On SAT representations of XOR constraints (towards a theory of good SAT representations)

Matthew Gwynne Oliver Kullmann

Computer Science Department
Swansea University
http://cs.swan.ac.uk/~csoliver/papers.html

LATA 2014
March 14, 2014

SAT solving

Still many textbooks on algorithms tell us:

If it's NP-complete, you can only do approximation ...

Slightly more advanced is the degradation of exact algorithms to "heuristics".

- That was always an Orwellian operation: "feasible = poly-time".
- Fortunately, not all practitioners and researchers listened.
- E.g. every microprocessor design today relies on SAT solving.

Today, due to SAT solvers, it is more appropriate to assume

$$P = NP$$

from a **practical point of view** — considering concrete, interesting problems!

Isn't this a slight exaggeration?

The further away a problem is from boolean logic, the harder it is for a SAT solver.

The task is to integrate
special reasoning of the problem domain
into SAT solving.

- Constraint solving (CSP) — that didn't work (for problems with a combinatorial core).
- Integration of such constraint reasoning into SAT solvers — doesn't seem to work currently.
- The SMT (SAT modulo Theory) framework: works, but only in an "industrial setting" (where the problems are big, but not really hard).

Our approach is to start a

Theory of "good" SAT representations

("good" representations integrate the special reasoning).

Upper and lower bounds for XOR constraints

Since CNF-representations are a restricted class of representations, we need

- upper and lower bounds on the sizes of "good" representations,
- where "good" can mean many things.

We consider the "case study" (interesting in its own rights) of

systems of XOR constraints
(parity constraints, systems of linear equations over $\mathbb{Z}_2$).

We show

- There is no good poly-size representation for the general case.
- But when considering parameters (like the number of equations), parts of a complex landscape with many non-trivial possibilities unfold (resp. we hope so).

Other approaches at intelligent XOR-translations

- While we show fpt in the number m of XOR-constraints, the weaker parameter n, the number of variables, was show fpt in Laitinen, Junttila, and Niemelä [18].
- Practical results (SAT benchmarks) for translating XOR-clause-sets into CNF-clause-sets are in Laitinen, Junttila, and Niemelä [17].
- These authors also introduced the DPLL(XOR) framework, for integrating dedicated XOR-reasoning into SAT solving (Laitinen, Junttila, and Niemelä [15, 16]).

The project: a theory of SAT representations

- The LATA 2014 paper Gwynne and Kullmann [12] (click), with underlying (arXiv) report Gwynne and Kullmann [10] (click).
- SOFSEM 2013 (click) and JAR (click) for the basic "hardness measures", measuring the "quality" of a representation: Gwynne and Kullmann [8, 11]
- Trading quality for size, showing that the various hardness measures yield hierarchies for the representation of boolean function: Gwynne and Kullmann [9] (arXiv; click)
- These "hardness measures" for proof complexity: Beyersdorff and Kullmann [4] (arXiv; click).

Outline

XOR-constraints I

Framework:

- Variable set $\mathcal{VA}$
- Literal set $\mathcal{LIT} \supset \mathcal{VA}$ with complementation $x \in \mathcal{LIT} \mapsto \overline{x} \in \mathcal{LIT}$
- **Clauses** are finite complement-free sets of literals: $\mathcal{CL}$.
- **Clause-sets** are finite set of clauses: $\mathcal{CLS}$.

By default, clause-sets are interpreted as CNF.

XOR-constraints II

We want to construct a "SAT representation" of some problem, which includes an XOR-constraint

$$x_1 \oplus \cdots \oplus x_n = \varepsilon,\ x_i \in \mathcal{LIT},\ \varepsilon \in \{0,1\}.$$

To make life easier, we assume $\varepsilon = 0$, and we handle that XOR-constraint simply as an

$$\textbf{XOR-clause } C := \{x_1, \ldots, x_n\} \in \mathcal{CL}.$$

And a system of XOR-constraint is simple handled as an

$$\textbf{XOR-clause-set } F \in \mathcal{CLS}.$$

The trivial representation of XOR-constraints I

There is precisely one CNF-clause-set, which is equivalent to an XOR-clause C, and we denote it by $\boldsymbol{X_0(C)} \in \mathcal{CLS}$.

$$\begin{aligned} X_0(\{a\}) &= \{\{\overline{a}\}\} \\ X_0(\{a,b\}) &= \{\{\overline{a},b\},\{a,\overline{b}\}\} \\ X_0(\{a,b,c\}) &= \{\{\overline{a},b,c\},\{a,\overline{b},c\},\{a,b,\overline{c}\},\{\overline{a},\overline{b},\overline{c}\}\} \end{aligned}$$

- $X_0(C)$ has 2^{n-1} clauses of length n.
- $X_0(C)$ is perfect for small n.

