Pseudo-Boolean Constraints: Reasoning and Compilation

Romain Wallon *(Advisors: Daniel Le Berre, Pierre Marquis, Stefan Mengel)*

September 11, 2017

CRIL - U. Artois & CNRS

Overview

Reasoning with Pseudo-Boolean Constraints

The usual resolution approach...

SAT solvers deal with information represented as propositional formulae, in conjunctive normal form (CNF)

The usual resolution approach...

SAT solvers deal with information represented as propositional formulae, in conjunctive normal form (CNF)

$$(a \vee b \vee \neg c) \wedge (a \vee \neg b \vee d)$$

The usual resolution approach...

SAT solvers deal with information represented as propositional formulae, in conjunctive normal form (CNF)

$$(a \vee b \vee \neg c) \wedge (a \vee \neg b \vee d)$$

To reason on such formulae, the resolution proof system can be used

$$\frac{x \vee \phi \qquad \neg x \vee \psi}{\phi \vee \psi}\ \text{(resolution)} \qquad\qquad \frac{l \vee l \vee \phi}{l \vee \phi}\ \text{(fusion)}$$

The usual resolution approach...

SAT solvers deal with information represented as propositional formulae, in conjunctive normal form (CNF)

$$(a \vee b \vee \neg c) \wedge (a \vee \neg b \vee d)$$

To reason on such formulae, the resolution proof system can be used

$$\frac{x \vee \phi \qquad \neg x \vee \psi}{\phi \vee \psi} \text{ (resolution)} \qquad\qquad \frac{l \vee l \vee \phi}{l \vee \phi} \text{ (fusion)}$$

When the formula is UNSAT, this proof system is used to find a proof of $\perp$

…is not efficient on some problems!

Definition (Pigeon-Hole Principle – PHP)

You cannot put p pigeons in $p-1$ holes!

...is not efficient on some problems!

Definition (Pigeon-Hole Principle – PHP)

You cannot put p pigeons in $p-1$ holes!

Example

Let us consider:

- p pigeons and h holes
- $x_{i,j}$ meaning that pigeon i is put in hole j

...is not efficient on some problems!

Definition (Pigeon-Hole Principle – PHP)

You cannot put p pigeons in $p-1$ holes!

Example

Let us consider:

- p pigeons and h holes
- $x_{i,j}$ meaning that pigeon i is put in hole j

The encoding is based on the following assertions:

Each pigeon is assigned at least one hole

and Each hole contains at most one pigeon

Definition (Pigeon-Hole Principle – PHP)

You cannot put p pigeons in $p-1$ holes!

Example

Let us consider:

- p pigeons and h holes
- $x_{i,j}$ meaning that pigeon i is put in hole j

A CNF encoding is:

$$\bigwedge_{i=1}^{p} \bigvee_{j=1}^{h} x_{i,j} \wedge \bigwedge_{i=1}^{p-1} \bigwedge_{j=i+1}^{p} \bigwedge_{k=1}^{h} (\neg x_{i,k} \vee \neg x_{j,k})$$

...is not efficient on some problems!

Definition (Pigeon-Hole Principle – PHP)

You cannot put p pigeons in $p-1$ holes!

Example

Let us consider:

- p pigeons and h holes
- $x_{i,j}$ meaning that pigeon i is put in hole j

A CNF encoding is:

$$\bigwedge_{i=1}^{p} \bigvee_{j=1}^{h} x_{i,j} \wedge \bigwedge_{i=1}^{p-1} \bigwedge_{j=i+1}^{p} \bigwedge_{k=1}^{h} (\neg x_{i,k} \vee \neg x_{j,k})$$

When $h < p$, an exponential number of resolution steps is required to prove unsatisfiability

Linear Pseudo-Boolean Constraints

Linear Pseudo-Boolean Constraints

A linear pseudo-Boolean constraint is of the form:

$$\sum_j a_j l_j \rhd k$$

where:

- $\forall j, a_j \in \mathbb{Z}$
- $\forall j, l_j$ is a literal (i.e. a boolean value)
- $\rhd \in \{<, \leqslant, =, \geqslant, >\}$
- $k \in \mathbb{Z}$ is the degree (threshold) of the constraint

PBC and CARD

We focus on two kinds of constraints

PBC and CARD

We focus on two kinds of constraints

Normalized pseudo-Boolean constraints are of the form:

$$\sum_j a_j l_j \geqslant k \quad \forall j, a_j \in \mathbb{N}, k \in \mathbb{N}$$

PBC and CARD

We focus on two kinds of constraints

Normalized pseudo-Boolean constraints are of the form:

$$\sum_j a_j l_j \geqslant k \quad \forall j, a_j \in \mathbb{N}, k \in \mathbb{N}$$

Cardinality constraints are of the form:

$$\sum_j l_j \geqslant k \quad k \in \mathbb{N}$$

PBC and CARD

We focus on two kinds of constraints

Normalized pseudo-Boolean constraints are of the form:

$$\sum_j a_j l_j \geqslant k \quad \forall j, a_j \in \mathbb{N}, k \in \mathbb{N}$$

Cardinality constraints are of the form:

$$\sum_j l_j \geqslant k \quad k \in \mathbb{N}$$

A formula of PBC (resp. CARD) is a conjunction of normalized constraints (resp. cardinality constraints)

Generalized Resolution

The proof system used to reason on PBC and CARD formulas is the generalized resolution proof system, which is more powerful than the resolution one [Hooker, 1988]

$$\frac{\alpha l + \sum_j a_j l_j \geqslant k \qquad \beta \bar{l} + \sum_j b_j l_j \geqslant k' \qquad \alpha \in \mathbb{N}^* \qquad \beta \in \mathbb{N}^*}{\sum_j (\beta a_j + \alpha b_j) l_j \geqslant \alpha k' + \beta k - \alpha\beta} \text{ (cancellation)}$$

$$\frac{\sum_j a_j l_j \geqslant k \qquad \forall j, a_j \geqslant 0 \qquad a_i > k}{k l_i + \sum_{j \neq i} a_j l_j \geqslant k} \text{ (saturation)}$$

Is it worth the effort?

The PBC encoding of PHP is:

$$\bigwedge_{i=1}^{p} atLeast(\{x_{i,1}, \ldots, x_{i,h}\}, 1) \wedge \bigwedge_{i=1}^{h} atMost(\{x_{1,j}, \ldots, x_{p,j}\}, 1)$$

Is it worth the effort?

The PBC encoding of PHP is:

$$\bigwedge_{i=1}^{p} atLeast(\{x_{i,1}, \ldots, x_{i,h}\}, 1) \wedge \bigwedge_{i=1}^{h} atLeast(\{\overline{x_{1,j}}, \ldots, \overline{x_{p,j}}\}, p\text{-}1)$$

Is it worth the effort?

The PBC encoding of PHP is:

$$\bigwedge_{i=1}^{p} \left(\sum_{j=1}^{h} x_{i,j} \geqslant 1 \right) \wedge \bigwedge_{j=1}^{h} \left(\sum_{i=1}^{p} \overline{x_{i,j}} \geqslant p - 1 \right)$$

Is it worth the effort?

The PBC encoding of PHP is:

$$\bigwedge_{i=1}^{p}\left(\sum_{j=1}^{h} x_{i,j} \geqslant 1\right) \wedge \bigwedge_{j=1}^{h}\left(\sum_{i=1}^{p} \overline{x_{i,j}} \geqslant p-1\right)$$

By using this encoding, one can solve a PHP instance in a linear number of steps [Haken, 1985 & Hooker, 1988]

From CARD to CNF

Let us consider the following cardinality constraint:

$$a + b + c + d + e \geqslant 3$$

From CARD to CNF

Let us consider the following cardinality constraint:

$$a + b + c + d + e \geqslant 3$$

Its CNF encoding is given below:

$$\begin{aligned}
&(a \vee b \vee c) \wedge (a \vee b \vee d) \wedge (a \vee b \vee e) \wedge (a \vee c \vee d) \wedge (a \vee c \vee e) \\
\wedge &(a \vee d \vee e) \wedge (b \vee c \vee d) \wedge (b \vee c \vee e) \wedge (b \vee d \vee e) \wedge (c \vee d \vee e)
\end{aligned}$$

From CARD to CNF

Let us consider the following cardinality constraint:

$$a + b + c + d + e \geqslant 3$$

Its CNF encoding is given below:

$$\begin{aligned}&(a \vee b \vee c) \wedge (a \vee b \vee d) \wedge (a \vee b \vee e) \wedge (a \vee c \vee d) \wedge (a \vee c \vee e)\\ \wedge&(a \vee d \vee e) \wedge (b \vee c \vee d) \wedge (b \vee c \vee e) \wedge (b \vee d \vee e) \wedge (c \vee d \vee e)\end{aligned}$$

This CNF encoding is the smallest which does not require to introduce new variables [Dixon, 2004]

Representing knowledge using PBC and CARD

Let us recap what we have seen

- pseudo-Boolean constraints enable to improve reasoning efficiency in some cases
- representing a problem in this language requires less space than CNF

Representing knowledge using PBC and CARD

Let us recap what we have seen

- pseudo-Boolean constraints enable to improve reasoning efficiency in some cases
- representing a problem in this language requires less space than CNF

With PBC or CARD, modeling problems is also more natural: subset-sum and knapsack require two normalized pseudo-Boolean constraints to be modeled

Representing knowledge using PBC and CARD

Let us recap what we have seen

- pseudo-Boolean constraints enable to improve reasoning efficiency in some cases
- representing a problem in this language requires less space than CNF

With PBC or CARD, modeling problems is also more natural: subset-sum and knapsack require two normalized pseudo-Boolean constraints to be modeled

Let us consider PBC and CARD as knowledge representation languages

A Knowledge Compilation Map

Knowledge compilation

Given a formula written in a specific language (e.g. CNF, DNF, etc.), one would like to perform operations on it

But sometimes they are too expensive to be performed

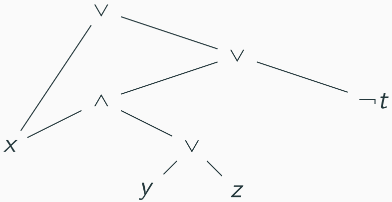
Knowledge compilation

Given a formula written in a specific language (e.g. CNF, DNF, etc.), one would like to perform operations on it

But sometimes they are too expensive to be performed

Compiling a formula is translating it into an other language to obtain an equivalent formula on which performing the wanted operations is easier

A circuit in Negative Normal Form is a DAG like this one:



