General Perspectives of Translating CSP into SAT

Oliver Kullmann

Computer Science, Swansea University, UK

CSPSAT 2012, June 16, 2012
(invited tutorial)

Tutorial on what?

- A framework for thinking about solving CSP problems via SAT.
- Concentration on finite crisp CSPs.

Emphasise on interesting research direction

A vision

A theory of translation of a (single) abstract constraint function f into a boolean clause-set F

which understands f as a "sporadic" entity.

CSP = uniform (the interface / the languages)
SAT = non-uniform / sporadic (the machine / the algorithms)

no more "one size fits all" —
every f gets its dedicated translation.

Two stages

The translation of f into F happens in two steps:

1. The (**abstract**) constraint function f is **encoded** into an (**abstract**) boolean function f'.
2. The boolean function f is **represented** via a clause-set F.

Thus "translation" consists of first "encoding" then "representation".

- "Encoding" takes care of the variables (the "meaning").
- "Representation" produces the "good" CNF (the "algorithmic side").

It seems essential to me to split
the usual "encoding" (as the whole process)
into these two steps — we need to study their **interplay**!

Outline

TOC: Semantics

1. variables, values
2. partial and total assignments
3. constraint functions and boolean functions
4. semantics
5. boolean algebra
6. isomorphic encodings.

Variables, values

We consider

- a universe $\mathcal{VA}$ of "variables",
- a set (the "universal domain") $\mathcal{DOM}$ of "values".

We consider here

1. only finite $\mathcal{DOM}$
2. with $|\mathcal{DOM}| \geq 2$.

Typically we consider two universal domains at the same time:
the domain $\mathcal{DOM}$ of the constraint problem,
and the domain $\{0, 1\}$ of the boolean translation.

Total assignments, constraint functions

For $V \subseteq \mathcal{VA}$ we use

$$\boldsymbol{\mathcal{TASS}(V)} := \mathcal{DOM}^V = \{f : V \to \mathcal{DOM}\}$$

for the set of **total assignments**.
A **constraint function** is map

$$f : \mathcal{TASS}(V) \to \{0, 1\}$$

for some finite $V \subset \mathcal{VA}$. We use:

- $\mathbf{var(f)} := V$.
- $\boldsymbol{\mathcal{CF}} = \mathcal{CF}(\mathcal{VA}, \mathcal{DOM})$ is the set of all constraint functions.
- $\boldsymbol{\mathcal{BF}} := \mathcal{CF}(\mathcal{VA}, \{0, 1\})$ is the set of all **boolean functions**.

The goal

Given a single constraint function f,
translate f in an **"intelligent way"** into a (boolean) CNF F.

- To emphasise: we want to employ

  all "possible" information

  on f, not restricting to the use of some language, framework etc.
- Of course, f can not be just a black box, but we want to emphasise that

  a good SAT translation should take advantage of the specific f,

  not taking it as coming from some (constraint) framework.
- Thus the translation should be "algorithmic", not "schematic".

Semantics

- A **partial assignment** is a map $\varphi : V \to \mathcal{DOM}$ for some finite $V \subset \mathcal{VA}$.
- The set of all partial assignments is $\mathcal{PASS}$.
- For $\varphi \in \mathcal{PASS}$ and $f \in \mathcal{CF}$ with $\mathrm{var}(\varphi) \supseteq \mathrm{var}(f)$ we use $\mathbf{f(\varphi)} := f(\varphi \mid V)$.

Now we can define **implication** for $f, g \in \mathcal{CF}$:

$$\mathbf{f} \models \mathbf{g} :\Longleftrightarrow \forall\, \varphi \in \mathcal{PASS} : \mathrm{var}(\varphi) \supseteq \mathrm{var}(f) \cup \mathrm{var}(g) \Rightarrow f(\varphi) \leq g(\varphi).$$

And **equivalence** is defined as bi-implication:

$$\mathbf{f} \cong \mathbf{g} :\Longleftrightarrow f \models g \wedge g \models f.$$

SEMANTICS!