We can use X_0 piecewise, obtaining the first general translation:

$$X_0 : \mathcal{CLS} \to \mathcal{CLS}$$

$$X_0(F) := \bigcup_{C \in F} X_0(C).$$

The trivial representation of XOR-constraints II

$X_0(C)$ is actually the unique equivalent CNF —
what to do for large n ?!?

The standard representation of XOR-constraints I

To obtain a **small** translation for arbitrary XOR-clauses C, we use **auxiliary variables**. We split up C, using new variables y_i for partial sums, e.g. for $C = \{x_1, \ldots, x_4\}$:

$$x_1 \oplus x_2 = y_2,\ y_2 \oplus x_3 = y_3,\ y_3 \oplus x_4 = 0.$$

In general C is split into an XOR-clause-set F' with $n-1$ XOR-clauses, and we obtain the representation

$$\boldsymbol{X_1(C)} := X_0(F') \in 3\text{–}\mathcal{CLS}.$$

We got

$$\boldsymbol{X_1} : \mathcal{CL} \to 3\text{–}\mathcal{CLS}.$$

The standard representation of XOR-constraints II

So we can represent a single XOR-constraint. If we have many of them, we apply the translation piecewise, obtaining

$$\boldsymbol{X_1} : \mathcal{CLS} \to 3\text{–}\mathcal{CLS}.$$

That is, for a general XOR-clause-set $F \in \mathcal{CLS}$ we get the representation

$$\boldsymbol{X_1(F)} := \bigcup_{C \in F} X_1(C) \in 3\text{–}\mathcal{CLS},$$

where new variables are used for the different XOR-clauses in F.

How good is this representation?

We now have a representation $X_1(F) \in 3\text{–}\mathcal{CLS}$ for arbitrary sets F of XOR-clauses.

- This is the default representation, used nearly everywhere.
- But is it “good” ?
- And can we do “better” ?!

Isn't it trivial?

Why can't we just solve F (Gaussian elimination), and that's it?

The point is that there are other "constraints" in the complete SAT problem (besides the XOR-constraints)

— and thus we need to represent *precisely* the solutions of F.

A **CNF-representation** F of a boolean function f means:

- $\mathrm{var}(f) \subseteq \mathrm{var}(F)$,
- the satisfying assignments of F projected to $\mathrm{var}(f)$ are precisely the satisfying assignments of f.

Basic linear algebra, or?

Given an XOR-clause-set F —
just compute a basis of the solution space!

HOWEVER, we want to *recognise* a solution when given to the representation, and for that a basis is just in the way.

But then $X_1(F)$ does already the job ?!

- Unit-clause propagation r_1 is a basic inference mechanism (explained later), computable in linear time.
- r_1 suffices for a total assignment to the variables of F to evaluate $X_1(F)$ to `TRUE` or `FALSE`.

Partial instantiation

A SAT solver does not work with *total assignments*, but with

partial assignments.

So the representation $X_1(F)$ will become partially instantiated.

- It might become unsatisfiable.
- It is known, starting with the seminal work of Tseitin [21], that these unsatisfiable problems in general are hard for resolution, and thus for SAT solvers.

The task is thus:

Making the representation fully *explicit*,
so that every unsatisfiable partial instantiation is recognised
by a simple mechanism like unit-clause propagation.

So then online Gaussian elimination?

Alright, we then run Gaussian elimination again and again, after every update to the partial assignment.

- That's precisely what CSP is about.
- One needs then to expand Gaussian elimination by update- and undo-operations (which has been done).

