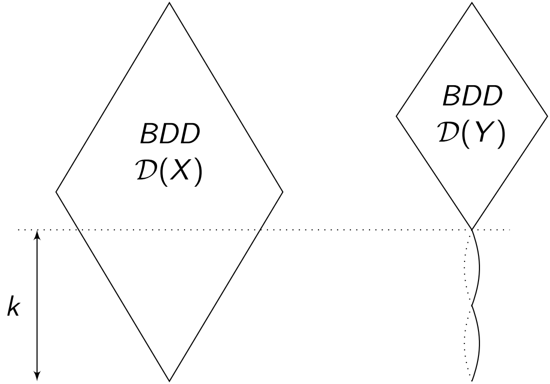
Div constraint: $X//2^k = Y$

Condition on the depth: $depth(X) = k + depth(Y)$
Propagation on Y: $\mathcal{D}(Y) \leftarrow \mathcal{D}(Y) \cap Pref(\mathcal{D}(X), depth(X) - k)$
Propagation on X: $\mathcal{D}(X) \leftarrow \mathcal{D}(X) \cap \mathcal{D}(Y) \cdot \text{COMPLETE}(k)$



Div constraint: $X//2^k = Y$

Condition on the depth: $depth(X) = k + depth(Y)$
Propagation on Y: $\mathcal{D}(Y) \leftarrow \mathcal{D}(Y) \cap Pref(\mathcal{D}(X), depth(X) - k)$
Propagation on X: $\mathcal{D}(X) \leftarrow \mathcal{D}(X) \cap \mathcal{D}(Y) \cdot \textsc{complete}(k)$



Div constraint: $X//2^k = Y$

Condition on the depth: $depth(X) = k + depth(Y)$
Propagation on Y: $\mathcal{D}(Y) \leftarrow \mathcal{D}(Y) \cap Pref(\mathcal{D}(X), depth(X) - k)$
Propagation on X: $\mathcal{D}(X) \leftarrow \mathcal{D}(X) \cap \mathcal{D}(Y) \cdot \textsc{complete}(k)$

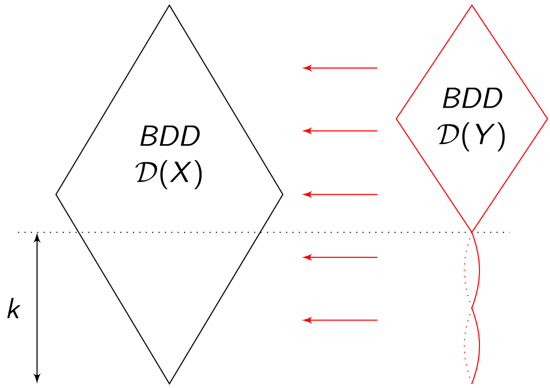


Table constraint

Table constraint

A *table constraint* is a constraint defined on n variables in extension by the set of allowed n-uplet.

Transformation to BDD

Let $\mathcal{T}$ the set representing a table constraint on 3 variables. Let

$$S = \{b^1 \cdot b^2 \cdot b^3 | (b^1, b^2, b^3) \in \mathcal{T}\}$$

S is a set of bit-vectors, thus it can be represented by a BDD.