- In the SAT world, there is typically a syntactical point of view, plus algorithmic transformations via equivalence and sat-equivalence.
- The notion of a "constraint" from CSP is often (not always) a strange hybrid between semantics, syntax and algorithmics.

With the notion of *constraint functions* and *boolean functions* we want to provide a true semantical underpinning.

Study the space of **all encodings** of a constraint function!
Study the space of **all representations** of a boolean function!

Boolean operations

For constraint functions $f, g \in \mathcal{CF}$ we have the natural operations

$$f \wedge g, \quad f \vee g, \quad \neg f \quad \in \mathcal{CF}$$

where $\mathrm{var}(f \square g) = \mathrm{var}(f) \cup \mathrm{var}(g)$.

- For finite $V \subset \mathcal{VA}$ we also have $0^V, 1^V \in \mathcal{CF}$.
- We use $0 := 0^\emptyset$ and $1 := 1^\emptyset$.
- $(\mathcal{CF}, \vee, \wedge, 0, 1)$ is an algebraic structure close to a boolean algebra.

Boolean algebra

Reminder: The standard model of a free boolean algebra over variables $\mathcal{VA}$ (yielding the free generators) is propositional logic modulo equivalence.

Lemma

$\mathcal{CF}' := \mathcal{CF}/\cong$ is a boolean algebra.

1. *If $\mathcal{VA}$ is finite:*
   1. $|\mathcal{CF}'| = 2^{(|\mathcal{DOM}|^{|\mathcal{VA}|})}$.
   2. *$\mathcal{CF}'$ is a free boolean algebra iff $|\mathcal{DOM}| = 2^k$ for some $k \in \mathbb{N}$.*
2. *If $\mathcal{VA}$ is infinite:*
   1. $|\mathcal{CF}'| = |\mathcal{VA}|$.
   2. *$\mathcal{CF}'$ is a free boolean algebra.*

Proof.

For $|\mathcal{DOM}| = 2^k$ an isomorphism is the "logarithmic encoding", while a finite boolean algebra B is free iff $|B| = 2^{2^n}$ for some $n \in \mathbb{N}_0$. For infinite $\mathcal{VA}$ the approximation error vanishes as $k \to \infty$. □

Details of the isomorphisms

The isomorphisms realise a first form of encoding:

isomorphic encodings.

For finite $\mathcal{VA}$ (using $n := |\mathcal{VA}|$):

1. Easiest the logarithmic encoding:
   1. Choose $\mathcal{DOM} = \{d_1, \ldots, d_{2^k}\}$.
   2. Use k variables for the bits of the binary representation of the indices.
2. But that's not everything:
   1. We get all isomorphisms by listing all $(2^k)^n = 2^{kn}$ total assignments,
   2. in any order, and then use binary representations, with kn binary positions (aka variables), as above.
3. In any case the number of solutions is maintained.

For infinite $\mathcal{VA}$ we have more flexibility: the boolean variables must still be "independent", but no "equal partitioning" anymore.

Q1: The greater freedom should be useful?!

- Considering finite $\mathcal{VA}$, it should be useful to consider non-standard isomorphic encodings,

  since splitting could be more "intelligent" ?!
- It seems essential here, that the boolean variables do not have to respect the boundaries of the original variables.
- For infinite $\mathcal{VA}$, can we make use of the still greater freedom !?
- We can use here a flexible, kind of "continuous" encoding.

Remark: For countable infinite $\mathcal{VA}$ we have $|\mathbb{R}|$-many automorphisms of $\mathcal{BF}(\mathcal{VA})$ (which correspond to the isomorphisms between $\mathcal{CF}$ and $\mathcal{BF}$). These automorphisms correspond precisely to the homeomorphisms of the Cantor set.

Wild encodings

For every constraint function f there is an isomorphic encoding (using finite $\mathcal{VA}$ is enough, padding the values) trivialising f:

Just sort the total assignments into first (say) the falsifying ones,
then the satisfying ones —
these constraint functions have "perfect" boolean translations.

Sure, that's too brutal — but it shows that in this direction something could be gained.

Q2: Is logarithmic enough?