Some compilation languages: $OBDD_<$

Let us consider $\phi = x \vee (y \wedge x) \vee (z \wedge x) \vee \neg t$

Some compilation languages: $OBDD_<$

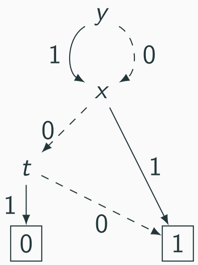
Let us consider $\phi = x \vee (y \wedge x) \vee (z \wedge x) \vee \neg t$

Given the order over the variables $y < x < t < z$,

Some compilation languages: $OBDD_<$

Let us consider $\phi = x \vee (y \wedge x) \vee (z \wedge x) \vee \neg t$

Given the order over the variables $y < x < t < z$, the Ordered Binary Decision Diagram representing ϕ, written $OBDD_<(\phi)$, is:



Some compilation languages: *IP*, *PI* et *MODS*

Let us consider $\phi = x \vee (y \wedge x) \vee (z \wedge x) \vee \neg t$

Some compilation languages: *IP*, *PI* et *MODS*

Let us consider $\phi = x \vee (y \wedge x) \vee (z \wedge x) \vee \neg t$

$$IP(\phi) = (x) \vee (\neg t)$$

Let us consider $\phi = x \vee (y \wedge x) \vee (z \wedge x) \vee \neg t$

$$IP(\phi) = (x) \vee (\neg t)$$

$$PI(\phi) = x \vee \neg t$$

Some compilation languages: *IP*, *PI* et *MODS*

Let us consider $\phi = x \vee (y \wedge x) \vee (z \wedge x) \vee \neg t$

$$IP(\phi) = (x) \vee (\neg t)$$

$$PI(\phi) = x \vee \neg t$$

$$\begin{aligned} MODS(\phi) = \quad & (x \wedge y \wedge z \wedge t) \vee & (x \wedge y \wedge z \wedge \neg t) \vee \\ & (x \wedge y \wedge \neg z \wedge t) \vee & (x \wedge y \wedge \neg z \wedge \neg t) \vee \\ & (x \wedge \neg y \wedge z \wedge t) \vee & (x \wedge \neg y \wedge z \wedge \neg t) \vee \\ & (x \wedge \neg y \wedge \neg z \wedge t) \vee & (x \wedge \neg y \wedge \neg z \wedge \neg t) \vee \\ & (\neg x \wedge y \wedge z \wedge \neg t) \vee & (\neg x \wedge y \wedge \neg z \wedge \neg t) \vee \\ & (\neg x \wedge \neg y \wedge z \wedge \neg t) \vee & (\neg x \wedge \neg y \wedge \neg z \wedge \neg t) \end{aligned}$$

A map to compare them all

To compare all these languages, Adnan Darwiche and Pierre Marquis proposed in 2002 a knowledge compilation map [DM02]

A map to compare them all

To compare all these languages, Adnan Darwiche and Pierre Marquis proposed in 2002 a knowledge compilation map [DM02]

Three criteria are taken into account to identify which language is the best to use w.r.t. the wanted operations

A map to compare them all

To compare all these languages, Adnan Darwiche and Pierre Marquis proposed in 2002 a knowledge compilation map [DM02]

Three criteria are taken into account to identify which language is the best to use w.r.t. the wanted operations

- succinctness

A map to compare them all

To compare all these languages, Adnan Darwiche and Pierre Marquis proposed in 2002 a knowledge compilation map [DM02]

Three criteria are taken into account to identify which language is the best to use w.r.t. the wanted operations

- succinctness
- queries

A map to compare them all

To compare all these languages, Adnan Darwiche and Pierre Marquis proposed in 2002 a knowledge compilation map [DM02]

Three criteria are taken into account to identify which language is the best to use w.r.t. the wanted operations

- succinctness
- queries
- transformations

Succinctness [DM02]

Succinctness captures the ability of a language to represent information using little space

Succinctness [DM02]

Succinctness captures the ability of a language to represent information using little space

L_1 is at least as succinct as L_2, denoted $L_1 \leqslant L_2$, iff there exists a polynomial p such that for every formula $\alpha \in L_2$, there exists an equivalent formula β where $|\beta| \leqslant p(|\alpha|)$

Succinctness [DM02]

Succinctness captures the ability of a language to represent information using little space

L_1 is at least as succinct as L_2, denoted $L_1 \leqslant L_2$, iff there exists a polynomial p such that for every formula $\alpha \in L_2$, there exists an equivalent formula β where $|\beta| \leqslant p(|\alpha|)$

In other words, $L_1 \leqslant L_2$ iff any formula $\alpha \in L_2$ can be written as a formula $\beta \in L_1$ of polynomial size

Succinctness [DM02]

Succinctness captures the ability of a language to represent information using little space

L_1 is at least as succinct as L_2, denoted $L_1 \leqslant L_2$, iff there exists a polynomial p such that for every formula $\alpha \in L_2$, there exists an equivalent formula β where $|\beta| \leqslant p(|\alpha|)$

In other words, $L_1 \leqslant L_2$ iff any formula $\alpha \in L_2$ can be written as a formula $\beta \in L_1$ of polynomial size

Note that there is no hypothesis on the time complexity of the algorithm needed to translate a formula from L_2 to L_1

Results from the KC map (succinctness)

Results from [DM02], [Bova-Capelli-Mengel-Slivovsky, 2016] and [Kaleyski, 2017]

	NNF	*DNNF*	*d − DNNF*	*sd − DNNF*	*FBDD*	*OBDD*	$OBDD_<$	*DNF*	*CNF*	*PI*	*IP*	*MODS*
NNF	$\leqslant$	$\leqslant$	$\leqslant$	$\leqslant$	$\leqslant$	$\leqslant$	$\leqslant$	$\leqslant$	$\leqslant$	$\leqslant$	$\leqslant$	$\leqslant$
DNNF	$\nleqslant$	$\leqslant$	$\leqslant$	$\leqslant$	$\leqslant$	$\leqslant$	$\leqslant$	$\leqslant$	$\nleqslant$	$\nleqslant$	$\leqslant$	$\leqslant$
d − DNNF	$\nleqslant$	$\nleqslant$	$\leqslant$	$\leqslant$	$\leqslant$	$\leqslant$	$\leqslant$	$\nleqslant^*$	$\nleqslant$	$\nleqslant$	?	$\leqslant$
sd − DNNF	$\nleqslant$	$\nleqslant$	$\leqslant$	$\leqslant$	$\leqslant$	$\leqslant$	$\leqslant$	$\nleqslant$	$\nleqslant$	$\nleqslant$	$\nleqslant$	$\leqslant$
FBDD	$\nleqslant$	$\nleqslant$	$\nleqslant$	$\nleqslant$	$\leqslant$	$\leqslant$	$\leqslant$	$\nleqslant$	$\nleqslant$	$\nleqslant$	$\nleqslant$	$\leqslant$
OBDD	$\nleqslant$	$\nleqslant$	$\nleqslant$	$\nleqslant$	$\nleqslant$	$\leqslant$	$\leqslant$	$\nleqslant$	$\nleqslant$	$\nleqslant$	$\nleqslant$	$\leqslant$
$OBDD_<$	$\nleqslant$	$\nleqslant$	$\nleqslant$	$\nleqslant$	$\nleqslant$	$\nleqslant$	$\leqslant$	$\nleqslant$	$\nleqslant$	$\nleqslant$	$\nleqslant$	$\leqslant$
DNF	$\nleqslant$	$\nleqslant$	$\nleqslant$	$\nleqslant$	$\nleqslant$	$\nleqslant$	$\nleqslant$	$\leqslant$	$\nleqslant$	$\nleqslant$	$\leqslant$	$\leqslant$
CNF	$\nleqslant$	$\nleqslant$	$\nleqslant$	$\nleqslant$	$\nleqslant$	$\nleqslant$	$\nleqslant$	$\nleqslant$	$\leqslant$	$\leqslant$	$\nleqslant$	$\leqslant$
PI	$\nleqslant$	$\nleqslant$	$\nleqslant$	$\nleqslant$	$\nleqslant$	$\nleqslant$	$\nleqslant$	$\nleqslant$	$\nleqslant$	$\leqslant$	$\nleqslant$	$\nleqslant$ (?)
IP	$\nleqslant$	$\nleqslant$	$\nleqslant$	$\nleqslant$	$\nleqslant$	$\nleqslant$	$\nleqslant$	$\nleqslant$	$\nleqslant$	$\nleqslant$	$\leqslant$	$\leqslant$
MODS	$\nleqslant$	$\nleqslant$	$\nleqslant$	$\nleqslant$	$\nleqslant$	$\nleqslant$	$\nleqslant$	$\nleqslant$	$\nleqslant$	$\nleqslant$	$\nleqslant$	$\leqslant$

Queries [DM02]

Given one or several formulas, what are the properties of these formulas?

Queries [DM02]

Given one or several formulas, what are the properties of these formulas?

CO (COnsistency) Is a formula consistent?

VA (VAlidity) Is a formula valid?

CE (Clausal Entailment) Is a given clause implied by a formula?

IM (IMplication) Is a formula implied by a given cube/term?

EQ (EQuivalence) Are two formulas equivalent?

SE (Sentential Entailment) Is a formula entailed by an other one?

Queries [DM02]

Given one or several formulas, what are the properties of these formulas?

CO (COnsistency) Is a formula consistent?

VA (VAlidity) Is a formula valid?

CE (Clausal Entailment) Is a given clause implied by a formula?

IM (IMplication) Is a formula implied by a given cube/term?

EQ (EQuivalence) Are two formulas equivalent?

SE (Sentential Entailment) Is a formula entailed by an other one?

CT (CounTing) How many models does a formula have?

Queries [DM02]

Given one or several formulas, what are the properties of these formulas?

CO (COnsistency) Is a formula consistent?

VA (VAlidity) Is a formula valid?

CE (Clausal Entailment) Is a given clause implied by a formula?

IM (IMplication) Is a formula implied by a given cube/term?

EQ (EQuivalence) Are two formulas equivalent?

SE (Sentential Entailment) Is a formula entailed by an other one?

CT (CounTing) How many models does a formula have?

ME (Model Enumeration) What are all the models of a formula?

