Knowledge Compilation Languages as Proof systems

Florent Capelli

July 12, 2019

22nd International Conference on Theory and Applications of Satisfiability Testing

Université de Lille, Inria, UMR 9189 - CRIStAL - Centre de Recherche en Informatique Signal et Automatique de Lille.

The problem #SAT

#SAT

input: CNF-formula $F = \bigwedge_i \bigvee_j \ell_{i,j}$

output: $\#F :=$ the number of satisfying assignments of F.

- Generic #P-complete problem
- Harder than SAT: $\mathrm{PH} \subseteq \mathrm{P}^{\#\mathrm{P}[1]}$ (Toda's Theorem)
- Hard even to approximate.
- Many practical (exact) #SAT-solvers: D4, Cachet, miniC2D, sharpSAT etc.

Trusting the tools

Should we trust the tools?

For SAT-solvers:

- Easy to check if an output assignment is indeed satisfying but what about UNSAT?
- Since 2013: mandatory certificate UNSAT Track of SAT competitions.
- Since 2016: mandatory in the Main Track.

Can we have the same level of exigence for #SAT-solvers?

What would be a certificate?

Certificates in SAT-solvers

Proofs are given in the `DRAT` format:

- Power comparable to (extended) resolution
- CDCL SAT-solvers can be easily modified to output DRAT certificates and it has been **crucial for adoption**:

> *Although resolution proof formats have been supported in the past, SAT Competition 2017 will only support clausal proofs. The main reason for this restriction is that* **no participant** *in recent years showed any interest in* **providing resolution** *as such proofs as* **too complicated to produce** *and they cost* **too much space to store***.*
>
> **Website of SAT Competition 2017**

Cook-Reckhow proof systems for $\#$SAT

Proof systems in the **Cook-Reckhow** sense:

- PTIME verifier $V\colon \text{CNF} \times \text{PROOFS} \to \mathbb{N} \cup \{\bot\}$
- $V(F, P)$ outputs
  - $\bot$ if P is not a valid proof for F
  - $N \in \mathbb{N}$ iff $N = \#F$
- Completeness: every CNF-formula F has a proof.

A naive proof system for #SAT

A proof that $\#F = N$ could be:

- The list $(S_1, \ldots, S_N)$ of satisfying assignments of F
- A refutation (*e.g.* using resolution) that

$$F \wedge \bigwedge_{i=1}^{N} \neg S_i$$

  is UNSAT[1].

Problems:

- The proof size $\geq \#F$
- Few formulas have small proofs: $x_1 \vee \cdots \vee x_n$ has no proof smaller than $2^n - 1$...

[1] S_i seen as a conjunction of literals on var(F).

Succinctly representing F

Can we prove $\#F$ without listing explicitly all solutions?

By **succinctly** representing all satisfying assignments of F in some data structure D:

1. D tractable: $\#F$ can be computed in time $poly(|D|)$
2. Check in PTIME that D actually represents F

Succinctly representing F

Can we prove $\#F$ without listing explicitly all solutions?

By **succinctly** representing all satisfying assignments of F in some data structure D:

1. D tractable: $\#F$ can be computed in time $poly(|D|)$
2. Check in PTIME that D actually represents F

Such data structures actually *almost* exist:

- Focus of Knowledge Compilation
- Represent Boolean functions with restricted Boolean circuits.
- For such a circuit C, querie such as counting can be solved in PTIME in $O(|C|)$.

Knowledge compilation

Many representations exist where **counting is tractable**:

- FBDD (Read-Once Branching Program)
- SDD
- d-DNNF
- Affine formulas etc.

Knowledge compilation

Many representations exist where **counting is tractable**:

- FBDD (Read-Once Branching Program)
- SDD
- d-DNNF
- Affine formulas etc.

Could an FBDD D be a certificate for $\#F$?

- Check that $D \Leftrightarrow F$ ie D represents **all models** of F.
- Compute $\#D$ in PTIME in $|D|$.

The good and the bad news

Unfortunately not enough:

- One can compute $\#D$ in PTIME.

The good and the bad news

Unfortunately not enough:

- One can compute $\#D$ in PTIME.
- One can check $D \rightarrow F$ in time $poly(D, F)$.
    - Thus one can check whether $\#D \leq \#F$.