Table constraint: $Table(X, Y, Z, \mathcal{T})$

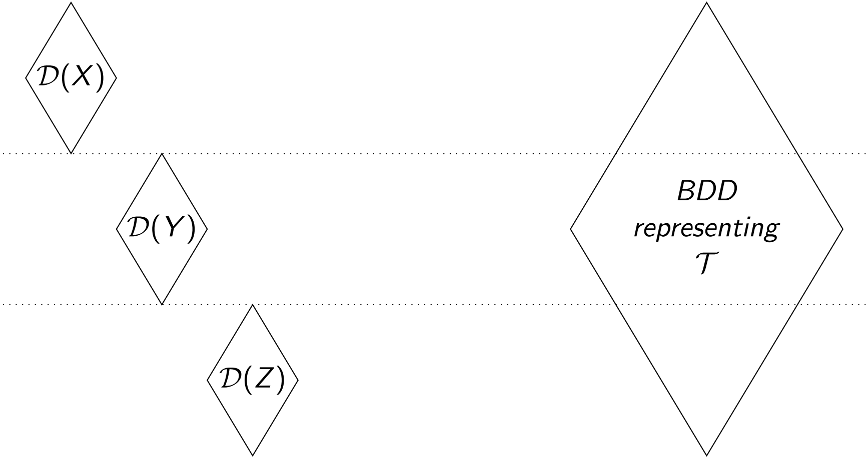


Table constraint: $Table(X, Y, Z, \mathcal{T})$

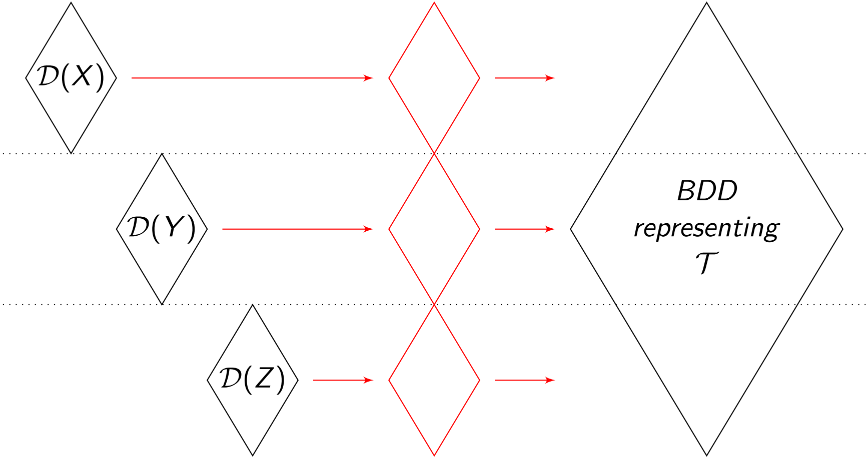


Table constraint: $Table(X, Y, Z, \mathcal{T})$

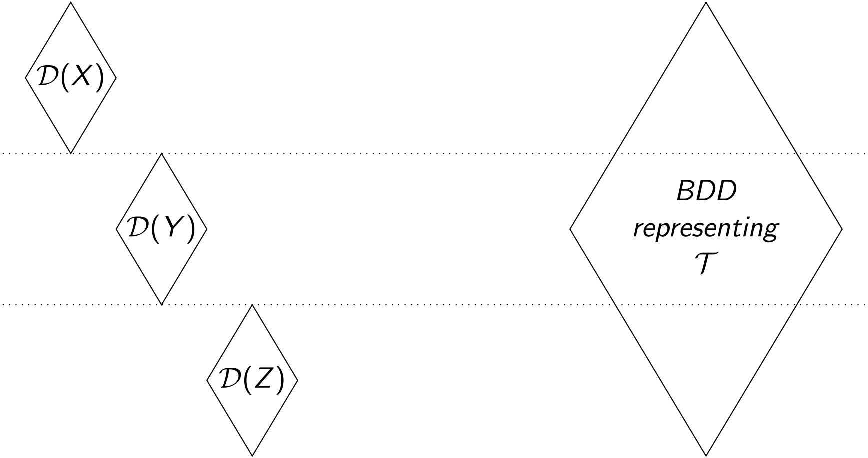
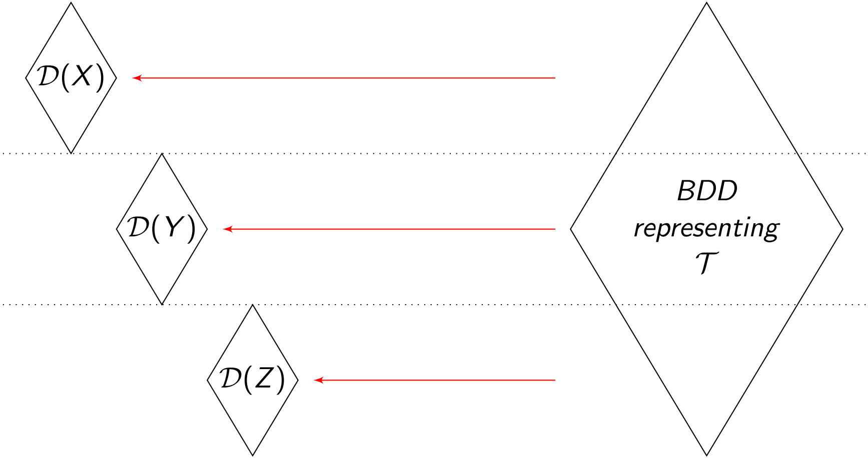
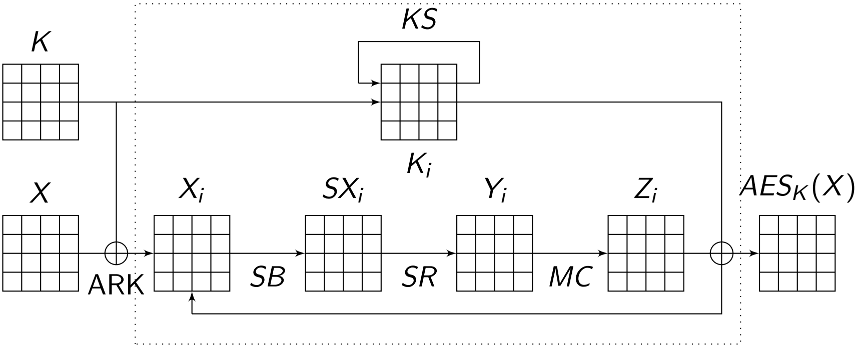