But we **want more!**

We want **one** CNF-representation which does the job (modulo a simple operation like unit-clause propagation)!

- In this way we can use a standard SAT solver.
- MORE IMPORTANTLY, the CNF-representation is fully instantiatable, and gives a meaningful result for **all partial instantiations** (also the satisfiable ones).

Resolution

First we need "resolution trees":

1. The resolution rule, the fundamental proof (refutation) rule for clause-sets, is
$$\frac{C \cup \{x\} \qquad D \cup \{\overline{x}\}}{C \cup D}$$
for clauses C, D and literals x with $x \notin C$, $\overline{x} \notin D$.
2. A resolution tree
$$T : F \vdash \bot,$$
deriving the empty clause $\bot := \emptyset \in \mathcal{CL}$ from $F \in \mathcal{CLS}$, is a binary tree, where
   - the leafs are labelled with elements from F,
   - each inner node performs a resolution step,
   - and the root is labelled with $\bot$.

A clause-set F is unsatisfiable ($F \in \mathcal{USAT}$) iff there is $T : F \vdash \bot$.

Hardness of unsatisfiable clause-sets

As introduced in Kullmann [13, 14] and further expanded in Gwynne and Kullmann [8, 11]:

Definition

For $F \in \mathcal{USAT}$ the **(tree-)hardness hd(F)** $\in \mathbb{N}_0$ is the minimum of Horton-Strahler numbers $\mathrm{hs}(T)$ for $T : F \vdash \bot$.

Special cases are:

- $\mathrm{hd}(F) = 0 \Leftrightarrow \bot \in F$.
- $\mathrm{hd}(F) \leq 1$ iff F can be refuted by **input resolution** (that's $T : F \vdash \bot$ with $\mathrm{hs}(T) \leq 1$).

From Torán [20], Esteban and Torán [7] it follows that $\mathrm{hd}(F) + 1$ is the **space complexity** of tree resolution for F.

Hardness for satisfiable clause-sets

As studied in [8, 11] (first mentioned in Ansótegui, Bonet, Levy, and Manyà [1]):

Definition

For $F \in \mathcal{CLS}$ we define $\mathbf{hd(F)} \in \mathbb{N}_0$ as the maximum of $\mathrm{hd}(F')$ over all $F' \in \mathcal{USAT}$ obtained from F by partial instantiation of some variables.

A "good representation" of a boolean function f basically means a representation F with "small" $\mathrm{hd}(F)$.

Most fundamental the class

$$\mathcal{UC} := \mathcal{UC}_1 := \{F \in \mathcal{CLS} : \mathrm{hd}(F) \leq 1\}.$$

The class $\mathcal{UC}$ has been introduced in del Val [6] for the purpose of Knowledge Compilation, strengthening $\mathcal{UC}_0$, where $F \in \mathcal{UC}_0$ iff F is the set of all *prime implicates* of some boolean function.

Relative hardness

Definition

For a set V of variables we define the relative hardness

$$\mathbf{hd}^{V}(F) \in \mathbb{N}_0$$

by considering only assignments to variables in V (regarding the instantiations, yielding those unsatisfiable F').

For a representation F of f we say:

- F is a representation of **absolute hardness** k iff $\mathrm{hd}(F) = k$,
- while F is a representation of **relative hardness** k iff $\mathrm{hd}^{\mathrm{var}(f)}(F) = k$.

Unit-clause propagation

A basic mechanism in determining satisfiability is

unit-clause propagation (UCP).

For example:

$$\{\ \underbrace{\{a\}}_{\text{unit-clause}}, \{\overline{a}, b\}, \{\overline{b}\}\ \} \xrightarrow{\langle a \to 1\rangle} \{\ \{b\}, \{\overline{b}\}\ \} \xrightarrow{\langle b \to 1\rangle} \{\ \bot\ \}.$$

- Detects and sets some forced assignments, repeatedly.
- Possible in linear time, and is confluent.
- Using the map $\mathbf{r_1} : \mathcal{CLS} \to \mathcal{CLS}$ for UCP we have

$$\mathrm{r}_1(F) := \begin{cases} \{\bot\} & \text{if } \bot \in F \\ \mathrm{r}_1(\langle x \to 1\rangle * F) & \text{if } \exists\, x \in \mathrm{lit}(F) : \bot \in \langle x \to 0\rangle * F \\ F & \text{otherwise} \end{cases}.$$