Is every other translation simulatable by just the logarithmic encoding,

when we allow **new variables**?

(I guess not. Via these new variables computations can be "performed", but these computations won't be "understandable" by SAT solvers.)

TOC: "Syntax"

1. clause-sets
2. partial assignments
3. UCP
4. QCNF representations
5. CNF representation
6. circuit representations.

Clause-sets and CNFs

- Literals are variables $v \in \mathcal{VA}$ and their **complements** $\overline{\mathbf{v}}$.
- A clause C is a finite and clash-free set of literals, i.e., $C \cap \overline{C} = \emptyset$.
- A **clause-set** is a finite set of clauses.
- The set of all clause-sets is $\boldsymbol{\mathcal{CLS}}$.

For example

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

is a clause-set. The default interpretation is

$$\mathrm{CNF}(F) = (a \vee b) \wedge (\neg a \vee c) \in \mathcal{BF}.$$

Remark: Clause-sets should be considered as (precise) combinatorial objects, as generalised hypergraphs.

Applying partial assignments

The application of a partial assignment $\varphi \in \mathcal{PASS}$ to a clause-set $F \in \mathcal{CLS}$ is denoted by

$$\varphi * F \in \mathcal{CLS}.$$

Satisfied clauses are removed, then falsified literals.

This yields an operation of the monoid $\mathcal{PASS}$ on the set $\mathcal{CLS}$.

A special clause-set is $\top := \emptyset$, a special clause is $\bot := \emptyset$.

- F is **satisfiable** iff there is $\varphi \in \mathcal{PASS}$ with $\varphi * F = \top$.
- $\top$ is satisfiable.
- $\{\bot\}$ is unsatisfiable.
- More generally, every F with $\bot \in F$ is unsatisfiable.

F is unsatisfiable iff $\mathrm{CNF}(F) \cong 0$.

UCP

By

$$\mathsf{r}_1 : \mathcal{CLS} \to \mathcal{CLS}$$

unit-clause propagation is denoted, that is,

- applying $F \rightsquigarrow \langle x \to 1 \rangle * F$ as long as there are unit-clauses $\{x\} \in F$,
- and reducing $F \rightsquigarrow \{\bot\}$ in case of $\bot \in F$.

For example

$$\mathsf{r}_1(\{\{a\}, \{a, b\}, \{\overline{a}, c\}, \{c, d\}, \{d, e\}\}) = \{\{d, e\}\}.$$

QCNF representations (too powerful)

QCNFs (quantified conjunctive normal forms) F yield a natural framework for representations of boolean functions:

1. The free variables of F are the variables of the boolean function f.
2. A satisfying assignment for f is one making F true.

The natural framework for "SAT representations" seems to be the Σ_1-CNF representations (allowing only existential quantifiers).

So the evaluation problem is not as hard as PSPACE.
Still, for evaluating our representations, we need to solve NP-problems.
So some further conditions will be needed.

Representing boolean functions

Spelling it out, and also pushing the existential quantifiers to the background:

- A clause-set F is a **(CNF-)representation** of the boolean function f if $\mathrm{var}(f) \subseteq \mathrm{var}(F)$ and the satisfying assignments of F projected to $\mathrm{var}(f)$ are (precisely) the satisfying assignments of f.
- Important special case: $\mathrm{var}(f) = \mathrm{var}(F)$, i.e., *without using new variables* (so F is equivalent to f).

We have $F \models f$, while in the other direction we need an extension of the satisfying assignment.
Using QCNF we can say that F is a representation of f iff

$$f = \exists_{v \in \mathrm{var}(F) \setminus \mathrm{var}(f)} F.$$

Examples

- $\{\{a, b\}, \{\overline{a}, c\}\}$ is a representation of $(a \vee b) \wedge (\neg a \vee c)$.
- Also $\{\{a, b\}, \{\overline{a}, c\}, \{d, e\}\}$ is a representation of the same function.
- Also $\{\{a, b\}, \{\overline{a}, c\}, \{b, c\}\}$ is.
- While $\{\{a, b, c\}\}$ is not.
- Neither is $\{\{a\}, \{b\}, \{c\}\}$

Question: which of the three representations is "strongest"?