Table constraint: $Table(X, Y, Z, \mathcal{T})$



AES



Differential analysis

Start with messages X, X', and keys K, K'. Compute $E_K(X)$ and $E_{K'}(X')$. What is the probability to get $E_K(X) \oplus E_{K'}(X')$ knowing $X \oplus X'$ and $K \oplus K'$?

Constraint program solving

Translation

- Variables: $\delta X = X \oplus X', \delta K = K \oplus K'$.
- Constraints: representation of the computations (Xor, mix column, S-box, ...)

Representation from [Minier et al., 2014, Gerault et al., 2016, Gerault et al., 2017, Gerault et al., 2018]

Phase 1 (SAT solver)

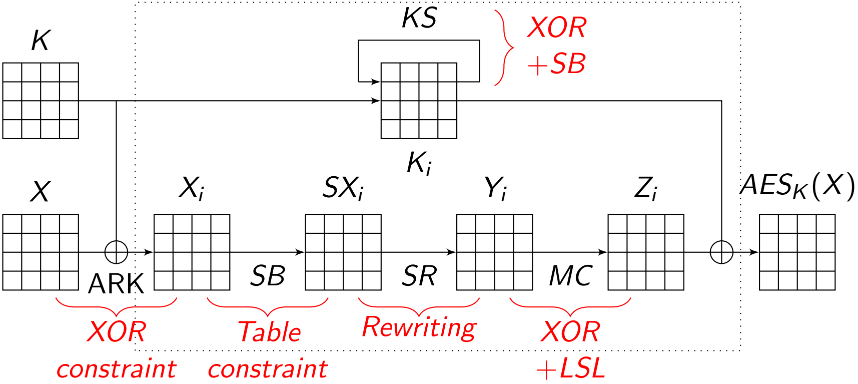
Abstract the bytes with variables

$$\Delta X = 0 \Leftrightarrow \delta X = 0^8$$
$$\Delta X = 1 \Leftrightarrow \delta X \in [1, 255]$$

Phase 2 (Constraint solver)

Find the real values of the bytes δX knowing ΔX

Can we solve it with BDDs ?



Conclusion, what I didn't talk about

Design and implementation of a BDD library for CP.