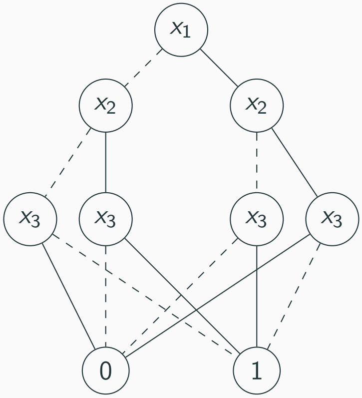
The good and the bad news

Unfortunately not enough:

- One can compute $\#D$ in PTIME.
- One can check $D \to F$ in time $poly(D, F)$.
    - Thus one can check whether $\#D \leq \#F$.
- Checking $F \to D$ is usually coNP-hard:
    - 0 can be succinctly represented.
    - Checking $F \to 0$ is equivalent to check whether F is UNSAT.

FBDD: Free Binary Decision Diagrams

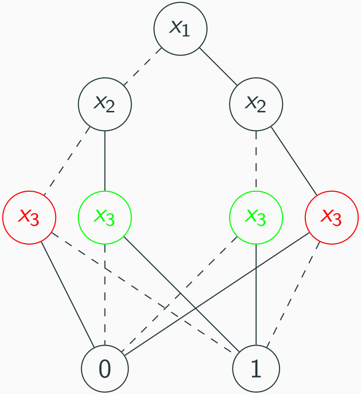
Compute a Boolean function by iteratively testing variables:



Read-Once: Each variable appears at most once on every path.

FBDD: Free Binary Decision Diagrams

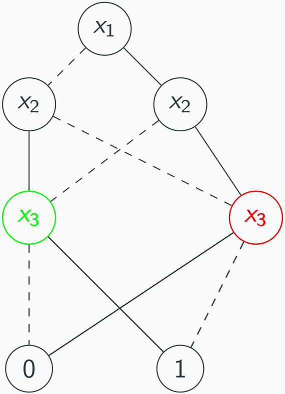
Compute a Boolean function by iteratively testing variables:



Read-Once: Each variable appears at most once on every path.

FBDD: Free Binary Decision Diagrams

Compute a Boolean function by iteratively testing variables:



Read-Once: Each variable appears at most once on every path.

Regular resolution

Resolution refutation is **regular** if on every path in the refutation DAG, each variable x is resolved at **most once**.

Regular resolution

Resolution refutation is **regular** if on every path in the refutation DAG, each variable x is resolved at **most once**.

Connection with FBDD: A regular refutation of F is

- FBDD with sinks labelled by clauses of F.
- Source to C-labelled sink path must refute C.

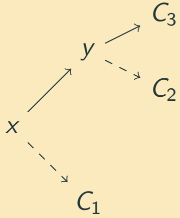
Regular resolution

Resolution refutation is **regular** if on every path in the refutation DAG, each variable x is resolved at **most once**.

Connection with FBDD: A regular refutation of F is

- FBDD with sinks labelled by clauses of F.
- Source to C-labelled sink path must refute C.

Refutation of $F = C_1 \wedge C_2 \wedge C_3 = x \wedge y \wedge (\neg x \vee \neg y)$



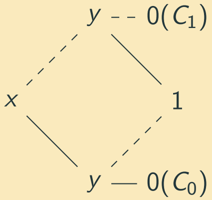
Certified FBDD

Previous model only works for **unsatisfiable** CNF: extend it with 1-sinks.

Certified FBDD

Previous model only works for **unsatisfiable** CNF: extend it with 1-sinks.

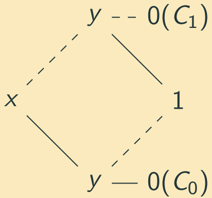
Enriched representation of $F = C_0 \wedge C_1 = (\neg x \vee \neg y) \wedge (x \vee y)$



Certified FBDD

Previous model only works for **unsatisfiable** CNF: extend it with 1-sinks.

Enriched representation of $F = C_0 \wedge C_1 = (\neg x \vee \neg y) \wedge (x \vee y)$



A **certified FBDD** D is **valid** if:

- 0-sinks are labeled with a clause C and
- each path from the source to a 0-sink must refute the label.
- If every label are clauses of F then: $\neg D \rightarrow \neg F$

Certified FBDD as proof system for $\#$SAT

Let D be a certified FBDD:

- Check in PTIME if D is valid.
- Check in PTIME if $F \Leftrightarrow D$
- $\#D$ can be computed in PTIME.

D **is a <u>proof</u> that** F **has exactly** $\#D$ **satisfying assignments.**

Can we find such proofs?