Why new variables?

When the boolean functions are used as "constraints", that is, we patch together various constraints to obtain our translations, then the additional variables need to be **new variables**.

So well, what's now their point? Without them evaluation would be trivial!

It gives us more space, for better representations.

For example, very simple boolean functions (given by short DNFs) don't have short representations without new variables!

Remark: Allowing that a class $\mathcal{C} \subseteq \mathcal{CLS}$ of clause-sets is allowed to represent boolean functions via new variables is called "existential closure" in the knowledge-compilation literature. See Bordeaux, Janota, Marques-Silva, and Marquis [5] for a recent overview.

Less powerful: Circuits (still too much)

In the SAT context, a natural restriction on (CNF-)representations is to demand that evaluation is efficient.

- For example evaluation must be possible by unit-clause propagation.
- More precisely, the representation F of f is "1-effective" iff for every $\varphi \in \mathcal{PASS}$ with $\mathrm{var}(\varphi) = \mathrm{var}(f)$ we have $\mathrm{r}_1(\varphi * F) \in \{\{\bot\}, \top\}$.
- Such representations are basically "the same" as circuits.

We will actually consider here only further restricted forms of representations — we want "good" representations!

TOC: From constraints to clause-sets

1. translations
2. encodings
3. direct encoding

The two steps

We can now specify the **translation process** $f \rightsquigarrow F$ somewhat more precisely:

Given is a constraint function $f \in \mathcal{CF}$.

1. First an "encoding" is performed, yielding a boolean function $f' \in \mathcal{BF}$.
2. Second a representation F of f' is chosen.

What now is an "encoding"?

Encodings

When is a boolean function $f' \in \mathcal{BF}$ an "encoding" of $f \in \mathcal{CF}$?:

1. Most relaxed is satisfiability-equivalence: f' must be satisfiable iff f is.
2. A stronger requirement is that there is a many-to-many relation (a left- and right-total relation) between the solutions of f' and the solutions of f.
   Conditions one this relation range from completely open (then we just have sat-equivalence) to efficiently and uniformly computable.
   Examples are the general isomorphic encodings (arbitrary $\mathcal{VA}$).
3. Strongest is a bijective relation.
   At the weakest level we just have that; for example the "wild" isomorphic encodings (finite $\mathcal{VA}$)
   At the strongest level we have efficient and uniform computability; for example the logarithmic isomorphic encodings.

The direct encoding

- The boolean variables are $v_\varepsilon \in \mathcal{VA} \times \mathcal{DOM}$.
- v_ε is true iff "$v = \varepsilon$".
- The encoding $D : \mathcal{CF} \to \mathcal{BF}$ is:
  - injective (not surjective)
  - $\wedge$- and $\vee$-homomorphic
  - not $\neg$-homomorphic (due to the "slack").
- Preserves number of solutions.

The boolean function relating the logarithmic and the direct encoding has "very good" representations:

Thus by adding new variables and defining equations we can "strongly simulate" the direct by the logarithmic encoding (such that a SAT solver can "understand" it).

Towards a theory of degrees of "goodness"

Now we have given a **boolean function** f and we are seeking a "good" representation F.

- In the remainder F is always a (CNF-)representation of f.
- At the first state we arrive at "very good representations" via the classes $\mathcal{PC}$ ("propagation completeness") and $\mathcal{UC}$ ("unit-refutation completeness").
- However this covers only a very small part of all representations — we need to measure how "good" the representation is!
- For that purpose we introduce the notion of **hardness** $\mathrm{hd} : \mathcal{CLS} \to \mathbb{N}_0$ (tree-resolution hardness).
- The smaller the hardness the better the representation.

TOC: Two dimensions

1. prime implicates, forced literals
2. PC and UC (two marks of the first dimension)
3. relative versus absolute (two marks of the second dimension).

Prime implicates (level 0)

- An **implicate** of a boolean function f is a CNF-clause C with $f \models C$.
- It is **prime** iff no literal can be removed (recall, a CNF-clause get the stronger the shorter it is).
- We denote by $\mathbf{prc_0(f)}$ the set of all prime implicates of f.