- Limited-width-BDDs,
- Bitwise propagators,
- Set operations

Theoretical work on BDDs

- Definition of merge value
- Analysis of the width
- Equivalence between domains

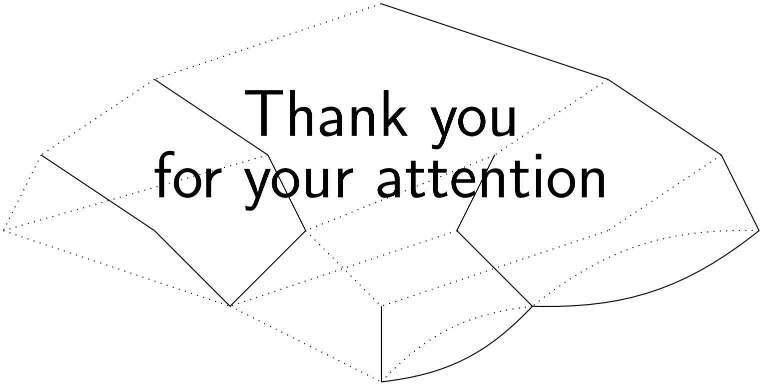
Future work

On BDDs:

- Integer domain: algebraic computations
- Global constraints: `all_different`, `global_cardinality`, `regular`
- Comparison with other domains (bit-vector domain)
- Improvement of heuristics
- General constraint $X \equiv Y \mod c$ (not $c = 2^k$)

About cryptography: project ANR Decrypt about CP and cryptography

- Analysis of other protocols
- Usage of BDDs to design protocols



Thank you for your attention

Andersen, H. R., Hadzic, T., Hooker, J. N., and Tiedemann, P. (2007).
A constraint store based on multivalued decision diagrams.
In *International Conference on Principles and Practice of Constraint Programming*, pages 118–132. Springer.

Bergman, D. (2013).
New techniques for discrete optimization.
PhD thesis, PhD thesis, Tepper School of Business, Carnegie Mellon University.

Bryant, R. E. (1986).
Graph-based algorithms for boolean function manipulation.
Computers, IEEE Transactions on, 100(8):677–691.

Dye, K. (2014).
Implementation of bit-vector variables in a cp solver, with an application to the generation of cryptographic s-boxes.

Gerault, D., Lafourcade, P., Minier, M., and Solnon, C. (2018).

Revisiting aes related-key differential attacks with constraint programming.
Information Processing Letters, 139:24–29.

Gerault, D., Minier, M., and Solnon, C. (2016).
Constraint programming models for chosen key differential cryptanalysis.
In *International Conference on Principles and Practice of Constraint Programming*, pages 584–601. Springer.

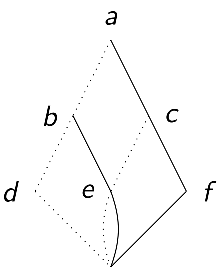
Gerault, D., Minier, M., and Solnon, C. (2017).
Using constraint programming to solve a cryptanalytic problem.
In *IJCAI 2017-International Joint Conference on Artificial Intelligence-Sister Conference Best Paper Track*, page 5.

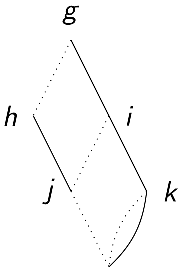
Michel, L. D. and Van Hentenryck, P. (2012).
Constraint satisfaction over bit-vectors.
In *Principles and Practice of Constraint Programming*, pages 527–543. Springer.

Minier, M., Solnon, C., and Reboul, J. (2014).

Solving a symmetric key cryptographic problem with constraint programming.
In *13th International Workshop on Constraint Modelling and Reformulation (ModRef), in conjunction with CP*, volume 14, pages 1–13.

Example: intersection

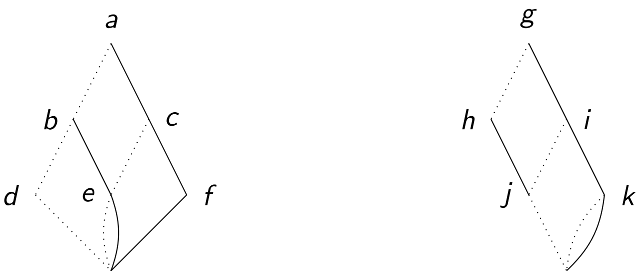




Intersections computed recursively: $M \cap M' = a \cap g$

- $b \cap h$
  - $d \cap F$
  - $e \cap j$
- $c \cap i$
  - $e \cap j$
  - $f \cap k$