Generalised unit-clause propagation

[13, 14] introduced the notion of

generalised unit-clause propagation

$\mathsf{r}_k : \mathcal{CLS} \to \mathcal{CLS}$, $k \in \mathbb{N}_0$.

$$\mathsf{r}_0(F) := \begin{cases} \{\bot\} & \text{if } \bot \in F \\ F & \text{otherwise} \end{cases}$$

$$\mathsf{r}_k(F) := \begin{cases} \mathsf{r}_k(\langle x \to 1\rangle * F) & \text{if } \exists\, x \in \mathrm{lit}(F) : \mathsf{r}_{k-1}(\langle x \to 0\rangle * F) = \{\bot\} \\ F & \text{otherwise} \end{cases}.$$

$\mathsf{r}_k(F)$ can be computed in time $\ell(F) \cdot n(F)^{2k-2}$.

Example: r_2 is more powerful r_1

$r_2 : \mathcal{CLS} \to \mathcal{CLS}$ is (full) *failed literal elimination*.

Consider

$$F := \{ \{a, b\}, \{a, \overline{b}\}, \{\overline{a}, b\}, \{\overline{a}, \overline{b}\} \}.$$

We have that

1. $r_1(F) = F$.
2. $r_2(F) = r_2(\langle a \to 1 \rangle * F) = \{\bot\}$, since

$$r_1(\langle a \to 0 \rangle * F) = r_1(\{ \{b\}, \{\overline{b}\} \}) = \{\bot\}.$$

Thus $\mathrm{hd}(F) = 2$.

Hardness via r_k

As shown in [13, 14]:

Theorem

For $F \in \mathcal{USAT}$ we have

$$\mathrm{hd}(F) = \min\{k \in \mathbb{N}_0 : \mathrm{r}_k(F) = \{\bot\}\}.$$

Monotonisation of boolean functions

Consider a boolean function f.

We want partial assignments to f,
handled by a boolean function $\widehat{f}$.

- Every variable is doubled.
- So we can encode "not assigned".

Now

$\widehat{f} = 0$ iff
the corresponding partial assignment
makes f unsatisfiable.

Example: the monotonisation of the bijective PHP^m_m function is the matching function (essentially).

Monotone circuits

Theorem

Consider a boolean function f and a representation F with

$$\mathrm{hd}^{\mathrm{var}(f)}(F) \leq 1.$$

From F we can compute in time $O(\ell(F) \cdot n(F)^2)$ a monotone circuit computing $\widehat{f}$.

Corollary

Boolean functions f_n have a CNF-representation F_n with $\mathrm{hd}^{\mathrm{var}(f_n)}(F_n) \leq 1$ and $\ell(F_n) = n^{O(1)}$ if and only if $\widehat{f_n}$ can be computed by monotone circuits of size polynomial in n.

(The predecessor of these results is Bessiere, Katsirelos, Narodytska, and Walsh [3] (with $\widehat{f}$ "hidden").)

No polysize good representations for XOR's

Exploiting Babai, Gál, and Wigderson [2] (monotone span programs):

Theorem

The size of representations of systems of XOR-constraints with bounded relative hardness is super-polynomial in the number of constraints.

Forced assignments

An assignment $\langle x \to 1 \rangle$ for a literal x and $F \in \mathcal{CLS}$ is called **forced**, if $\langle x \to 0 \rangle * F \in \mathcal{USAT}$.

Thus $\langle x \to 1 \rangle * F$ is sat-equivalent to F.

- So we can (and should!) apply the partial assignment $\langle x \to 1 \rangle$.
- Detection of a forced assignment is coNP-complete.
- So special cases need to be considered.
- The r_k detect and eliminate some forced assignments.
- With $k = n(F)$ we get all forced assignments.

Btw, for a forced assignment $\langle x \to 1 \rangle$ also the literal x is called **forced**.

Propagation hardness

Let $\mathsf{r}_\infty(F) := \mathsf{r}_{n(F)}(F)$, that is, r_∞ applies all forced assignments.