The clause-set $F = \mathrm{prc}_0(f)$ is a very strong representation of f, basically the (unique) representation of hardness 0 — but in general too large.

The whole effort of "good representations" can (could) be understood as compression of $\mathrm{prc}_0(f)$.

Forced literals and assignments

- A literal x is **forced** for a boolean function f iff $f \models x$.
- For a clause-set F this is equivalent to $\langle x \to 0 \rangle * F \notin \mathcal{SAT}$.
- Accordingly we say that the assignment $\langle x \to 1 \rangle$ is **forced**.

By the way, we use throughout the basic logical law

$$f \models g \Leftrightarrow f \wedge \neg g \models 0.$$

Note that if g is a clause, then $\neg g$ is essentially a partial assignment.

GAC (= relative PC)

The works in SAT being concerned about "good" representations F for f borrowed the concept of "Generalised Arc Consistence" (**GAC**) from CSP in the following way:

> If after applying a partial assignment φ to f we obtain a forced assignment, then we obtain it from $\varphi * F$ via UCP.

More precisely: Let $\mathsf{r}_\infty : \mathcal{CLS} \to \mathcal{CLS}$ denote full elimination of forced assignments. Then the condition is

$$\forall\, \varphi \in \mathcal{PASS} : \mathrm{var}(\varphi) \subseteq \mathrm{var}(f) \Rightarrow \mathsf{r}_\infty(\varphi * F) = \mathsf{r}_1(\varphi * F).$$

PC ("absolute")

We cared about the prime implicates of f.

What about the prime implicates of F ?!
In other words, what about *arbitrary* partial assignments?!

We argue that we need also the prime implicates of F. There is a suitable concept in the literature (Bordeaux and Marques-Silva [4]), namely "Propagation Completeness" (**PC**), being investigated rather recently and employed under different circumstances:

A clause-set F is PC iff whenever $\varphi * F$ has a forced assignment, then we obtain it from $\varphi * F$ via UCP.

More precisely:

$$\mathcal{PC} := \{F \in \mathcal{CLS} \mid \forall\, \varphi \in \mathcal{PASS} : \mathsf{r}_\infty(\varphi * F) = \mathsf{r}_1(\varphi * F)\}.$$

UC more fundamental than PC I

We do *not* take $\mathcal{PC}$ as our starting point and generalise it to $\mathcal{PC}_k$ — this is a later step (not discussed here)!

Instead we start with "unit-refutation complete", called UC here, as first investigated in del Val [7].

$$\mathcal{UC} := \{C \in \mathcal{CLS} \mid \forall\, \varphi \in \mathcal{PASS} : \varphi * F \notin \mathcal{SAT} \Rightarrow \mathsf{r}_1(\varphi * F) = \{\bot\}\}.$$

We have

$$F \in \mathcal{UC} \iff \forall\, C \in \mathsf{prc}_0(F) : \mathsf{r}_1(\varphi_C * F) = \{\bot\},$$

where $\varphi_C := \langle x \to 0 : x \in C\rangle$.

UC more fundamental than PC II

This condition is simpler than propagation-completeness and directly related to proof complexity, as shown in Kullmann [10, 11].

- By definition $\mathcal{PC} \subset \mathcal{UC}$.
- In del Val [7] methods for computing representations $F' \in \mathcal{UC}$ of $F \in \mathcal{CLS}$ are given, however only *without new variables*.
- This is reasonable, since only "sporadic" F are considered, and for such "random" inputs no algorithmic methods are known (yet), which could handle new variables.
- However in the general context of *representation*, new variables are important.

Level 1 of "goodness"

- Level 0 of "goodness" was the condition $\mathrm{prc}_0(F) \subseteq F$.
- Level 1 of "goodness" is the condition $F \in \mathcal{UC}$.

Can we generalise this approach?! (And can we put $\mathcal{PC}$ into the picture?)

TOC: Some discussions

1. relative versus absolute
2. knowledge compilation
3. succinctness
4. possibilities other than hardness.