Finding a proof

Explore the solution space with caching:

Exhaustive DPLL: $\#F = \#F[x \mapsto 0] + \#F[x \mapsto 1]$

Finding a proof

Explore the solution space with caching:

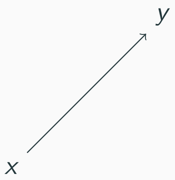
Exhaustive DPLL: $\#F = \#F[x \mapsto 0] + \#F[x \mapsto 1]$

x

Finding a proof

Explore the solution space with caching:

Exhaustive DPLL: $\#F = \#F[x \mapsto 0] + \#F[x \mapsto 1]$



Finding a proof

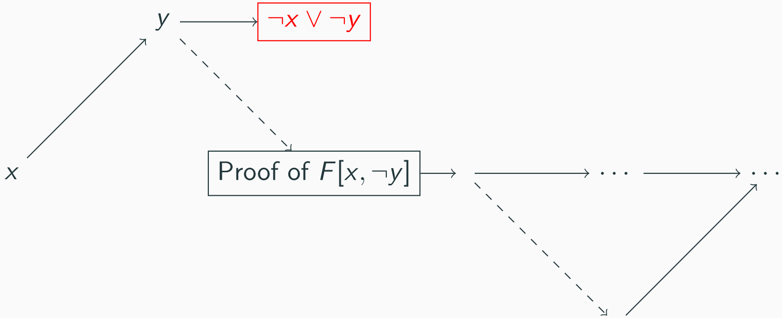
Explore the solution space with caching:

Exhaustive DPLL: $\#F = \#F[x \mapsto 0] + \#F[x \mapsto 1]$

$$\begin{tikzcd}
 & y \arrow[r] & \boxed{\neg x \vee \neg y} \\
x \arrow[ur] & &
\end{tikzcd}$$

Explore the solution space with caching:

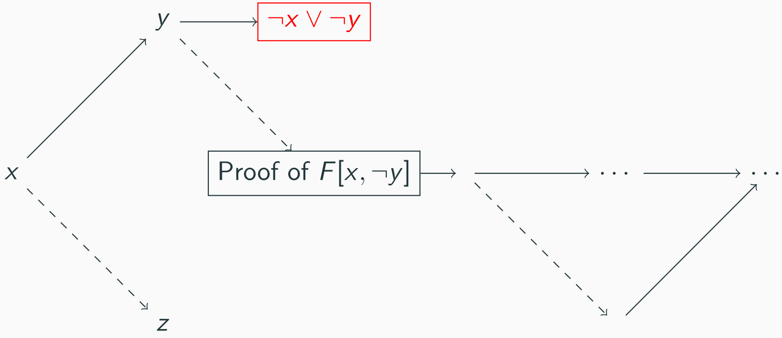
Exhaustive DPLL: $\#F = \#F[x \mapsto 0] + \#F[x \mapsto 1]$



Finding a proof

Explore the solution space with caching:

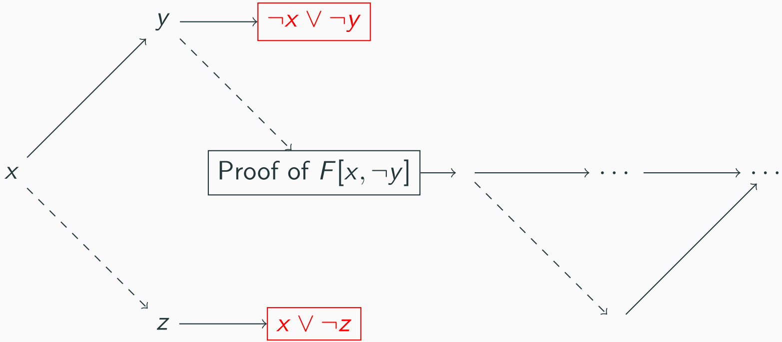
Exhaustive DPLL: $\#F = \#F[x \mapsto 0] + \#F[x \mapsto 1]$



Finding a proof

Explore the solution space with caching:

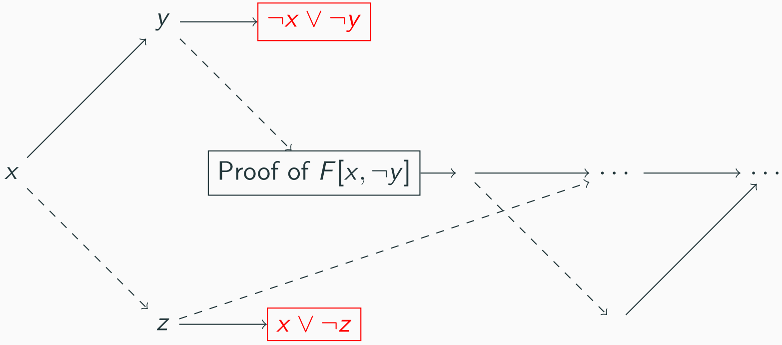
Exhaustive DPLL: $\#F = \#F[x \mapsto 0] + \#F[x \mapsto 1]$