Example: intersection



Intersections computed recursively: $M \cap M' = a \cap g$

- $b \cap h$
  - $d \cap F$
  - $e \cap j$
- $c \cap i$
  - $e \cap j$
  - $f \cap k$

Implementation of intersection

Idea

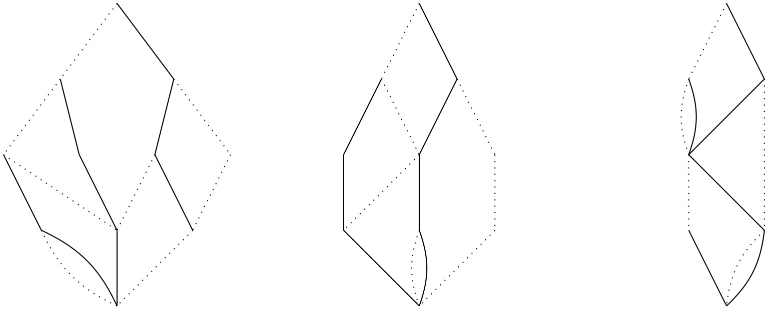
Use caching to compute only once the functions

```
Function INTER(B1, B2) returns B1 ∩ B2
    if (B1, B2) ∈ inter_hash then
        return FIND(inter_hash, (B1, B2))
    else
        if B1 = F or B2 = F then
            result ← F
        else if B1 = T or B2 = T then
            result ← T
        else
            Let a1, b1 such that B1 = Node(a1, b1)
            Let a2, b2 such that B2 = Node(a2, b2)
            result ← BDD_OF(INTER(a1, a2), INTER(b1, b2))
        end
        ADD_TO_HASH(inter_hash, (B1, B2), result)
        return result
    end
```

Limited-width-BDDs

Goal

Having a polynomial size representation (depending on the depth)



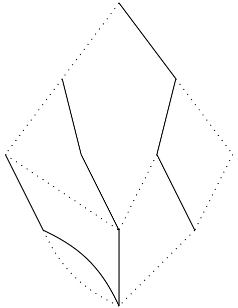
(a)
4-limited-width-BDD representing S

(b)
3-limited-width-BDD: $S \cup \{15\}$

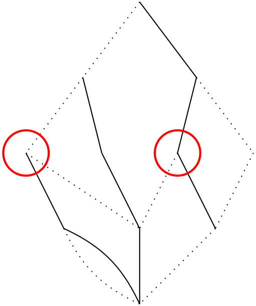
(c)
2-limited-width-BDD: $S \cup \{5, 6, 9, 15\}$

Figure 3: Representation of the set
$S = \{1, 2, 3, 7, 8, 13, 14\} = \{0001, 0010, 0011, 0111, 1000, 1101, 1110\}$

How to generate them ?

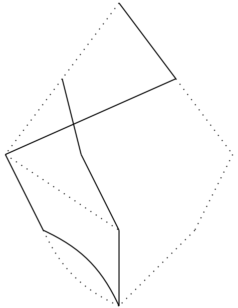


How to generate them ?



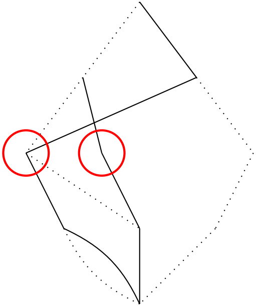
- Merging paths $00 \cdot \{01, 10, 11\}$ and $11 \cdot \{01, 10\}$

How to generate them ?



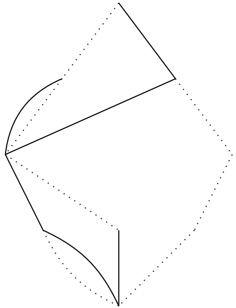
- Merging paths $00 \cdot \{01, 10, 11\}$ and $11 \cdot \{01, 10\}$
  - Adds the bit-vector 1111

How to generate them ?