Relative versus absolute

It seems that the interaction of new variables and the PC/UC-condition has not been discussed yet:

- From a knowledge compilation point of view it makes good sense to consider UC with new variables, but using the relative condition, as pointed out by Bordeaux et al. [5].
- They favour the relative over the absolute condition, though they can not prove it to be more succinct.
- Fair enough! They only want to represent knowledge, and then one doesn't need queries using the new variables.
- However SAT solvers should not be impeded!

Q3: Can we always reach absoluteness?

Likely there are sequences of boolean functions such that

> we have good representations for the relative condition,
> but no good representations for the absolute condition.

"Good" means now

> polysize and bounded hardness (relative or absolute).

Likely this can be shown for the whole range of the $\mathcal{PC}_k, \mathcal{UC}_k$-hierarchy.

Knowledge compilation?

The area of knowledge compilation (AI) has some relations to the theory of "good representations F of a boolean function f",

- since "all knowledge" about f should be presented in F,
- and this in an "accessible form",
- but the question is *how* the "retrieval" works!

"Good" SAT representations needs to be "understandable" by SAT solvers. This is rather different from knowledge compilation, where one has much more freedom in the retrieval mechanism.

And furthermore, although the prime implicates of f are surely "prime knowledge" of f, that's not all — "absolute access" is needed.

Size is not everything

Under appropriate complexity-theoretical assumptions:

- QCNF is more succinct than Σ_1-CNF.
- Σ_1-CNF is more succinct than circuits.

Provably Jukna [9] (Section 9.11):

Circuits are more succinct than monotone circuits.

As shown in Bessiere, Katsirelos, Narodytska, and Walsh [3]:

relative UC $\sim$ relative PC $\sim$ monotone circuits.

Also our hierarchy collapses under the relative view-point.

This total collapse of everything to monotone circuits is due to the unaccountable handling of new variables.

This (we conjecture!) changes with the absolute condition.

All conclusions must be "nicely" derivable

Our general framework for a "good" representation F of f is:

All conclusions $F \models C$ must be "easily derivable".

Note again, that we consider all implication of F, not just those of f,

since splitting on new variables is important (the absolute condition).

So reasonable first measures are

- the binary logarithm of the maximum of (dag-)resolution complexity of deriving $F \models C$,
- the binary logarithm of the maximum of tree-resolution complexity of deriving $F \models C$.

A "good representation" then has logarithmically bounded measures.

We believe that "hardness" has nicer properties, but
especially the full resolution point of view might have potential.

TOC: A hierarchy of reductions

$$\mathsf{r}_k : \mathcal{CLS} \to \mathcal{CLS}.$$

See [10, 11] for the theory.

Generalised UCP

Definition

The maps $\mathbf{r_k} : \mathcal{CLS} \to \mathcal{CLS}$ for $k \in \mathbb{N}_0$ are defined as follows:

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

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

Lemma

The maps are well-defined (don't depend on the choices), and apply (only) forced assignments. The bigger k the more forced assignments are found, and r_∞ applies all forced assignments.

We have $\mathrm{r}_\infty(F) = \mathrm{r}_{n(F)}(F)$ for all $F \in \mathcal{CLS}$.

Practical applications

- r_1 is UCP (unit-clause propagation).
- r_2 is failed-literal reduction (full).
- Running $r_0, r_1, \ldots$ achieves quasi-automatisation of tree-resolution.
- Remark: In [10, 11] there is also an algorithmic extension of hardness to satisfiable clause-sets, and so we have also "hardness" for finding satisfying assignments.
- This approach is also the kernel of the Stalmarck-approach (at CNF-level).

TOC: The measure

$$\mathsf{hd} : \mathcal{CLS} \to \mathbb{N}_0.$$

"Hardness"

The hierarchy of r_k-reductions yields the following notion of **hardness**:

Definition

The **hardness** $\mathrm{hd}(F)$ of unsatisfiable F is the minimal k with $r_k(F) = \{\bot\}$.