Now **phd(F)** $\in \mathbb{N}_0$ ("p-hardness") for $F \in \mathcal{CLS}$ is the

smallest k such that for all partial assignments φ
we have $\mathsf{r}_\infty(\varphi * F) = \mathsf{r}_k(\varphi * F)$.

Let $\mathcal{PC}_k := \{F \in \mathcal{CLS} : \mathsf{phd}(F) \leq k\}$.

- $\mathcal{UC}_{k-1} \subset \mathcal{PC}_k \subset \mathcal{UC}_k$.
- $\mathcal{PC}_1 = \mathcal{PC}$ was introduced in Pipatsrisawat and Darwiche [19], Bordeaux and Marques-Silva [5] (**unit-propagation complete**).

Relative p-hardness

For the definition of $\mathrm{phd}^V(F)$ for $F \in \mathcal{CLS}$ and a set V of variables one has to be a bit careful:

1. In the conference version ([12]) just the partial assignments φ were restricted to $\mathrm{var}(\varphi) \subseteq V$.
2. However also the forced literals x must be restricted to $\mathrm{var}(x) \in V$.

That is, for **relative p-hardness** k we only say that forced literals *over the original variables* are realised via r_k.

FPT in number of XOR-constraints I

Theorem

By adding all implied XOR-clauses to a system of m XOR-clauses, translating each via X_1, we obtain a representation of relative p-hardness 1, with running time fixed-parameter tractable (fpt) in m (i.e., running-time 2^m times a polynomial in the length of the system).

We believe that this can be strengthened in two dimensions:

- Instead of the "relative condition", we can obtain the "absolute condition" $\mathcal{PC}$.
- Instead of fpt in m, we can obtain fpt in the treewidth of the incidence graph.

FPT in number of XOR-constraints II

As a preliminary result in this direction, we can handle $m = 2$:

Lemma

By factoring out the common part of two XOR-clauses, we obtain a translation for $m = 2$ to $\mathcal{PC}$ in linear time.

Summary and outlook

I We believe there is a whole world to be discovered.

II Hopefully a theory of “good SAT representations” will emerge which truly brings theory (proof theory) and practice (SAT solving) together.

III The translation of XOR-systems is a good first test-case: Despite the bad news “no poly-size good representation”, there seem to be a lot of opportunities for good representations (under various circumstances).

End

(references on the remaining slides).

For my papers see
http://cs.swan.ac.uk/~csoliver/papers.html.

Bibliography I

[1] Carlos Ansótegui, María Luisa Bonet, Jordi Levy, and Felip Manyà. Measuring the hardness of SAT instances. In Dieter Fox and Carla Gomes, editors, *Proceedings of the 23th AAAI Conference on Artificial Intelligence (AAAI-08)*, pages 222–228, 2008.

[2] László Babai, Anna Gál, and Avi Wigderson. Superpolynomial lower bounds for monotone span programs. *Combinatorica*, 19 (3):301–319, March 1999.

[3] Christian Bessiere, George Katsirelos, Nina Narodytska, and Toby Walsh. Circuit complexity and decompositions of global constraints. In *Twenty-First International Joint Conference on Artificial Intelligence (IJCAI-09)*, pages 412–418, 2009.

Bibliography II

[4] Olaf Beyersdorff and Oliver Kullmann. Hardness measures and resolution lower bounds. Technical Report arXiv:1310.7627v2 [cs.CC], arXiv, February 2014.

[5] Lucas Bordeaux and Joao Marques-Silva. Knowledge compilation with empowerment. In Mária Bieliková, Gerhard Friedrich, Georg Gottlob, Stefan Katzenbeisser, and György Turán, editors, *SOFSEM 2012: Theory and Practice of Computer Science*, volume 7147 of *Lecture Notes in Computer Science*, pages 612–624. Springer, 2012.

[6] Alvaro del Val. Tractable databases: How to make propositional unit resolution complete through compilation. In *Proceedings of the 4th International Conference on Principles of Knowledge Representation and Reasoning (KR'94)*, pages 551–561, 1994.