Results from the KC map (queries) [DM02]

$\mathcal{L}$	CO	VA	CE	IM	EQ	SE	CT	ME
NNF	∘	∘	∘	∘	∘	∘	∘	∘
$DNNF$	✓	∘	✓	∘	∘	∘	∘	✓
$d - DNNF$	✓	✓	✓	✓	?	∘	✓	✓
$sd - DNNF$	✓	✓	✓	✓	?	∘	✓	✓
BDD	∘	∘	∘	∘	∘	∘	∘	∘
$FBDD$	✓	✓	✓	✓	?	∘	✓	✓
$OBDD$	✓	✓	✓	✓	✓	∘	✓	✓
$OBDD_<$	✓	✓	✓	✓	✓	✓	✓	✓
DNF	✓	∘	✓	∘	∘	∘	∘	✓
CNF	∘	✓	∘	✓	∘	∘	∘	∘
PI	✓	✓	✓	✓	✓	✓	∘	✓
IP	✓	✓	✓	✓	✓	✓	∘	✓
$MODS$	✓	✓	✓	✓	✓	✓	✓	✓

✓ Verified ∘ Not verified (unless P = NP)

Transformations [DM02]

Given one or several formulas, transform them into a formula equivalent in the considered language to the application of a logical operator

Transformations [DM02]

Given one or several formulas, transform them into a formula equivalent in the considered language to the application of a logical operator

CD (ConDitioning) Compute $\phi | \tau$ where τ is a consistent cube/term

SFO (Singleton FOrgetting) Compute $\exists x . \phi \equiv (\phi | x) \vee (\phi | \overline{x})$

FO (FOrgetting) Compute $\exists X . \phi$ where X is a set of variables

$\wedge$C (Closure under $\wedge$) Compute $\bigwedge_{i=1}^{n} \phi_i$

$\wedge$BC (Bounded Closure under $\wedge$) Compute $\bigwedge_{i=1}^{n} \phi_i$, where $n \leqslant N$

$\vee$C (Closure under $\vee$) Compute $\bigvee_{i=1}^{n} \phi_i$

$\vee$BC (Bounded Closure under $\vee$) Compute $\bigvee_{i=1}^{n} \phi_i$, where $n \leqslant N$

$\neg$C (Closure under $\neg$) Compute $\neg \phi$

Results from the KC map (transformations) [DM02]

$\mathcal{L}$	CD	FO	SFO	$\wedge C$	$\wedge BC$	$\vee C$	$\vee BC$	$\neg C$
NNF	✓	○	✓	✓	✓	✓	✓	✓
$DNNF$	✓	✓	✓	○	○	✓	✓	○
$d-DNNF$	✓	○	○	○	○	○	○	?
$sd-DNNF$	✓	○	○	○	○	○	○	?
BDD	✓	○	✓	✓	✓	✓	✓	✓
$FBDD$	✓	•	○	•	○	•	○	✓
$OBDD$	✓	•	✓	•	○	•	○	✓
$OBDD_<$	✓	•	✓	•	✓	•	✓	✓
DNF	✓	✓	✓	•	✓	✓	✓	•
CNF	✓	○	✓	✓	✓	•	✓	•
PI	✓	✓	✓	•	•	•	✓	•
IP	✓	•	•	•	✓	•	•	•
$MODS$	✓	✓	✓	•	✓	•	•	•

✓ Verified ○ Not verified (unless P = NP) • Not verified

Properties of pseudo-Boolean constraints

Some interesting (but hard) problems on a single constraint

	CO	VA	CE	IM	EQ	SE	CT	ME
1-CARD	?	?	?	?	?	?	?	?
1-PBC	?	?	?	?	?	?	?	?

Some interesting (but hard) problems on a single constraint

	CO	VA	CE	IM	EQ	SE	CT	ME
1-CARD	?	?	?	?	?	?	?	?
1-PBC	?	?	?	?	?	?	?	?

Increasible degree:

$$9w + 6x + 3y + z \geqslant 11$$

Some interesting (but hard) problems on a single constraint

	CO	VA	CE	IM	EQ	SE	CT	ME
1-CARD	?	?	?	?	?	?	?	?
1-PBC	?	?	?	?	?	?	?	?

Increasible degree:

$$9w + 6x + 3y + z \geqslant 11$$

Some interesting (but hard) problems on a single constraint

	CO	VA	CE	IM	EQ	SE	CT	ME
1-CARD	?	?	?	?	?	?	?	?
1-PBC	?	?	?	?	?	?	?	?

Increasible degree:

$$9w + 6x + 3y + z \geqslant 11 \equiv 9w + 6x + 3y + z \geqslant 12$$

Some interesting (but hard) problems on a single constraint

	CO	VA	CE	IM	EQ	SE	CT	ME
1-CARD	?	?	?	?	?	?	?	?
1-PBC	?	?	?	?	?	?	?	?

Increasible degree: (coNP-hard: reduction from $\overline{\text{subset-sum}}$)

$$9w + 6x + 3y + z \geqslant 11 \equiv 9w + 6x + 3y + z \geqslant 12$$

Some interesting (but hard) problems on a single constraint

	CO	VA	CE	IM	EQ	SE	CT	ME
1-CARD	?	?	?	?	?	?	?	?
1-PBC	?	?	?	?	?	?	?	?

Increasible degree: (coNP-hard: reduction from $\overline{\text{subset-sum}}$)

$$9w + 6x + 3y + z \geqslant 11 \equiv 9w + 6x + 3y + z \geqslant 12$$

Dependency on a variable:

$$9w + 6x + 3y + z \geqslant 11$$

Some interesting (but hard) problems on a single constraint

	CO	VA	CE	IM	EQ	SE	CT	ME
1-CARD	?	?	?	?	?	?	?	?
1-PBC	?	?	?	?	?	?	?	?

Increasible degree: (coNP-hard: reduction from $\overline{\text{subset-sum}}$)

$$9w + 6x + 3y + z \geqslant 11 \equiv 9w + 6x + 3y + z \geqslant 12$$

Dependency on a variable:

$$9w + 6x + 3y + z \geqslant 11$$

Some interesting (but hard) problems on a single constraint

	CO	VA	CE	IM	EQ	SE	CT	ME
1-CARD	?	?	?	?	?	?	?	?
1-PBC	?	?	?	?	?	?	?	?

Increasible degree: (coNP-hard: reduction from $\overline{\text{subset-sum}}$)

$$9w + 6x + 3y + z \geqslant 11 \equiv 9w + 6x + 3y + z \geqslant 12$$

Dependency on a variable:

$$9w + 6x + 3y + z \geqslant 11 \equiv 9w + 6x + 3y \geqslant 11$$

Some interesting (but hard) problems on a single constraint

	CO	VA	CE	IM	EQ	SE	CT	ME
1-CARD	?	?	?	?	?	?	?	?
1-PBC	?	?	?	?	?	?	?	?

Increasible degree: (coNP-hard: reduction from $\overline{\text{subset-sum}}$)

$$9w + 6x + 3y + z \geqslant 11 \equiv 9w + 6x + 3y + z \geqslant 12$$

Dependency on a variable: (NP-hard: reduction from $\overline{\text{increasible degree}}$)

$$9w + 6x + 3y + z \geqslant 11 \equiv 9w + 6x + 3y \geqslant 11$$

Querying a single pseudo-Boolean constraint

	CO	VA	CE	IM	EQ	SE	CT	ME
1-CARD	?	?	?	?	?	?	?	?
1-PBC	?	?	?	?	?	?	?	?

Querying a single pseudo-Boolean constraint

	CO	VA	CE	IM	EQ	SE	CT	ME
1-CARD	✓	?	?	?	?	?	?	?
1-PBC	✓	?	?	?	?	?	?	?

Consistency can be checked by summing the weights

$$3a + 2\overline{b} + c \geqslant 3 \qquad 3a + 2\overline{b} + c \geqslant 7$$

Querying a single pseudo-Boolean constraint

	CO	VA	CE	IM	EQ	SE	CT	ME
1-CARD	✓	?	?	?	?	?	?	?
1-PBC	✓	?	?	?	?	?	?	?

Consistency can be checked by summing the weights

$$3a + 2\overline{b} + c \geqslant 3 \quad ✓ \quad 3a + 2\overline{b} + c \geqslant 7$$

Querying a single pseudo-Boolean constraint

	CO	VA	CE	IM	EQ	SE	CT	ME
1-CARD	$\checkmark$	?	?	?	?	?	?	?
1-PBC	$\checkmark$	?	?	?	?	?	?	?

Consistency can be checked by summing the weights

$$3a + 2\overline{b} + c \geqslant 3 \quad ✓ \quad 3a + 2\overline{b} + c \geqslant 7 \quad ✗$$

Querying a single pseudo-Boolean constraint

	CO	VA	CE	IM	EQ	SE	CT	ME
1-CARD	✓	✓	?	?	?	?	?	?
1-PBC	✓	✓	?	?	?	?	?	?

Consistency can be checked by summing the weights

$$3a + 2\overline{b} + c \geqslant 3 \quad ✓ \quad 3a + 2\overline{b} + c \geqslant 7 \quad ✗$$

A normalized pseudo-Boolean constraint is valid iff its degree is 0

Querying a single pseudo-Boolean constraint

	CO	VA	CE	IM	EQ	SE	CT	ME
1-CARD	✓	✓	✓	✓	?	?	?	✓
1-PBC	✓	✓	✓	✓	?	?	?	✓

Consistency can be checked by summing the weights

$$3a + 2\overline{b} + c \geqslant 3 \quad ✓ \quad 3a + 2\overline{b} + c \geqslant 7 \quad ✗$$

A normalized pseudo-Boolean constraint is valid iff its degree is 0

Properties of pseudo-Boolean constraints give these results

Querying a single pseudo-Boolean constraint

	CO	VA	CE	IM	EQ	SE	CT	ME
1-CARD	✓	✓	✓	✓	?	✓	?	✓
1-PBC	✓	✓	✓	✓	?	?	?	✓

Consistency can be checked by summing the weights

$$3a + 2\overline{b} + c \geqslant 3 \quad ✓ \quad 3a + 2\overline{b} + c \geqslant 7 \quad ✗$$

A normalized pseudo-Boolean constraint is valid iff its degree is 0

Properties of pseudo-Boolean constraints give these results

$\sum_{l \in L} l \geqslant k \models \sum_{l' \in L'} l' \geqslant k'$ iff $k' \leqslant 0$ or $|L \setminus L'| \leqslant k - k'$

Querying a single pseudo-Boolean constraint

	CO	VA	CE	IM	EQ	SE	CT	ME
1-CARD	✓	✓	✓	✓	✓	✓	?	✓
1-PBC	✓	✓	✓	✓	?	?	?	✓

Consistency can be checked by summing the weights

$$3a + 2\overline{b} + c \geqslant 3 \quad ✓ \quad 3a + 2\overline{b} + c \geqslant 7 \quad ✗$$