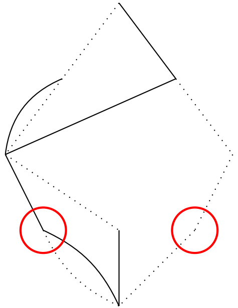
- Merging paths $00 \cdot \{01, 10, 11\}$ and $11 \cdot \{01, 10\}$
  - Adds the bit-vector 1111
- Merging paths $00 \cdot \{01, 10, 11\}$ and $01 \cdot 11$

How to generate them ?



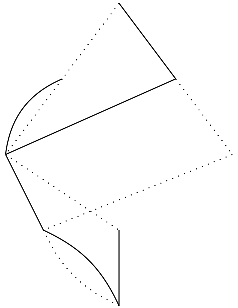
- Merging paths $00 \cdot \{01, 10, 11\}$ and $11 \cdot \{01, 10\}$
  - Adds the bit-vector 1111
- Merging paths $00 \cdot \{01, 10, 11\}$ and $01 \cdot 11$
  - Adds the bit-vectors $01 \cdot \{01, 10\}$

How to generate them ?



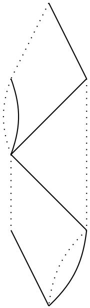
- Merging paths $00 \cdot \{01, 10, 11\}$ and $11 \cdot \{01, 10\}$
  - Adds the bit-vector 1111
- Merging paths $00 \cdot \{01, 10, 11\}$ and $01 \cdot 11$
  - Adds the bit-vectors $01 \cdot \{01, 10\}$
- Merging paths $001 \cdot \{0, 1\}$ and $100 \cdot 0$

How to generate them ?



- Merging paths $00 \cdot \{01, 10, 11\}$ and $11 \cdot \{01, 10\}$
  - Adds the bit-vector 1111
- Merging paths $00 \cdot \{01, 10, 11\}$ and $01 \cdot 11$
  - Adds the bit-vectors $01 \cdot \{01, 10\}$
- Merging paths $001 \cdot \{0, 1\}$ and $100 \cdot 0$
  - Adds the bit-vector 1001

How to generate them ?



- Merging paths $00 \cdot \{01, 10, 11\}$ and $11 \cdot \{01, 10\}$
  - Adds the bit-vector 1111
- Merging paths $00 \cdot \{01, 10, 11\}$ and $01 \cdot 11$
  - Adds the bit-vectors $01 \cdot \{01, 10\}$
- Merging paths $001 \cdot \{0, 1\}$ and $100 \cdot 0$
  - Adds the bit-vector 1001

What is the over-approximation

Definition (Merge Value)

The *merge value* of two nodes ($mv(u, v)$) is the number of bit-vectors added when merging the two nodes u and v.

Theorem

Let B be a BDD of root r, and u and v two nodes of the same layer. Let p_u (resp p_v) the number of paths that go from r to u (resp. v). The merge value is then

$$\begin{aligned} mv(u, v) &= p_u|\gamma(v)\backslash\gamma(u)| + p_v|\gamma(u)\backslash\gamma(v)| \\ &= p_u|\gamma(v)| + p_v|\gamma(u)| - (p_u + p_v)|\gamma(u) \cap \gamma(v)| \end{aligned}$$

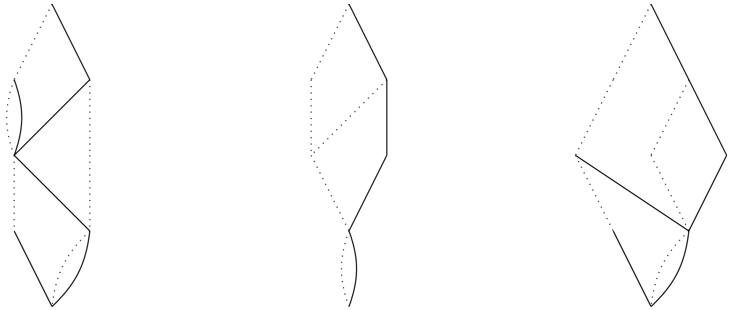
Consistency for limited BDDs

Issue

The intersection may increases the width

Solution from MDD-consistency

Consistency of M with respect to M': delete edges of M that are not in paths of M'.