1. $2^{\mathrm{hd}(F)} \leq \mathrm{Comp}^*_R(F) \leq (n(F)+1)^{\mathrm{hd}(F)}$.
2. $\mathrm{hd}(F)+1$ is the space complexity of tree-resolution.
3. $\mathrm{hd}(F)$ is the level of nested input-resolution needed.

[10, 11] allow also for oracles for SAT and UNSAT decisions. This will become important in future investigations, but here we only use UNSAT and the trivial oracle for it, just looking for the empty clause.

Hardness generalised

As mentioned, in [10, 11] hardness is also extended to satisfiable clause-sets, motivated by SAT-decision purposes. Here we use a stronger measurement (hard to evaluate):

- First mentioned in Ansótegui, Bonet, Levy, and Manyà [1].
- Further developed in Gwynne and Kullmann [8].
- For a clause-set F the hardness is the maximum of $\mathrm{hd}(\varphi * F)$ for such partial assignments φ where $\varphi * F$ is unsatisfiable.
- I.e., the maximum of $\mathrm{hd}(\langle x \to 0 : x \in C\rangle * F)$ over $F \models C$.
- This is the maximum level of **nested input resolution** needed to derive all $F \models C$.

TOC: UC via hardness

1. $\mathcal{UC}_1 = \mathcal{UC}$
2. $\mathcal{UC}_k$
3. $\mathcal{SLUR}$
4. $\mathcal{SLUR}_k$

Recognising UC

We have

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

This is just the well-known refutational equivalence of unit-resolution and input-resolution.

- So we can generalise!
- As explained in [10, 11], for higher levels we have to stick to input-resolution, which "parameterises" tree-resolution, while generalised unit-resolution "parameterises" dag-resolution.

(Regarding the already mentioned possibility of considering dag-resolution, it might be interesting in the future to actually consider generalised unit-resolution (see [10, 11]) and the hardness whd based on it.)

The generalised UC hierarchy

For $k \in \mathbb{N}_0$:

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

- $\mathcal{UC}_0$ is the set of clause-sets F which contain all their prime implicates (i.e., $\mathrm{prc}_0(F) \subseteq F$).
- $\mathcal{UC}_1 = \mathcal{UC}$.
- Membership decision for $\mathcal{UC}_0$ is polynomial-time, while it is coNP-complete for $k \geq 1$.

The fundamental lemma:

$F \in \mathcal{UC}_k$ if and only if for every $C \in \mathrm{prc}_0(F)$ there is a derivation of C from F via k-times nested input resolution.

Q4: A proper representation hierarchies

Conjecture

For every $k \in \mathbb{N}$ there are sequences of boolean functions with good representations of hardness ("softness") k, while there is no such good representation for $k-1$.

It is rather easy to show

$$\mathcal{PC}_0 \subset \mathcal{UC}_0 \subset \mathcal{PC}_1 \subset \mathcal{UC}_1 \subset \mathcal{PC}_2 \subset \ldots$$

The strengthened conjecture is that we have a proper representation hierarchy regarding this refined, intertwined levels.

SLUR

In Schlipf, Annexstein, Franco, and Swaminathan [12] the SLUR-algorithm was introduced:

1. "Single lookahead unit resolution".
2. Input $F \in \mathcal{CLS}$, output "SAT", "UNSAT" or "don't know".
3. If $\mathrm{r}_1(F) = \{\bot\}$, then output "UNSAT".
4. Now only outputs "SAT" and "don't know" are possible, in the following loop:
   1. If $F = \top$, then output "SAT".
   2. Otherwise consider a literal x.
   3. If $\mathrm{r}_1(\langle x \to 1\rangle * F) = \{\bot\}$, then simplify $F \rightsquigarrow F\langle x \to 0\rangle * F$.
   4. Otherwise simplify $F \rightsquigarrow F\langle x \to 1\rangle * F$.
   5. If $F = \{\bot\}$, then output "don't know".
5. $\mathcal{SLUR}$ is the class of clause-sets F where SLUR never outputs "don't know".

SLUR properly turned into a hierarchy

To obtain the **right** generalisation of $\mathcal{SLUR}$,

just replace r_1 by r_k for $k \in \mathbb{N}_0$!