A normalized pseudo-Boolean constraint is valid iff its degree is 0

Properties of pseudo-Boolean constraints give these results

$\sum_{l \in L} l \geqslant k \models \sum_{l' \in L'} l' \geqslant k'$ iff $k' \leqslant 0$ or $|L \backslash L'| \leqslant k - k'$

Querying a single pseudo-Boolean constraint

	CO	VA	CE	IM	EQ	SE	CT	ME
1-CARD	✓	✓	✓	✓	✓	✓	?	✓
1-PBC	✓	✓	✓	✓	○	?	?	✓

Consistency can be checked by summing the weights

$$3a + 2\overline{b} + c \geqslant 3 \quad ✓ \quad 3a + 2\overline{b} + c \geqslant 7 \quad ✗$$

A normalized pseudo-Boolean constraint is valid iff its degree is 0

Properties of pseudo-Boolean constraints give these results

$\sum_{l \in L} l \geqslant k \models \sum_{l' \in L'} l' \geqslant k'$ iff $k' \leqslant 0$ or $|L \backslash L'| \leqslant k - k'$

Reduction from increasible degree

Querying a single pseudo-Boolean constraint

	CO	VA	CE	IM	EQ	SE	CT	ME
1-CARD	✓	✓	✓	✓	✓	✓	?	✓
1-PBC	✓	✓	✓	✓	○	○	?	✓

Consistency can be checked by summing the weights

$$3a + 2\overline{b} + c \geqslant 3 \quad ✓ \quad 3a + 2\overline{b} + c \geqslant 7 \quad ✗$$

A normalized pseudo-Boolean constraint is valid iff its degree is 0

Properties of pseudo-Boolean constraints give these results

$\sum_{l \in L} l \geqslant k \models \sum_{l' \in L'} l' \geqslant k'$ iff $k' \leqslant 0$ or $|L \setminus L'| \leqslant k - k'$

Reduction from increasible degree

Querying a single pseudo-Boolean constraint

	CO	VA	CE	IM	EQ	SE	CT	ME
1-CARD	✓	✓	✓	✓	✓	✓	✓	✓
1-PBC	✓	✓	✓	✓	○	○	?	✓

Consistency can be checked by summing the weights

$$3a + 2\overline{b} + c \geqslant 3 \quad ✓ \quad 3a + 2\overline{b} + c \geqslant 7 \quad ✗$$

A normalized pseudo-Boolean constraint is valid iff its degree is 0

Properties of pseudo-Boolean constraints give these results

$\sum_{l \in L} l \geqslant k \models \sum_{l' \in L'} l' \geqslant k'$ iff $k' \leqslant 0$ or $|L \backslash L'| \leqslant k - k'$

Reduction from increasible degree

There are $\sum_{j=k}^{n} \binom{n}{j}$ models of a cardinality constraint $\sum_{i=1}^{n} l_j \geqslant n$

Querying a single pseudo-Boolean constraint

	CO	VA	CE	IM	EQ	SE	CT	ME
1-CARD	✓	✓	✓	✓	✓	✓	✓	✓
1-PBC	✓	✓	✓	✓	○	○	○	✓

Consistency can be checked by summing the weights

$$3a + 2\overline{b} + c \geqslant 3 \quad ✓ \quad 3a + 2\overline{b} + c \geqslant 7 \quad ✗$$

A normalized pseudo-Boolean constraint is valid iff its degree is 0

Properties of pseudo-Boolean constraints give these results

$\sum_{l \in L} l \geqslant k \models \sum_{l' \in L'} l' \geqslant k'$ iff $k' \leqslant 0$ or $|L \setminus L'| \leqslant k - k'$

Reduction from increasible degree

There are $\sum_{j=k}^{n} \binom{n}{j}$ models of a cardinality constraint $\sum_{i=1}^{n} l_j \geqslant n$

Reduction from subset-sum

Transforming a single pseudo-Boolean constraint

	CD	FO	SFO	$\wedge$C	$\wedge$BC	$\vee$C	$\vee$BC	$\neg$C
1-CARD	?	?	?	?	?	?	?	?
1-PBC	?	?	?	?	?	?	?	?

Transforming a single pseudo-Boolean constraint

	CD	FO	SFO	$\wedge$C	$\wedge$BC	$\vee$C	$\vee$BC	$\neg$C
1-CARD	✓	?	?	?	?	?	?	?
1-PBC	✓	?	?	?	?	?	?	?

Conditioning is just replacing a variable by 0 or 1

Transforming a single pseudo-Boolean constraint

	CD	FO	SFO	$\wedge$C	$\wedge$BC	$\vee$C	$\vee$BC	$\neg$C
1-CARD	✓	?	✓	?	?	?	?	?
1-PBC	✓	?	✓	?	?	?	?	?

Conditioning is just replacing a variable by 0 or 1

Forgetting one variable can be computed in polytime:

$$\exists x. \left(ax + \sum_{j=0}^{n} a_j l_j \geqslant k \right) \equiv \left(\sum_{j=0}^{n} a_j l_j \geqslant k - a \right) \vee \left(\sum_{j=0}^{n} a_j l_j \geqslant k \right)$$

Transforming a single pseudo-Boolean constraint

	CD	FO	SFO	$\wedge$C	$\wedge$BC	$\vee$C	$\vee$BC	$\neg$C
1-CARD	✓	?	✓	?	?	?	?	?
1-PBC	✓	?	✓	?	?	?	?	?

Conditioning is just replacing a variable by 0 or 1

Forgetting one variable can be computed in polytime:

$$\exists x. \left(ax + \sum_{j=0}^{n} a_j l_j \geqslant k \right) \equiv \left(\sum_{j=0}^{n} a_j l_j \geqslant k - a \right)$$

Transforming a single pseudo-Boolean constraint

	CD	FO	SFO	$\wedge$C	$\wedge$BC	$\vee$C	$\vee$BC	$\neg$C
1-CARD	✓	✓	✓	?	?	?	?	?
1-PBC	✓	✓	✓	?	?	?	?	?

Conditioning is just replacing a variable by 0 or 1

Forgetting one variable can be computed in polytime:

$$\exists x. \left(ax + \sum_{j=0}^{n} a_j l_j \geqslant k \right) \equiv \left(\sum_{j=0}^{n} a_j l_j \geqslant k - a \right)$$

Transforming a single pseudo-Boolean constraint

	CD	FO	SFO	$\wedge$C	$\wedge$BC	$\vee$C	$\vee$BC	$\neg$C
1-CARD	✓	✓	✓	?	?	?	?	✓
1-PBC	✓	✓	✓	?	?	?	?	✓

Conditioning is just replacing a variable by 0 or 1

Forgetting one variable can be computed in polytime:

$$\exists x. \left(ax + \sum_{j=0}^{n} a_j l_j \geqslant k \right) \equiv \left(\sum_{j=0}^{n} a_j l_j \geqslant k - a \right)$$

Negation is computable in polytime: $\neg \left(\sum_{j=1}^{n} a_j l_j \geqslant n \right) \equiv \sum_{j=1}^{n} a_j l_j < n$

Transforming a single pseudo-Boolean constraint

	CD	FO	SFO	$\wedge$C	$\wedge$BC	$\vee$C	$\vee$BC	$\neg$C
1-CARD	✓	✓	✓	•	•	•	•	✓
1-PBC	✓	✓	✓	•	•	•	•	✓

Conditioning is just replacing a variable by 0 or 1

Forgetting one variable can be computed in polytime:

$$\exists x. \left(ax + \sum_{j=0}^{n} a_j l_j \geqslant k \right) \equiv \left(\sum_{j=0}^{n} a_j l_j \geqslant k - a \right)$$

Negation is computable in polytime: $\neg \left(\sum_{j=1}^{n} a_j l_j \geqslant n \right) \equiv \sum_{j=1}^{n} a_j l_j < n$

Conjunctions and disjunctions are not computable in general since both languages are not expressive enough

One constraint is not enough

In general, a propositional formula may require more than a single pseudo-Boolean constraint to be expressed

One constraint is not enough

In general, a propositional formula may require more than a single pseudo-Boolean constraint to be expressed

$$\phi = x \oplus y$$

One constraint is not enough

In general, a propositional formula may require more than a single pseudo-Boolean constraint to be expressed

$$\phi = x \oplus y$$

We need to use a conjunction *of a set of constraints: PBC or CARD*

PBC and CARD as compilation languages

Succinctness of PBC and CARD

Succinctness of PBC and CARD

$CARD \not\leqslant PBC$ because translating $\kappa = kx + \sum_{j=1}^{2k} x_j \geqslant k$ into CARD requires clauses, and there is an exponential number of them

Succinctness of PBC and CARD

$CARD \nleqslant PBC$ because translating $\kappa = kx + \sum_{j=1}^{2k} x_j \geqslant k$ into CARD requires clauses, and there is an exponential number of them

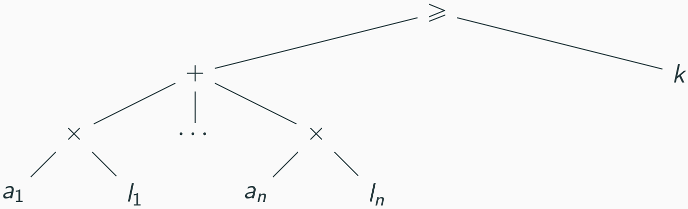
$$\bigwedge_{\substack{I \subset 1..2k \\ |I| = k+1}} \left(x \vee \bigvee_{i \in I} x_i \right)$$

Succinctness of PBC and CARD

$NNF \leqslant PBC$ because a formula from PBC can be seen as an arithmetic circuit, and such a circuit can be translated into a polysize NNF circuit [Vollmer, 1999]

Succinctness of PBC and CARD

$NNF \leqslant PBC$ because a formula from PBC can be seen as an arithmetic circuit, and such a circuit can be translated into a polysize NNF circuit [Vollmer, 1999]



Succinctness of PBC and CARD

$PBC \nleqslant IP$ because $\bigvee_{i=1}^{n}(x_i \wedge y_i)$ requires an exponential number of constraints to be expressed

Succinctness of PBC and CARD

$PBC \nleqslant OBDD_<$ because parity function can only be represented in PBC with clauses

Succinctness of PBC and CARD

$PBC \nleqslant OBDD_{<}$ because parity function can only be represented in PBC with clauses

$\phi = x \oplus y \oplus z$

Succinctness of PBC and CARD

$PBC \nleqslant OBDD_<$ because parity function can only be represented in PBC with clauses

$$
\begin{aligned}
\phi &= x \oplus y \oplus z \\
&\equiv (x \vee y \vee z) \wedge (x \vee \neg y \vee \neg z) \wedge (\neg x \vee y \vee \neg z) \wedge (\neg x \vee \neg y \vee z)
\end{aligned}
$$