Bibliography III

[7] Juan Luis Esteban and Jacobo Torán. Space bounds for resolution. *Information and Computation*, 171(1):84–97, November 2001.

[8] Matthew Gwynne and Oliver Kullmann. Generalising and unifying SLUR and unit-refutation completeness. In Peter van Emde Boas, Frans C. A. Groen, Giuseppe F. Italiano, Jerzy Nawrocki, and Harald Sack, editors, *SOFSEM 2013: Theory and Practice of Computer Science*, volume 7741 of *Lecture Notes in Computer Science (LNCS)*, pages 220–232. Springer, 2013. doi: 10.1007/978-3-642-35843-2_20.

[9] Matthew Gwynne and Oliver Kullmann. Trading inference effort versus size in CNF knowledge compilation. Technical Report arXiv:1310.5746v2 [cs.CC], arXiv, November 2013.

Bibliography IV

[10] Matthew Gwynne and Oliver Kullmann. On SAT representations of XOR constraints. Technical Report arXiv:1309.3060v4 [cs.CC], arXiv, December 2013.

[11] Matthew Gwynne and Oliver Kullmann. Generalising unit-refutation completeness and SLUR via nested input resolution. *Journal of Automated Reasoning*, 52(1):31–65, January 2014. doi: 10.1007/s10817-013-9275-8.

[12] Matthew Gwynne and Oliver Kullmann. On SAT representations of XOR constraints. In Adrian-Horia Dediu, Carlos Martín-Vide, José-Luis Sierra, and Bianca Truthe, editors, *LATA 2014: Language and Automata Theory and Applications, 8th International Conference*, volume 8370 of *Lecture Notes in Computer Science (LNCS)*, pages 409–420. Springer, 2014. doi: 10.1007/978-3-319-04921-2_33.

Bibliography V

[13] Oliver Kullmann. Investigating a general hierarchy of polynomially decidable classes of CNF's based on short tree-like resolution proofs. Technical Report TR99-041, Electronic Colloquium on Computational Complexity (ECCC), October 1999.

[14] Oliver Kullmann. Upper and lower bounds on the complexity of generalised resolution and generalised constraint satisfaction problems. *Annals of Mathematics and Artificial Intelligence*, 40 (3-4):303–352, March 2004.

[15] Tero Laitinen, Tommi Junttila, and Ilkka Niemelä. Extending clause learning DPLL with parity reasoning. In Helder Coelho, Rudi Studer, and Michael Wooldridge, editors, *ECAI 2010 – 19th European Conference on Artificial Intelligence*, pages 21–26. IOS Press, 2010.

Bibliography VI

[16] Tero Laitinen, Tommi Junttila, and Ilkka Niemelä. Extending clause learning SAT solvers with complete parity reasoning. In *ICTAI 2012 – 24th International Conference on Tools with Artificial Intelligence*, pages 65–72, 2012.

[17] Tero Laitinen, Tommi Junttila, and Ilkka Niemelä. Classifying and propagating parity constraints. In Michela Milano, editor, *Principles and Practice of Constraint Programming – CP 2012*, volume 7514 of *Lecture Notes in Computer Science*, pages 357–372. Springer, 2012.

[18] Tero Laitinen, Tommi Junttila, and Ilkka Niemelä. Simulating parity reasoning. In Ken McMillan, Aart Middeldorp, and Andrei Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning – LPAR 2013*, volume 8312 of *Lecture Notes in Computer Science*, pages 568–583. Springer, 2013.

Bibliography VII

[19] Knot Pipatsrisawat and Adnan Darwiche. On the power of clause-learning SAT solvers as resolution engines. *Artificial Intelligence*, 175(2):512–525, 2011.

[20] Jacobo Torán. Lower bounds for space in resolution. In *Computer Science Logic, 13th International Workshop, CSL'99*, volume 1683 of *Lecture Notes in Computer Science*, pages 362–373. Springer, 1999.

[21] G.S. Tseitin. On the complexity of derivation in propositional calculus. In *Seminars in Mathematics*, volume 8. V.A. Steklov Mathematical Institute, Leningrad, 1968. English translation: Studies in mathematics and mathematical logic, Part II (A.O. Slisenko, editor), 1970, pages 115-125.