Finding a proof

Explore the solution space with caching:

Exhaustive DPLL: $\#F = \#F[x \mapsto 0] + \#F[x \mapsto 1]$



Solving #SAT in practice : exhaustive DPLL

Most exact tools for #SAT are unfortunately not working exactly as before:

Exhaustive DPLL (#DPLL)

- $\#F = \#F[x \mapsto 0] + \#F[x \mapsto 1]$
- $\#(F_1 \wedge F_2) = \#F_1 \times \#F_2$ if $\mathrm{var}(F_1) \cap \mathrm{var}(F_2) = \emptyset$

Solving #SAT in practice : exhaustive DPLL

Most exact tools for #SAT are unfortunately not working exactly as before:

Exhaustive DPLL (#DPLL)

- $\#F = \#F[x \mapsto 0] + \#F[x \mapsto 1]$
- $\#(F_1 \wedge F_2) = \#F_1 \times \#F_2$ if $\mathrm{var}(F_1) \cap \mathrm{var}(F_2) = \emptyset$

Main work on:

- **Heuristics**: on how to choose x.
- **Caching policy**: cache already computed subformulas.
- **In practice**: D4, DMC, Cachet, miniC2D, sharpSAT etc.

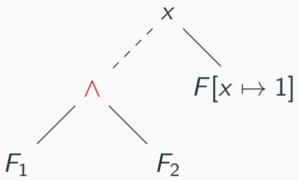
Solving #SAT in practice : exhaustive DPLL

Most exact tools for #SAT are unfortunately not working exactly as before:

Exhaustive DPLL (#DPLL)

- $\#F = \#F[x \mapsto 0] + \#F[x \mapsto 1]$
- $\#(F_1 \wedge F_2) = \#F_1 \times \#F_2$ if $\mathrm{var}(F_1) \cap \mathrm{var}(F_2) = \emptyset$

Underlying circuit of exhaustive DPLL: Decision-DNNF [Huang, Darwiche]:



DecDNNF = **FBDD + $\wedge$-gates with disjoint variables**

Certified DecDNNF

A certified DecDNNF D: same as certified FBDD with decomposable $\wedge$-gates:

- Checking that D is valid still PTIME
- Computing $\#D$ still PTIME.
- Checking $D \Leftrightarrow F$ PTIME: check that every root to 0-sink paths refutes the label.

Proof system for $\#$SAT.

Certified DecDNNF

A certified DecDNNF D: same as certified FBDD with decomposable $\wedge$-gates:

- Checking that D is valid still PTIME
- Computing $\#D$ still PTIME.
- Checking $D \Leftrightarrow F$ PTIME: check that every root to 0-sink paths refutes the label.

Proof system for $\#$**SAT.**

- Existing tools can be modified to output resembling certificates.

Circuit based proof systems

Certified DecDNNF = proof systems for every tractable task on DecDNNF.

Example:

- F CNF on variables X, $Y \subseteq X$. Find

$$\max_{\tau \models F} \#\{y \in Y \mid \tau(y) = 1\}.$$

Representation	Complexity
CNF	coNP-hard
DecDNNF	Linear

- Certified DecDNNF are a **proof system** for the **Hamming weight** problem.

Circuit based proof systems

Certified DecDNNF = proof systems for every tractable task on DecDNNF.

Example:

- F CNF on variables X, $Y \subseteq X$. Find

$$\max_{\tau \models F} \#\{y \in Y \mid \tau(y) = 1\}.$$

Representation	Complexity
CNF	coNP-hard
DecDNNF	Linear

- Certified DecDNNF are a **proof system** for the **Hamming weight** problem.

Application: proof system for maxSAT(F) by taking $Y = \{s_C \mid C \in F\}$ and CNF

$$\tilde{F} = \bigwedge_{C \in F} s_C \vee C$$

Conclusion

Take-away message:

- We introduced a proof system for #SAT, realistic for DPLL-based #SAT solver.
- If only $F \rightarrow D$ or $D \rightarrow F$: proof for lower/upper bound on $\#F$.
- Certified circuits may be used to certify other problems on CNF (though it may be more far fetched): maxSAT, minimal Hamming Weight, weighted satisfying assignment etc.

Future work

- Implement a **certified** Knowledge Compiler / #SAT solver (work in progress with J.M. Lagniez, P. Marquis and Fanny Canivet):
  - Modify D4 to output certificate,
  - Need to incorporate clauses learned by calling oracle SAT-solvers,
  - Implement a "checker"
- Can we certify other classes of circuits used in Knowledge Compilation?
- How does it compare with existing maxSAT proof systems?