(a) BDD M (b) BDD M' (c) BDD $M \cap M'$

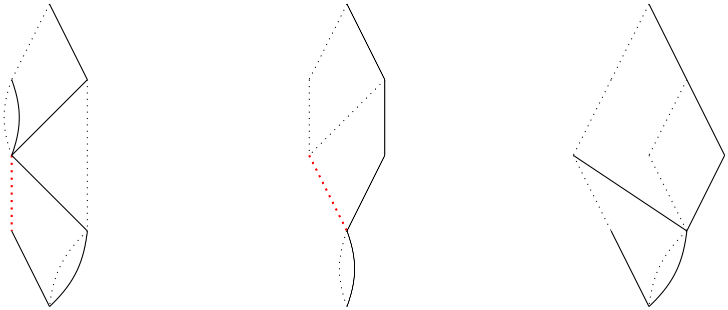
Consistency for limited BDDs

Issue

The intersection may increases the width

Solution from MDD-consistency

Consistency of M with respect to M': delete edges of M that are not in paths of M'.



(a) BDD M (b) BDD M' (c) BDD $M \cap M'$

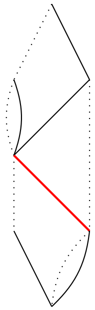
Consistency for limited BDDs

Issue

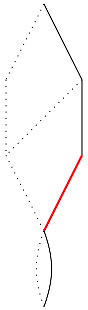
The intersection may increases the width

Solution from MDD-consistency

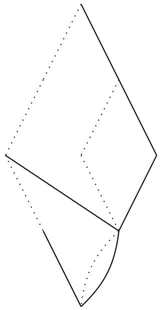
Consistency of M with respect to M': delete edges of M that are not in paths of M'.



(a) BDD M



(b) BDD M'



(c) BDD $M \cap M'$

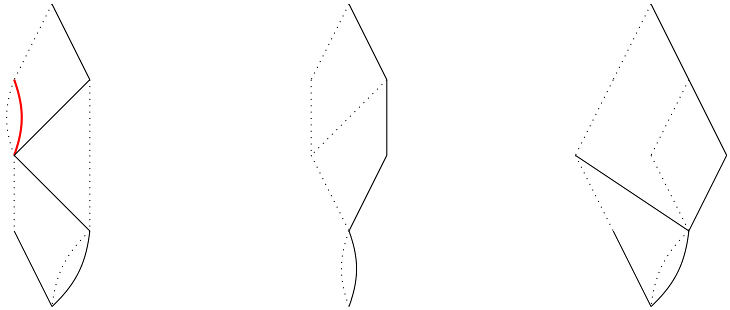
Consistency for limited BDDs

Issue

The intersection may increases the width

Solution from MDD-consistency

Consistency of M with respect to M': delete edges of M that are not in paths of M'.



(a) BDD M (b) BDD M' (c) BDD $M \cap M'$

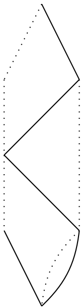
Consistency for limited BDDs

Issue

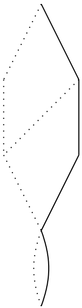
The intersection may increases the width

Solution from MDD-consistency

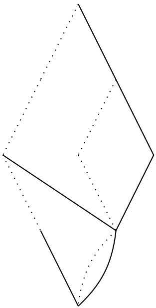
Consistency of M with respect to M': delete edges of M that are not in paths of M'.



(a) BDD M



(b) BDD M'



(c) BDD $M \cap M'$

Multiple consistency

Improvement

We often need to do the consistency of M w.r.t $\bigcup_i M_i$

Algorithm

Simply check if the edges are in paths of at least one BDD in the set

Cost

If the BDDs come from the same original BDD, we share a lot of computations.

Refining

Improvement

The width will only decrease with this consistency. This limits the expressiveness.

Solution 1

Choose nodes to split: have twice the same node in the BDD, an then do a propagation that will change one and not the other.
$\rightarrow$ Not really great with a functional implementation.

Solution 2

Increase the width during consistency.
$\rightarrow$ Great compromise between computing the intersection and doing the consistency