Succinctness of PBC and CARD

$PBC \nleqslant OBDD_<$ because parity function can only be represented in PBC with clauses

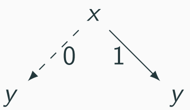
$$\begin{aligned}\phi &= x \oplus y \oplus z \\ &\equiv (x \vee y \vee z) \wedge (x \vee \neg y \vee \neg z) \wedge (\neg x \vee y \vee \neg z) \wedge (\neg x \vee \neg y \vee z)\end{aligned}$$

x

Succinctness of PBC and CARD

$PBC \not\leqslant OBDD_<$ because parity function can only be represented in PBC with clauses

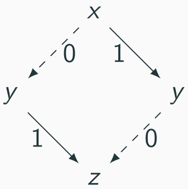
$$
\begin{aligned}
\phi &= x \oplus y \oplus z \\
&\equiv (x \vee y \vee z) \wedge (x \vee \neg y \vee \neg z) \wedge (\neg x \vee y \vee \neg z) \wedge (\neg x \vee \neg y \vee z)
\end{aligned}
$$



Succinctness of PBC and CARD

$PBC \not\leqslant OBDD_{<}$ because parity function can only be represented in PBC with clauses

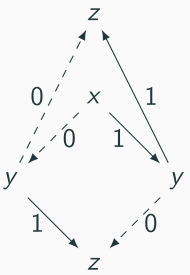
$$
\begin{aligned}
\phi &= x \oplus y \oplus z \\
&\equiv (x \vee y \vee z) \wedge (x \vee \neg y \vee \neg z) \wedge (\neg x \vee y \vee \neg z) \wedge (\neg x \vee \neg y \vee z)
\end{aligned}
$$



Succinctness of PBC and CARD

$PBC \not\leqslant OBDD_<$ because parity function can only be represented in PBC with clauses

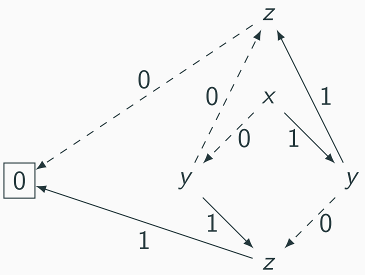
$$\begin{aligned}\phi &= x \oplus y \oplus z \\ &\equiv (x \vee y \vee z) \wedge (x \vee \neg y \vee \neg z) \wedge (\neg x \vee y \vee \neg z) \wedge (\neg x \vee \neg y \vee z)\end{aligned}$$



Succinctness of PBC and CARD

$PBC \not\leqslant OBDD_{<}$ because parity function can only be represented in PBC with clauses

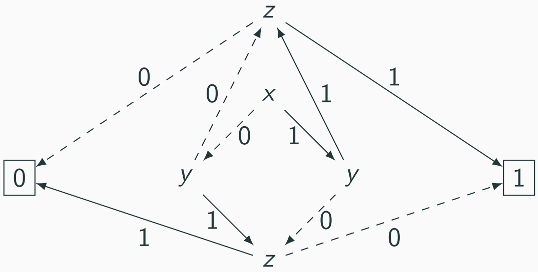
$$\phi = x \oplus y \oplus z$$
$$\equiv (x \vee y \vee z) \wedge (x \vee \neg y \vee \neg z) \wedge (\neg x \vee y \vee \neg z) \wedge (\neg x \vee \neg y \vee z)$$



Succinctness of PBC and CARD

$PBC \not\leqslant OBDD_<$ because parity function can only be represented in PBC with clauses

$$
\begin{aligned}
\phi &= x \oplus y \oplus z \\
&\equiv (x \vee y \vee z) \wedge (x \vee \neg y \vee \neg z) \wedge (\neg x \vee y \vee \neg z) \wedge (\neg x \vee \neg y \vee z)
\end{aligned}
$$



Succinctness of PBC and CARD

	$CARD$	PBC
NNF	?	?
$DNNF$	?	?
$d-DNNF$	?	?
$sd-DNNF$	?	?
$FBDD$	?	?
$OBDD$	?	?
$OBDD_<$	?	?
DNF	?	?
CNF	?	?
PI	?	?
IP	?	?
$MODS$	?	?
$CARD$	$\geqslant$	$\geqslant$
PBC	?	$\geqslant$

	$CARD$	PBC
NNF	?	?
$DNNF$	?	?
$d-DNNF$	?	?
$sd-DNNF$	?	?
$FBDD$	?	?
$OBDD$	?	?
$OBDD_<$	?	?
DNF	?	?
CNF	?	?
PI	?	?
IP	?	?
$MODS$	?	?
$CARD$	$\leqslant$	?
PBC	$\leqslant$	$\leqslant$

▷ Proven ▷ [Dixon, 2004] ▷ Transitivity

Succinctness of PBC and CARD

	$CARD$	PBC
NNF	?	?
$DNNF$	?	?
$d - DNNF$	?	?
$sd - DNNF$	?	?
$FBDD$	?	?
$OBDD$	?	?
$OBDD_<$	?	?
DNF	?	?
CNF	?	?
PI	?	?
IP	?	?
$MODS$	?	?
$CARD$	$\geqslant$	$\geqslant$
PBC	$\not\geqslant$	$\geqslant$

	$CARD$	PBC
NNF	?	?
$DNNF$	?	?
$d - DNNF$	?	?
$sd - DNNF$	?	?
$FBDD$	?	?
$OBDD$	?	?
$OBDD_<$	?	?
DNF	?	?
CNF	?	?
PI	?	?
IP	?	?
$MODS$	?	?
$CARD$	$\leqslant$	$\not\leqslant$
PBC	$\leqslant$	$\leqslant$

▷ Proven ▷ [Dixon, 2004] ▷ Transitivity

Succinctness of PBC and CARD

	CARD	*PBC*
NNF	?	?
DNNF	?	?
$d-DNNF$	?	?
$sd-DNNF$	?	?
FBDD	?	?
OBDD	?	?
$OBDD_<$	?	?
DNF	?	?
CNF	$\geqslant$	$\geqslant$
PI	?	?
IP	?	?
MODS	?	?
CARD	$\geqslant$	$\geqslant$
PBC	$\ngeqslant$	$\geqslant$

	CARD	*PBC*
NNF	?	?
DNNF	?	?
$d-DNNF$	?	?
$sd-DNNF$	?	?
FBDD	?	?
OBDD	?	?
$OBDD_<$	?	?
DNF	?	?
CNF	?	?
PI	?	?
IP	?	?
MODS	?	?
CARD	$\leqslant$	$\nleqslant$
PBC	$\leqslant$	$\leqslant$

▷ Proven ▷ [Dixon, 2004] ▷ Transitivity

Succinctness of PBC and CARD

	CARD	*PBC*
NNF	?	?
DNNF	?	?
d − DNNF	?	?
sd − DNNF	?	?
FBDD	?	?
OBDD	?	?
$OBDD_<$	?	?
DNF	?	?
CNF	$\geqslant$	$\geqslant$
PI	$\geqslant$	$\geqslant$
IP	?	?
MODS	$\geqslant$	$\geqslant$
CARD	$\geqslant$	$\geqslant$
PBC	$\not\geqslant$	$\geqslant$

	CARD	*PBC*
NNF	?	?
DNNF	?	?
d − DNNF	?	?
sd − DNNF	?	?
FBDD	?	?
OBDD	?	?
$OBDD_<$	?	?
DNF	?	?
CNF	?	?
PI	?	?
IP	?	?
MODS	?	?
CARD	$\leqslant$	$\not\leqslant$
PBC	$\leqslant$	$\leqslant$

▷ Proven ▷ [Dixon, 2004] ▷ Transitivity

Succinctness of PBC and CARD

	$CARD$	PBC
NNF	?	?
$DNNF$	?	?
$d-DNNF$	?	?
$sd-DNNF$	?	?
$FBDD$	?	?
$OBDD$	?	?
$OBDD_<$	?	?
DNF	?	?
CNF	$\geqslant$	$\geqslant$
PI	$\geqslant$	$\geqslant$
IP	?	?
$MODS$	$\geqslant$	$\geqslant$
$CARD$	$\geqslant$	$\geqslant$
PBC	$\ngeqslant$	$\geqslant$

	$CARD$	PBC
NNF	?	?
$DNNF$	?	?
$d-DNNF$	?	?
$sd-DNNF$	?	?
$FBDD$	?	?
$OBDD$	?	?
$OBDD_<$	?	?
DNF	?	?
CNF	$\nleqslant$	$\nleqslant$
PI	?	?
IP	?	?
$MODS$	?	?
$CARD$	$\leqslant$	$\nleqslant$
PBC	$\leqslant$	$\leqslant$

▷ Proven ▷ [Dixon, 2004] ▷ Transitivity

Succinctness of PBC and CARD

	$CARD$	PBC
NNF	?	?
$DNNF$	?	?
$d-DNNF$	?	?
$sd-DNNF$	?	?
$FBDD$	?	?
$OBDD$	?	?
$OBDD_<$	?	?
DNF	?	?
CNF	$\geqslant$	$\geqslant$
PI	$\geqslant$	$\geqslant$
IP	?	?
$MODS$	$\geqslant$	$\geqslant$
$CARD$	$\geqslant$	$\geqslant$
PBC	$\not\geqslant$	$\geqslant$

	$CARD$	PBC
NNF	?	?
$DNNF$	$\not\leqslant$	$\not\leqslant$
$d-DNNF$	$\not\leqslant$	$\not\leqslant$
$sd-DNNF$	$\not\leqslant$	$\not\leqslant$
$FBDD$	$\not\leqslant$	$\not\leqslant$
$OBDD$	$\not\leqslant$	$\not\leqslant$
$OBDD_<$	$\not\leqslant$	$\not\leqslant$
DNF	$\not\leqslant$	$\not\leqslant$
CNF	$\not\leqslant$	$\not\leqslant$
PI	$\not\leqslant$	$\not\leqslant$
IP	$\not\leqslant$	$\not\leqslant$
$MODS$	$\not\leqslant$	$\not\leqslant$
$CARD$	$\leqslant$	$\not\leqslant$
PBC	$\leqslant$	$\leqslant$

▷ Proven ▷ [Dixon, 2004] ▷ Transitivity

Succinctness of PBC and CARD

	CARD	*PBC*
NNF	?	?
DNNF	?	?
$d-DNNF$	?	?
$sd-DNNF$	?	?
FBDD	?	?
OBDD	?	?
$OBDD_<$	?	?
DNF	?	?
CNF	$\geqslant$	$\geqslant$
PI	$\geqslant$	$\geqslant$
IP	?	?
MODS	$\geqslant$	$\geqslant$
CARD	$\geqslant$	$\geqslant$
PBC	$\ngeqslant$	$\geqslant$