See [8] why this is better than the previous (three) approaches in Čepek, Kučera, and Vlček [6], Balyo, Štefan Gurský, Kučera, and Vlček [2].

It all boils down to the fundamental lemma.

$\mathcal{UC}_k = \mathcal{SLUR}_k$

Once everything is in place, then it's not hard to show that

$$\mathcal{UC}_k = \mathcal{SLUR}_k$$

for all $k \in \mathbb{N}_0$.

- So for $\mathcal{UC}_k$ a simple SAT-decision algorithm is available, which outputs also a satisfying assignment (by a form of self-reduction).
- And this simple SAT-decision algorithm also characterises $\mathcal{UC}_k$.

Hardness — to be softened, not measured

Ansótegui et al. [1] proposed "hardness" (for unsatisfiable clause-sets) as measure of solver-hardness.

- Let's call a clause-set *k-soft* if $\mathrm{hd}(F) \leq k$.
- We think of the main role of hardness for *satisfiable* clause-sets as being a **target**, to construct soft clause-sets (representing relevant boolean functions).

Using Čepek et al. [6] (where SLUR is handled) we get:

Lemma

For $k \geq 1$ the decision $\mathrm{hd}(F) \leq k$ is coNP-complete.

(Recall that for unsatisfiable F computation of $\mathrm{hd}(F)$ can be done in time $O(n^{2\,\mathrm{hd}(F)})$.)

"Hardness" does not measure solution-hardness,
but **representation-hardness**.

Algorithmically finding good representations

The theory outlined allows many possibilities for constructing good representations, algorithmically or schematically.

Conclusions

Key points:

- Framework for translating CSP into SAT (first encoding, then representation).
- Search for (useful) wild encodings!
- Introduced the "absolute criterion".
- Hardness (and p-hardness) measure representation-goodness.

Next steps:

- Show that many interesting examples, which have good representations fulfilling the relative condition, actually can be made fulfilling the absolute conditions.
- Prove the various separation conjectures.
- The theory can be generalised by using oracles: $\mathrm{hd}_{\mathcal{U}}$ (this allows for example to represent all-different constraints) — use it!
- The theory can also be extended to consider dag-resolution, by $\mathrm{whd}_{\mathcal{U}}$ (instead of tree-resolution) — develop this theory.

End

(references on the remaining slides).

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] Tomáš Balyo, Štefan Gurský, Petr Kučera, and Václav Vlček. On hierarchies over the SLUR class. International Symposium on Artificial Intelligence and Mathematics, January 2012. http://www.cs.uic.edu/bin/view/Isaim2012/AcceptedPapers.

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

Bibliography II

[4] 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.

[5] Lucas Bordeaux, Mikoláš Janota, Joao Marques-Silva, and Pierre Marquis. On unit-refutation complete formulae with existentially quantified variables. In *Knowledge Representation 2012 (KR 2012)*, 2012.

[6] Ondřej Čepek, Petr Kučera, and Václav Vlček. Properties of SLUR formulae. In Mária Bieliková, Gerhard Friedrich, Georg Gottlob, Stefan Katzenbeisser, and György Turán, editors, *SOFSEM 2012: Theory and Practice of Computer Science*, volume 7147 of *LNCS Lecture Notes in Computer Science*, pages 177–189. Springer, 2012.

Bibliography III

[7] 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.

[8] Matthew Gwynne and Oliver Kullmann. Generalising unit-refutation completeness and SLUR via nested input resolution. Technical Report arXiv:1204.6529v2 [cs.LO], arXiv, May 2012.

[9] Stasys Jukna. *Boolean Function Complexity: Advances and Frontiers*, volume 27 of *Algorithms and Combinatorics*. Springer, 2012. ISBN 978-3-642-24507-7.

[10] 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.

Bibliography IV

[11] 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.

[12] John S. Schlipf, Fred S. Annexstein, John V. Franco, and R.P. Swaminathan. On finding solutions for extended Horn formulas. *Information Processing Letters*, 54:133–137, 1995.