	CARD	*PBC*
NNF	$\leqslant$	$\leqslant$
DNNF	$\nleqslant$	$\nleqslant$
$d-DNNF$	$\nleqslant$	$\nleqslant$
$sd-DNNF$	$\nleqslant$	$\nleqslant$
FBDD	$\nleqslant$	$\nleqslant$
OBDD	$\nleqslant$	$\nleqslant$
$OBDD_<$	$\nleqslant$	$\nleqslant$
DNF	$\nleqslant$	$\nleqslant$
CNF	$\nleqslant$	$\nleqslant$
PI	$\nleqslant$	$\nleqslant$
IP	$\nleqslant$	$\nleqslant$
MODS	$\nleqslant$	$\nleqslant$
CARD	$\leqslant$	$\nleqslant$
PBC	$\leqslant$	$\leqslant$

▷ Proven ▷ [Dixon, 2004] ▷ Transitivity

Succinctness of PBC and CARD

	$CARD$	PBC
NNF	?	?
$DNNF$	?	?
$d-DNNF$	?	?
$sd-DNNF$	?	?
$FBDD$	?	?
$OBDD$	?	?
$OBDD_<$	$\not\geqslant$	$\not\geqslant$
DNF	?	?
CNF	$\geqslant$	$\geqslant$
PI	$\geqslant$	$\geqslant$
IP	?	?
$MODS$	$\geqslant$	$\geqslant$
$CARD$	$\geqslant$	$\geqslant$
PBC	$\not\geqslant$	$\geqslant$

	$CARD$	PBC
NNF	$\leqslant$	$\leqslant$
$DNNF$	$\not\leqslant$	$\not\leqslant$
$d-DNNF$	$\not\leqslant$	$\not\leqslant$
$sd-DNNF$	$\not\leqslant$	$\not\leqslant$
$FBDD$	$\not\leqslant$	$\not\leqslant$
$OBDD$	$\not\leqslant$	$\not\leqslant$
$OBDD_<$	$\not\leqslant$	$\not\leqslant$
DNF	$\not\leqslant$	$\not\leqslant$
CNF	$\not\leqslant$	$\not\leqslant$
PI	$\not\leqslant$	$\not\leqslant$
IP	$\not\leqslant$	$\not\leqslant$
$MODS$	$\not\leqslant$	$\not\leqslant$
$CARD$	$\leqslant$	$\not\leqslant$
PBC	$\leqslant$	$\leqslant$

▷ Proven ▷ [Dixon, 2004] ▷ Transitivity

Succinctness of PBC and CARD

	$CARD$	PBC
NNF	$\not\geqslant$	$\not\geqslant$
$DNNF$	$\not\geqslant$	$\not\geqslant$
$d-DNNF$	$\not\geqslant$	$\not\geqslant$
$sd-DNNF$	$\not\geqslant$	$\not\geqslant$
$FBDD$	$\not\geqslant$	$\not\geqslant$
$OBDD$	$\not\geqslant$	$\not\geqslant$
$OBDD_<$	$\not\geqslant$	$\not\geqslant$
DNF	?	?
CNF	$\geqslant$	$\geqslant$
PI	$\geqslant$	$\geqslant$
IP	?	?
$MODS$	$\geqslant$	$\geqslant$
$CARD$	$\geqslant$	$\geqslant$
PBC	$\not\geqslant$	$\geqslant$

	$CARD$	PBC
NNF	$\leqslant$	$\leqslant$
$DNNF$	$\not\leqslant$	$\not\leqslant$
$d-DNNF$	$\not\leqslant$	$\not\leqslant$
$sd-DNNF$	$\not\leqslant$	$\not\leqslant$
$FBDD$	$\not\leqslant$	$\not\leqslant$
$OBDD$	$\not\leqslant$	$\not\leqslant$
$OBDD_<$	$\not\leqslant$	$\not\leqslant$
DNF	$\not\leqslant$	$\not\leqslant$
CNF	$\not\leqslant$	$\not\leqslant$
PI	$\not\leqslant$	$\not\leqslant$
IP	$\not\leqslant$	$\not\leqslant$
$MODS$	$\not\leqslant$	$\not\leqslant$
$CARD$	$\leqslant$	$\not\leqslant$
PBC	$\leqslant$	$\leqslant$

▷ Proven ▷ [Dixon, 2004] ▷ Transitivity

Succinctness of PBC and CARD

	$CARD$	PBC
NNF	$\not\geqslant$	$\not\geqslant$
$DNNF$	$\not\geqslant$	$\not\geqslant$
$d-DNNF$	$\not\geqslant$	$\not\geqslant$
$sd-DNNF$	$\not\geqslant$	$\not\geqslant$
$FBDD$	$\not\geqslant$	$\not\geqslant$
$OBDD$	$\not\geqslant$	$\not\geqslant$
$OBDD_<$	$\not\geqslant$	$\not\geqslant$
DNF	?	?
CNF	$\geqslant$	$\geqslant$
PI	$\geqslant$	$\geqslant$
IP	$\not\geqslant$	$\not\geqslant$
$MODS$	$\geqslant$	$\geqslant$
$CARD$	$\geqslant$	$\geqslant$
PBC	$\not\geqslant$	$\geqslant$

	$CARD$	PBC
NNF	$\leqslant$	$\leqslant$
$DNNF$	$\not\leqslant$	$\not\leqslant$
$d-DNNF$	$\not\leqslant$	$\not\leqslant$
$sd-DNNF$	$\not\leqslant$	$\not\leqslant$
$FBDD$	$\not\leqslant$	$\not\leqslant$
$OBDD$	$\not\leqslant$	$\not\leqslant$
$OBDD_<$	$\not\leqslant$	$\not\leqslant$
DNF	$\not\leqslant$	$\not\leqslant$
CNF	$\not\leqslant$	$\not\leqslant$
PI	$\not\leqslant$	$\not\leqslant$
IP	$\not\leqslant$	$\not\leqslant$
$MODS$	$\not\leqslant$	$\not\leqslant$
$CARD$	$\leqslant$	$\not\leqslant$
PBC	$\leqslant$	$\leqslant$

▷ Proven ▷ [Dixon, 2004] ▷ Transitivity

Succinctness of PBC and CARD

	$CARD$	PBC
NNF	$\not\geqslant$	$\not\geqslant$
$DNNF$	$\not\geqslant$	$\not\geqslant$
$d-DNNF$	$\not\geqslant$	$\not\geqslant$
$sd-DNNF$	$\not\geqslant$	$\not\geqslant$
$FBDD$	$\not\geqslant$	$\not\geqslant$
$OBDD$	$\not\geqslant$	$\not\geqslant$
$OBDD_<$	$\not\geqslant$	$\not\geqslant$
DNF	$\not\geqslant$	$\not\geqslant$
CNF	$\geqslant$	$\geqslant$
PI	$\geqslant$	$\geqslant$
IP	$\not\geqslant$	$\not\geqslant$
$MODS$	$\geqslant$	$\geqslant$
$CARD$	$\geqslant$	$\geqslant$
PBC	$\not\geqslant$	$\geqslant$

	$CARD$	PBC
NNF	$\leqslant$	$\leqslant$
$DNNF$	$\not\leqslant$	$\not\leqslant$
$d-DNNF$	$\not\leqslant$	$\not\leqslant$
$sd-DNNF$	$\not\leqslant$	$\not\leqslant$
$FBDD$	$\not\leqslant$	$\not\leqslant$
$OBDD$	$\not\leqslant$	$\not\leqslant$
$OBDD_<$	$\not\leqslant$	$\not\leqslant$
DNF	$\not\leqslant$	$\not\leqslant$
CNF	$\not\leqslant$	$\not\leqslant$
PI	$\not\leqslant$	$\not\leqslant$
IP	$\not\leqslant$	$\not\leqslant$
$MODS$	$\not\leqslant$	$\not\leqslant$
$CARD$	$\leqslant$	$\not\leqslant$
PBC	$\leqslant$	$\leqslant$

▷ Proven ▷ [Dixon, 2004] ▷ Transitivity

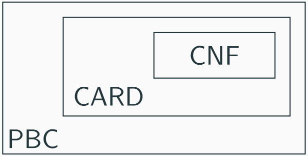
Querying a set of constraints

	CO	VA	CE	IM	EQ	SE	CT	ME
CNF	○	✓	○	✓	○	○	○	○
CARD	?	?	?	?	?	?	?	?
PBC	?	?	?	?	?	?	?	?

Querying a set of constraints

	CO	VA	CE	IM	EQ	SE	CT	ME
CNF	$\circ$	$\checkmark$	$\circ$	$\checkmark$	$\circ$	$\circ$	$\circ$	$\circ$
CARD	$\circ$	?	$\circ$	?	$\circ$	$\circ$	$\circ$	$\circ$
PBC	$\circ$	?	$\circ$	?	$\circ$	$\circ$	$\circ$	$\circ$

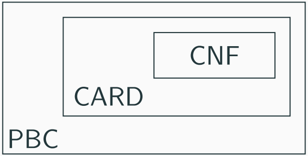
By functional reduction, NP-hard problems for CNF are NP-hard for CARD and PBC



Querying a set of constraints

	CO	VA	CE	IM	EQ	SE	CT	ME
CNF	○	✓	○	✓	○	○	○	○
CARD	○	✓	○	✓	○	○	○	○
PBC	○	✓	○	✓	○	○	○	○

By functional reduction, NP-hard problems for CNF are NP-hard for CARD and PBC



Properties of pseudo-Boolean constraints give the other results

Transforming a set of constraints

	CD	FO	SFO	$\wedge$C	$\wedge$BC	$\vee$C	$\vee$BC	$\neg$C
CNF	✓	○	✓	✓	✓	●	✓	●
CARD	?	?	?	?	?	?	?	?
PBC	?	?	?	?	?	?	?	?

Transforming a set of constraints

	CD	FO	SFO	$\wedge$C	$\wedge$BC	$\vee$C	$\vee$BC	$\neg$C
CNF	✓	○	✓	✓	✓	●	✓	●
CARD	✓	○	?	✓	?	?	?	?
PBC	✓	○	?	✓	?	?	?	?

Arguments for CNF can be applied to PBC and CARD

Transforming a set of constraints

	CD	FO	SFO	$\wedge$C	$\wedge$BC	$\vee$C	$\vee$BC	$\neg$C
CNF	✓	○	✓	✓	✓	●	✓	●
CARD	✓	○	?	✓	✓	?	?	?
PBC	✓	○	?	✓	✓	?	?	?

Arguments for CNF can be applied to PBC and CARD

Transforming a set of constraints

	CD	FO	SFO	$\wedge$C	$\wedge$BC	$\vee$C	$\vee$BC	$\neg$C
CNF	✓	○	✓	✓	✓	●	✓	●
CARD	✓	○	?	✓	✓	?	●	?
PBC	✓	○	?	✓	✓	?	●	?

Arguments for CNF can be applied to PBC and CARD

$\sum_{i=1}^{2n} x_i \neq n$ can only be expressed with an exponential number of clauses, and

$$\sum_{i=1}^{2n} x_i \neq n \equiv \left(\sum_{i=1}^{2n} x_i < n\right) \vee \left(\sum_{i=1}^{2n} x_i > n\right)$$

Transforming a set of constraints

	CD	FO	SFO	$\wedge$C	$\wedge$BC	$\vee$C	$\vee$BC	$\neg$C
CNF	✓	○	✓	✓	✓	●	✓	●
CARD	✓	○	?	✓	✓	●	●	?
PBC	✓	○	?	✓	✓	●	●	?

Arguments for CNF can be applied to PBC and CARD

$\sum_{i=1}^{2n} x_i \neq n$ can only be expressed with an exponential number of clauses, and

$$\sum_{i=1}^{2n} x_i \neq n \equiv \left(\sum_{i=1}^{2n} x_i < n\right) \vee \left(\sum_{i=1}^{2n} x_i > n\right)$$

Transforming a set of constraints

	CD	FO	SFO	$\wedge$C	$\wedge$BC	$\vee$C	$\vee$BC	$\neg$C
CNF	✓	○	✓	✓	✓	●	✓	●
CARD	✓	○	?	✓	✓	●	●	●
PBC	✓	○	?	✓	✓	●	●	●

Arguments for CNF can be applied to PBC and CARD

$\sum_{i=1}^{2n} x_i \neq n$ can only be expressed with an exponential number of clauses, and

$$\sum_{i=1}^{2n} x_i \neq n \equiv \left(\sum_{i=1}^{2n} x_i < n\right) \vee \left(\sum_{i=1}^{2n} x_i > n\right)$$

Transforming a set of constraints

	CD	FO	SFO	$\wedge$C	$\wedge$BC	$\vee$C	$\vee$BC	$\neg$C
CNF	✓	○	✓	✓	✓	●	✓	●
CARD	✓	○	?	✓	✓	●	●	●
PBC	✓	○	?	✓	✓	●	●	●

Arguments for CNF can be applied to PBC and CARD

$\sum_{i=1}^{2n} x_i \neq n$ can only be expressed with an exponential number of clauses, and

$$\sum_{i=1}^{2n} x_i \neq n \equiv \left(\sum_{i=1}^{2n} x_i < n \right) \vee \left(\sum_{i=1}^{2n} x_i > n \right)$$

Recap of the results

Succinctness: $PBC < CARD < CNF$

Recap of the results

Succinctness: $PBC < CARD < CNF$

Queries:

	CO	VA	CE	IM	EQ	SE	CT	ME
CNF	$\circ$	$\checkmark$	$\circ$	$\checkmark$	$\circ$	$\circ$	$\circ$	$\circ$
CARD	$\circ$	$\checkmark$	$\circ$	$\checkmark$	$\circ$	$\circ$	$\circ$	$\circ$
PBC	$\circ$	$\checkmark$	$\circ$	$\checkmark$	$\circ$	$\circ$	$\circ$	$\circ$

Recap of the results

Succinctness: $PBC < CARD < CNF$

Queries:

	CO	VA	CE	IM	EQ	SE	CT	ME
CNF	∘	✓	∘	✓	∘	∘	∘	∘
CARD	∘	✓	∘	✓	∘	∘	∘	∘
PBC	∘	✓	∘	✓	∘	∘	∘	∘

Transformations:

	CD	FO	SFO	$\wedge$C	$\wedge$BC	$\vee$C	$\vee$BC	$\neg$C
CNF	✓	∘	✓	✓	✓	•	✓	•
CARD	✓	∘	?	✓	✓	•	•	•
PBC	✓	∘	?	✓	✓	•	•	•

Recap of the results

Succinctness: $PBC < CARD < CNF$

Queries:

	CO	VA	CE	IM	EQ	SE	CT	ME
CNF	$\circ$	$\checkmark$	$\circ$	$\checkmark$	$\circ$	$\circ$	$\circ$	$\circ$
CARD	$\circ$	$\checkmark$	$\circ$	$\checkmark$	$\circ$	$\circ$	$\circ$	$\circ$
PBC	$\circ$	$\checkmark$	$\circ$	$\checkmark$	$\circ$	$\circ$	$\circ$	$\circ$

Transformations:

	CD	FO	SFO	$\wedge$C	$\wedge$BC	$\vee$C	$\vee$BC	$\neg$C
CNF	$\checkmark$	$\circ$	$\checkmark$	$\checkmark$	$\checkmark$	$\bullet$	$\checkmark$	$\bullet$
CARD	$\checkmark$	$\circ$	?	$\checkmark$	$\checkmark$	$\bullet$	$\bullet$	$\bullet$
PBC	$\checkmark$	$\circ$	?	$\checkmark$	$\checkmark$	$\bullet$	$\bullet$	$\bullet$

Compared to CNF, PBC and CARD are strictly more succinct, but the same queries and less transformations can be computed in polytime

What’s next?

Open question: SFO?

Is there any polytime algorithm to compute the forgetting of a variable x in a PBC formula K?

Open question: SFO?

Is there any polytime algorithm to compute the forgetting of a variable x in a PBC formula K?

$$\exists x.K \equiv (K|x) \vee (K|\overline{x})$$

Singleton Forgetting (SFO): an example

Let us consider:

- $\kappa_1 = x + a + b \geqslant 2$
- $\kappa_2 = \overline{x} + 2c + 2d \geqslant 3$

Singleton Forgetting (SFO): an example

Let us consider:

- $\kappa_1 = x + a + b \geqslant 2$
- $\kappa_2 = \overline{x} + 2c + 2d \geqslant 3$

Then:

$$\frac{x + a + b \geqslant 2 \qquad \overline{x} + 2c + 2d \geqslant 3}{a + b + 2c + 2d \geqslant 4}$$

Singleton Forgetting (SFO): an example

Let us consider:

- $\kappa_1 = x + a + b \geqslant 2$
- $\kappa_2 = \overline{x} + 2c + 2d \geqslant 3$

Then:

$$\frac{x + a + b \geqslant 2 \qquad \overline{x} + 2c + 2d \geqslant 3}{a + b + 2c + 2d \geqslant 4}$$

But:

$$\exists x.(\kappa_1 \wedge \kappa_2) \equiv a + b + c + d \geqslant 3$$

Singleton Forgetting (SFO): an example

Let us consider:

- $\kappa_1 = x + a + b \geqslant 2$
- $\kappa_2 = \overline{x} + 2c + 2d \geqslant 3 \equiv \overline{x} + c + d \geqslant 2$

Then:

$$\frac{x + a + b \geqslant 2 \qquad \overline{x} + 2c + 2d \geqslant 3}{a + b + 2c + 2d \geqslant 4}$$

But:

$$\exists x.(\kappa_1 \wedge \kappa_2) \equiv a + b + c + d \geqslant 3$$

Singleton Forgetting (SFO): an example

Let us consider:

- $\kappa_1 = x + a + b \geqslant 2$
- $\kappa_2 = \overline{x} + 2c + 2d \geqslant 3 \equiv \overline{x} + c + d \geqslant 2$

Then:

$$\frac{x + a + b \geqslant 2 \qquad \overline{x} + c + d \geqslant 2}{a + b + c + d \geqslant 3}$$

But:

$$\exists x.(\kappa_1 \wedge \kappa_2) \equiv a + b + c + d \geqslant 3$$

Singleton Forgetting (SFO): an other example

Let us consider:

- $\kappa_1 = x + a + b \geqslant 2$
- $\kappa_3 = \overline{x} + c + 2d + 3e \geqslant 5$

Singleton Forgetting (SFO): an other example

Let us consider:

- $\kappa_1 = x + a + b \geqslant 2$
- $\kappa_3 = \overline{x} + c + 2d + 3e \geqslant 5$

Then:

$$\frac{x + a + b \geqslant 2 \qquad \overline{x} + c + 2d + 3e \geqslant 5}{a + b + c + 2d + 3e \geqslant 6}$$

Singleton Forgetting (SFO): an other example

Let us consider:

- $\kappa_1 = x + a + b \geqslant 2$
- $\kappa_3 = \overline{x} + c + 2d + 3e \geqslant 5$

Then:

$$\frac{x + a + b \geqslant 2 \qquad \overline{x} + c + 2d + 3e \geqslant 5}{a + b + c + 2d + 3e \geqslant 6}$$

But:

$$\exists x.(\kappa_1 \wedge \kappa_3) \equiv 2a + 2b + c + 3d + 4e \geqslant 9$$

Singleton Forgetting (SFO): an other example

Let us consider:

- $\kappa_1 = x + a + b \geqslant 2 \equiv x + 2a + 2b \geqslant 3$
- $\kappa_3 = \overline{x} + c + 2d + 3e \geqslant 5$

Then:

$$\frac{x + a + b \geqslant 2 \qquad \overline{x} + c + 2d + 3e \geqslant 5}{a + b + c + 2d + 3e \geqslant 6}$$

But:

$$\exists x.(\kappa_1 \wedge \kappa_3) \equiv 2a + 2b + c + 3d + 4e \geqslant 9$$

Singleton Forgetting (SFO): an other example

Let us consider:

- $\kappa_1 = x + a + b \geqslant 2 \equiv x + 2a + 2b \geqslant 3$
- $\kappa_3 = \overline{x} + c + 2d + 3e \geqslant 5$

Then:

$$\frac{x + 2a + 2b \geqslant 3 \qquad \overline{x} + c + 2d + 3e \geqslant 5}{2a + 2b + c + 2d + 3e \geqslant 7}$$

But:

$$\exists x.(\kappa_1 \wedge \kappa_3) \equiv 2a + 2b + c + 3d + 4e \geqslant 9$$

Singleton Forgetting (SFO): an other example

Let us consider:

- $\kappa_1 = x + a + b \geqslant 2 \equiv x + 2a + 2b \geqslant 3$
- $\kappa_3 = \overline{x} + c + 2d + 3e \geqslant 5 \Rrightarrow \overline{x} + c + 3d + 4e \geqslant 7$

Then:

$$\frac{x + 2a + 2b \geqslant 3 \qquad \overline{x} + c + 2d + 3e \geqslant 5}{2a + 2b + c + 2d + 3e \geqslant 7}$$

But:

$$\exists x.(\kappa_1 \wedge \kappa_3) \equiv 2a + 2b + c + 3d + 4e \geqslant 9$$

Singleton Forgetting (SFO): an other example

Let us consider:

- $\kappa_1 = x + a + b \geqslant 2 \equiv x + 2a + 2b \geqslant 3$
- $\kappa_3 = \overline{x} + c + 2d + 3e \geqslant 5 \dashv \overline{x} + c + 3d + 4e \geqslant 7$

Then:

$$\frac{x + 2a + 2b \geqslant 3 \qquad \overline{x} + c + 3d + 4e \geqslant 7}{2a + 2b + c + 3d + 4e \geqslant 9}$$

But:

$$\exists x.(\kappa_1 \wedge \kappa_3) \equiv 2a + 2b + c + 3d + 4e \geqslant 9$$

Compile!

PBC and CARD are not good compilation languages

Compile!

PBC and CARD are not good compilation languages

We need to define sub-languages of PBC which enable to perform more queries and transformations

Compile!

PBC and CARD are not good compilation languages

We need to define sub-languages of PBC which enable to perform more queries and transformations

As an example, we can define pseudo-Boolean prime implicants and implicates

- κ is an IP-PBC of K iff $\kappa \models K$ and, if $\kappa' \models K$ and $\kappa \models \kappa'$, then $\kappa' \equiv \kappa$
- κ is a PI-PBC of K iff $K \models \kappa$ and, if $K \models \kappa'$ and $\kappa' \models \kappa$, then $\kappa' \equiv \kappa$

Compile!

PBC and CARD are not good compilation languages

We need to define sub-languages of PBC which enable to perform more queries and transformations

As an example, we can define pseudo-Boolean prime implicants and implicates

- κ is an IP-PBC of K iff $\kappa \models K$ and, if $\kappa' \models K$ and $\kappa \models \kappa'$, then $\kappa' \equiv \kappa$
- κ is a PI-PBC of K iff $K \models \kappa$ and, if $K \models \kappa'$ and $\kappa' \models \kappa$, then $\kappa' \equiv \kappa$

Canonical form

Can we define a canonical form for pseudo-Boolean constraints?

Canonical form

Can we define a canonical form for pseudo-Boolean constraints?

We need a unique way to write the constraint, which must be as simple as possible

Canonical form

Can we define a canonical form for pseudo-Boolean constraints?

We need a unique way to write the constraint, which must be as simple as possible

$$9w + 6x + 3y + z \geqslant 11$$

Canonical form

Can we define a canonical form for pseudo-Boolean constraints?

We need a unique way to write the constraint, which must be as simple as possible

$$9w + 6x + 3y + z \geqslant 11 \equiv \qquad 9w + 6x + 3y \geqslant 11$$

Canonical form

Can we define a canonical form for pseudo-Boolean constraints?

> *We need a unique way to write the constraint, which must be as simple as possible*

$$
\begin{aligned}
9w + 6x + 3y + z \geqslant 11 &\equiv \quad 9w + 6x + 3y \geqslant 11 \\
&\equiv \quad 9w + 6x + 3y \geqslant 12
\end{aligned}
$$

Canonical form

Can we define a canonical form for pseudo-Boolean constraints?

We need a unique way to write the constraint, which must be as simple as possible

$$\begin{aligned} 9w + 6x + 3y + z \geqslant 11 &\equiv & 9w + 6x + 3y \geqslant 11 \\ &\equiv & 9w + 6x + 3y \geqslant 12 \\ &\equiv & 3w + 2x + y \geqslant 4 \end{aligned}$$

Canonical form

Can we define a canonical form for pseudo-Boolean constraints?

We need a unique way to write the constraint, which must be as simple as possible

$$
\begin{aligned}
9w + 6x + 3y + z \geqslant 11 &\equiv & 9w + 6x + 3y \geqslant 11 \\
&\equiv & 9w + 6x + 3y \geqslant 12 \\
&\equiv & 3w + 2x + y \geqslant 4 \\
&\equiv & 2w + x + y \geqslant 3
\end{aligned}
$$

Canonical form

Can we define a canonical form for pseudo-Boolean constraints?

We need a unique way to write the constraint, which must be as simple as possible

$$
\begin{aligned}
9w + 6x + 3y + z \geqslant 11 &\equiv & 9w + 6x + 3y \geqslant 11 \\
&\equiv & 9w + 6x + 3y \geqslant 12 \\
&\equiv & 3w + 2x + y \geqslant 4 \\
&\equiv & 2w + x + y \geqslant 3 \\
&\equiv & (w \geqslant 1) \wedge (x + y \geqslant 1)
\end{aligned}
$$

Canonical form

Can we define a canonical form for pseudo-Boolean constraints?

> *We need a unique way to write the constraint, which must be as simple as possible*

$$
\begin{aligned}
9w + 6x + 3y + z \geqslant 11 \equiv {} & 9w + 6x + 3y \geqslant 11 \\
\equiv {} & 9w + 6x + 3y \geqslant 12 \\
\equiv {} & 3w + 2x + y \geqslant 4 \\
\equiv {} & 2w + x + y \geqslant 3 \\
\equiv {} & (w \geqslant 1) \wedge (x + y \geqslant 1) \\
\equiv {} & w \wedge (x \vee y)
\end{aligned}
$$

Canonical form: a sketch of definition

A pseudo-Boolean constraint $\kappa = \sum_{j=1}^{n} a_j l_j \geqslant k$ is in canonical form iff it satisfies the following properties:

Canonical form: a sketch of definition

A pseudo-Boolean constraint $\kappa = \sum_{j=1}^{n} a_j l_j \geqslant k$ is in canonical form iff it satisfies the following properties:

- $\forall x \in Var(\kappa)$, x appears in κ only once

Canonical form: a sketch of definition

A pseudo-Boolean constraint $\kappa = \sum_{j=1}^{n} a_j l_j \geqslant k$ is in canonical form iff it satisfies the following properties:

- $\forall x \in Var(\kappa)$, x appears in κ only once
- $\forall i, i' \in 1..n, i \leqslant i' \iff a_i \geqslant a_{i'}$, i.e. literals are sorted by descending weights

Canonical form: a sketch of definition

A pseudo-Boolean constraint $\kappa = \sum_{j=1}^{n} a_j l_j \geqslant k$ is in canonical form iff it satisfies the following properties:

- $\forall x \in Var(\kappa)$, x appears in κ only once
- $\forall i, i' \in 1..n, i \leqslant i' \iff a_i \geqslant a_{i'}$, i.e. literals are sorted by descending weights
- there exists a model of κ which is a model of $\sum_{j=1}^{n} a_j l_j = k$

Canonical form: a sketch of definition

A pseudo-Boolean constraint $\kappa = \sum_{j=1}^{n} a_j l_j \geqslant k$ is in canonical form iff it satisfies the following properties:

- $\forall x \in Var(\kappa)$, x appears in κ only once
- $\forall i, i' \in 1..n, i \leqslant i' \iff a_i \geqslant a_{i'}$, i.e. literals are sorted by descending weights
- there exists a model of κ which is a model of $\sum_{j=1}^{n} a_j l_j = k$
- $\forall j \in 1..n$, there exists no other constraint $\kappa' \equiv \kappa$ such that l_j has a weight $a'_j < a_j$ in κ'

Canonical form: a sketch of definition

A pseudo-Boolean constraint $\kappa = \sum_{j=1}^{n} a_j l_j \geqslant k$ is in canonical form iff it satisfies the following properties:

- $\forall x \in Var(\kappa)$, x appears in κ only once
- $\forall i, i' \in 1..n, i \leqslant i' \iff a_i \geqslant a_{i'}$, i.e. literals are sorted by descending weights
- there exists a model of κ which is a model of $\sum_{j=1}^{n} a_j l_j = k$
- $\forall j \in 1..n$, there exists no other constraint $\kappa' \equiv \kappa$ such that l_j has a weight $a'_j < a_j$ in κ'

For a given constraint κ, is there always a unique "canonical form" of κ?

Canonical form: a sketch of definition

A pseudo-Boolean constraint $\kappa = \sum_{j=1}^{n} a_j l_j \geqslant k$ is in canonical form iff it satisfies the following properties:

- $\forall x \in Var(\kappa)$, x appears in κ only once
- $\forall i, i' \in 1..n, i \leqslant i' \iff a_i \geqslant a_{i'}$, i.e. literals are sorted by descending weights
- there exists a model of κ which is a model of $\sum_{j=1}^{n} a_j l_j = k$
- $\forall j \in 1..n$, there exists no other constraint $\kappa' \equiv \kappa$ such that l_j has a weight $a'_j < a_j$ in κ'

For a given constraint κ, is there always a unique "canonical form" of κ?

Implement an efficient pseudo-Boolean solver

Implement an efficient pseudo-Boolean solver

- Investigate why pseudo-Boolean solvers are not as efficient in practice as they should theoretically be

Implement an efficient pseudo-Boolean solver

- Investigate why pseudo-Boolean solvers are not as efficient in practice as they should theoretically be

- Use arbitrary precision only when needed

Implement an efficient pseudo-Boolean solver

- Investigate why pseudo-Boolean solvers are not as efficient in practice as they should theoretically be
- Use arbitrary precision only when needed
- Find a better solution than *reduction* for learning

Implement an efficient pseudo-Boolean solver

- Investigate why pseudo-Boolean solvers are not as efficient in practice as they should theoretically be
- Use arbitrary precision only when needed
- Find a better solution than *reduction* for learning
- Find a solution to the fact that generalized resolution is not implication-complete

Conclusion

Conclusion

Recap:

- Pseudo-Boolean constraints properties
- Pseudo-Boolean constraints as a compilation language

Conclusion

Recap:

- Pseudo-Boolean constraints properties
- Pseudo-Boolean constraints as a compilation language

Future works:

- Get a better understanding of pseudo-Boolean constraints
- Define PBC sublanguages for compilation
- Implement an efficient solver using PBC and CARD

Pseudo-Boolean Constraints: Reasoning and Compilation

Romain Wallon *(Advisors: Daniel Le Berre, Pierre Marquis, Stefan Mengel)*

September 11, 2017

CRIL - U. Artois & CNRS

UNIVERSITÉ D'ARTOIS